POLITECNICO DI TORINO

DEPARTMENT OF MANAGEMENT AND PRODUCTION ENGINEERING

Course in Engineering and Management

Master's Degree Thesis

The Development and Quality Control of the Testing Process of an Applicative for the Management of Credit Recovery Positions

Mentor:

Prof. Ing. Domenico Augusto Maisano

Candidate:

Giovanni Campisi

Graduation session March 2023

ABSTRACT

The paper follows the evolution of the PGR - Upload project undertaken during my internship at Alten Italia. The subject of the discussion is the testing process carried out for a banking institution's application in which the focus is the implementation of the Upload functionality, an integration to the PGR application used for the management of retail customers' credits with reference to non-performing loans, through which the user will be able to carry out massive actions, centralising the process and the monitoring of requests.
The project consists of 179 tests, including both front-end and back-end tests.

At the end of the test execution phase, data on the outcome of the tests are collected in company reports and analysed using appropriate KPIs.

The test phase was then subjected to quality control methodologies, in an attempt to support the test executions through control charts and sampling plans so as to provide a clear and comprehensive view of the work performed as a tester in an outsourced engineering company.

Abbreviations and acronyms:

QM = Quality Management
AFU = Functional Analysis
BE = Back-End
FE = Front-End
NPL = Non-Performing Loan
QA = Quality Assurance
ST = System Test
TB = Testbook
TF = Test Factory
SSP = Single Sampling Plan
DSP = Double Sampling Plan
CC = Control Chart

Contents

Introduction to the thesis

The purpose of this thesis is to describe and analyse the system test project of PGR-Upload, undertaken during my internship as a tester at Alten Italia.

Portale Gestione Retail (PGR) is a banking software that supports our client, a banking institution, in the management of non-performing retail's credit positions; the project under examination implies the integration and implementation of new functions to this portal, including a new applicative through which to perform massive actions on different positions.

The management of credit recovery positions is an issue of great importance for banks, which must continuously strive to improve the effectiveness and efficiency of the processes involved. In this context, system tests represent an important phase in the software lifecycle, as they make it possible to verify and validate the correct functioning of the banking application, identifying any defects and guaranteeing the reliability of the system.

To this end, the new applicative will be described in detail in order to provide a basis for understanding the entire testing process, from the analysis of the new functions to the creation of the different test cases and their execution, which aim at verifying the correct implementation of the expected functionality.

In order to provide a more complete view, moreover, results obtained during the execution phase will then be analysed through project reports drafted during the project, and through the calculation of some critical KPIs in the testing sector.

Once the system test process has been described, we will proceed with the application of statistical control methods, in particular with the identification of the most appropriate acceptance-sampling plans and control charts for the context under examination. The impact of these control methods on the quality of the system test will then be analysed, assessing the effectiveness of the tools adopted and any criticalities that may have emerged.

Finally, the conclusions of the thesis will be presented, with an analysis of the results obtained. The thesis thus aims to make a contribution to the understanding and optimisation of a system test of a banking software, proposing statistical control methods in support of the entire process.

The above is described throughout the paper, which is organised in seven chapters and supplemented with two appendices, Appendix A and Appendix B.

If we get into more specifics, the first chapter presents the general context and work environment in which companies operate in today's market; shedding light on the factors that led to the rise of the Quality Assurance concept and the consequent importance of the testing process. Starting from this context, the testing process is described in detail, presenting its main objectives, the different

techniques that can be adopted and the related risks. This is leveraged to describe the context in which Alten Italia operates.

In the second chapter we get into the heart of the paper. In fact, the main inputs that led our client to the development of the PGR applicative are initially analysed, and to do so an introduction to the world of credit recovery is necessary to understand its importance.
Moreover, the new applicative functionalities are described in detail, so as to show the process by which we go from the individual functionality to the construction of the test case that aims at verifying it works properly.
Some examples of identified test cases are also reported, both for front-end and back-end of the system.

In the third and fourth chapters, an overview of test execution is provided.
Specifically, in the third chapter it is done by analysing its progress and results through the data contained in the project reports and, in doing so, we will also refer to some of the detected anomalies.
In Chapter 4, on the other hand, the results are analysed through the computation of some critical KPIs, following a brief theoretical introduction of the topic.

In the fifth and sixth chapters, the previously mentioned statistical control methods are applied to the project under consideration.
In Chapter 5 acceptance control methods are applied, in particular the construction of a single and a double sampling plan are proposed as alternatives to performing a large number of test cases.
In Chapter 6, on the other hand, a p-control chart is proposed as a method to monitor and control the entire testing process.

The seventh and final chapter sets out the due conclusions, based on the results obtained in the development of the paper. In addition, an outlook of future implementations and developments related to the PGR worksite is briefly presented, presenting three new projects that will keep me involved in the coming months.

The two appendices are dedicated to auxiliary material that contributes to the completion of the analysis presented.

Appendix A describes the acceptance control process in detail, complementing the analysis conducted in the fifth chapter.

Appendix B, on the other hand, comes in to support the analysis carried out in Chapter 6, introducing the concept and the application of control charts.

Chapter 1. Introduction to a new market's reality, the concepts of Quality Assurance and Testing

The first chapter presents an overview of the work environment and of the new realities that companies are facing in the market today, which have brought the importance of the Quality Assurance concept to the fore and which, in return, has made the testing process one of the main focuses. Starting from the context in which it developed and the needs that led to the birth of testing, the different methodologies that lead to the fulfilment of the objectives that this process sets out to achieve, the types of tests and their techniques, as well as the risks related are presented.

It is in this context that ALTEN Group, a European leader in consulting for advanced engineering and ICT technologies, operates.

1.1 What is Testing?

The current market environment is characterized by high levels of dynamism and complexity, by ever-increasing competition, in which companies must maintain a flexibility capable of coping with ever more frequent changes to survive [1].
The need to implement such changes arises from several factors; one of the most significant is the fact that companies have shifted their focus from pure production to the customer, making the satisfaction of customer needs a priority. This focus on the variety and variability of customer needs must inevitably lead to the adoption of organisational and operational structures capable of adapting to the changes taking place, both in the market and in the technological world.

Technological innovation is changing the world we live in. In this context, the digitisation of businesses is an unstoppable phenomenon which entails a revolution so profound that it radically changes the working reality of many companies and the way they conduct and organise their work [2].

So, this emerging global scenario demands from companies in all sectors a dynamism and openness towards innovation and new technological directions that is without precedent, as well as the ability to deal with an increasingly demanding and technologically advanced customer.
The key role assumed by software and the increasing spread of IoT technologies have raised the level of market expectations and further consolidated the relationship between customer experience and business results, making the quality, availability, and reliability of software applications one of the key factors directly impacting the future of companies [A].

Software systems are an integral part of everyday life, from business applications (e.g., banking) to consumer products (e.g., cars). Most people have had experiences with software that did not work as expected. Software that does not work properly can cause many problems, including loss of money, time, business reputation.

It is precisely in this context that the recent emergence of digital testing takes place.

Software testing is a way to assess the quality of software and to reduce the risk of software failure in production.
A common misperception is that testing consists only of executing tests, that is, running the software and checking the results. Software testing is a process that includes many different activities and running tests is only one of these activities. The testing process also includes activities such as planning, analysing, designing, and implementing tests, and reporting on test progress and results.
Another common misperception about testing is that it focuses entirely on the verification of requirements, user stories or other specifications. Testing also includes verifying that the system meets specific requirements, that is, that the system meets the needs of the user and other stakeholders when placed in its operational environment.

For each project, testing objectives may include:
- Preventing defects through the evaluation of work products such as requirements, user story, design, and code
- Checking whether all specified requirements have been met
- Checking whether the test object is complete and validating whether it works as users and other stakeholders expect
- Increase confidence in the quality level of the test object
- Find failures and defects, so as to reduce the level of risk of inadequate software quality
- Provide sufficient information to stakeholders to enable them to make informed decisions, especially regarding the quality level of the test object
- Comply with contractual, legal, or regulatory requirements or standards, and/or verify the compliance of the test object with such requirements or standards

Rigorous testing of components and systems, including their associated documentation, can help reduce the risk of failures occurring in production. When defects are detected, and subsequently corrected, this contributes to the quality of the components or systems.
In IT, it is quite common to detect situations where software and systems are released into operation and downstream of that release failures are observed or stakeholder requirements are not met, due to the presence of defects. However, by using appropriate testing techniques, one is able to reduce the frequency of such problematic releases, especially when these techniques are applied with the appropriate level of expertise in testing, at the appropriate levels of testing, and at the right points in the software development life cycle [3].

1.1.1 The seven principles of testing

Over the past 50 years, various principles on testing have been suggested to offer general guidelines for testing.

Principle 1 - Testing shows the presence of defects, not their absence
Testing can show the presence of defects, but it cannot prove their absence. Testing reduces the likelihood of undetected defects remaining in the software, but even if no defects are found, testing is not proof of correctness, i.e., the absence of defects.

Principle 2 - Exhaustive testing is impossible
Testing everything, all combinations of inputs and preconditions, is not practicable except for trivial cases. Risk analysis, testing techniques and priorities should be used to focus the testing effort, rather than attempting exhaustive testing.

Principle 3 - Early Testing Saves Time and Money
To find defects early, it is necessary to start testing activities as early as possible in the software development life cycle. Early testing is sometimes referred to as 'shift left'. Performing testing early in the software development lifecycle helps reduce or eliminate costly subsequent changes.

Principle 4 - Defects tend to form clusters
A small number of modules usually contain most of the defects discovered during testing prior to release or are responsible for most of the failures. Clusters of expected defects and those actually observed in testing or in operation are an important input to a risk analysis used to focus testing effort.

Principle 5 - Beware of the pesticide paradox
If the same tests are repeated over and over again, presumably these tests will not detect any new defects. In order to detect new defects, existing tests and test data may have to be modified, and new tests may have to be designed: tests are no longer effective in detecting defects, just as pesticides are no longer effective in killing insects after prolonged use.

Principle 6 - Testing is context-dependent
Testing is performed differently in different contexts.

Principle 7 - Absence of Faults is a False Belief
Some organisations expect testers to be able to perform all possible tests and find all possible defects, but Principles 2 and 1 tell us, respectively, that this is impossible. Furthermore, it is a false belief to expect that merely finding and fixing a large number of defects will guarantee the success of a system. For example, performing thorough testing of all specified requirements and correcting all detected defects may still not be sufficient to prevent the release of a system that is difficult to use, does not meet users' needs and expectations, or is inferior to other competitive systems.

1.1.2 Testing within the Software Development Life Cycle

A software development lifecycle model describes the activities performed in each phase of a software development project, and how they relate to each other in a logical and chronological manner.

There are several software development lifecycle models, each of which requires different approaches to testing.
An important part of a tester's role is to be familiar with software development lifecycle models, so that appropriate testing activities can be carried out. In fact, the two concepts are closely linked: for every development activity, there is a corresponding testing activity; analysis and test design for a given level of testing begin during the corresponding development activity.
In addition, testers are often involved in work product reviews, so that they actively participate in the definition and refinement of requirements and design.

Software development life cycle models are classified into sequential models and iterative or incremental models.
The former, also called Waterfall models, describe the software development process as a linear and sequential flow of activities. This means that any stage in the development process should begin when the previous stage has been completed; consequently, in this model, testing activities occur when all other development activities have been completed.
The latter, also referred to as the V-model, implement the principle of “testing anticipato” (early testing), whereby the testing process is integrated within the development process. These models are also referred to as incremental because the activities of defining requirements, designing, developing, and testing a system are performed incrementally: software functionality increases during development.
This is another major difference from sequential models, which release software that contains the complete set of functionalities.

Related to each development level is a test level, defined as an instance of the test process that contains a group of tasks that are managed together.
Each test level is characterised by specific attributes, such as objectives, test basis, test object, typical failures and defects, approaches and responsibilities.
The test levels are:

- Component testing: also called unit or module testing, it focuses on components that can be tested separately. Its main objectives include reducing risk, verifying that the functional and non-functional behaviour of the component corresponds to what was designed, increasing its quality, and detecting defects to prevent them in higher levels of testing. Examples of test bases used are detailed design and component specifications; while test objects are components and codes to which typical defects are related such as functionality or incorrect code, i.e., different from what is described in the design specifications. Component testing is usually performed by the developer who wrote the code.

A practical example is now given, in an e-commerce software application, unit tests can be used to verify the proper functioning of individual functions, such as calculating the cart total, adding a product to the cart, and checkout.

- Integration testing: it focuses on the interactions between components or systems. The main objectives include goals very similar to those of component testing.
  The test bases used are Use Case, Sequence diagram and Workflow; while typical test objects are databases, subsystems or interfaces involving some typical defects such as incorrect or missing data, interface misalignments, communication errors between systems or between components. In the case of component integration tests, the responsibility lies with the developers; in contrast, it is the responsibility of the testers in the case of system integration tests. In any case, testers should be familiar with the systems architecture.
  A practical example may be in testing a web application, integration tests can be used to verify that different functions of the application, such as login, navigation, and checkout, work properly together. For example, the test can verify that the user can login, that the navigation page is correctly displayed, and that the user can complete the checkout securely and reliably.

- System testing: it focuses on the behaviour and capabilities of an entire system or product with the same objectives as in the two previous cases. System testing produces information that is crucial for stakeholders to make release decisions. Typical test bases are software and system requirements specifications, user and system manuals, and risk analysis reports. Typical objects of this category of tests are applications, hardware and software systems, operating systems that may include defects such as miscalculations or errors in functional or non-functional behaviour. It is typically carried out by independent testers.
  An example may be in testing the online payment system, the system test verifies that the online payment system is secure, reliable, and that transactions are processed correctly; the test can verify that the user can make a purchase, that the system correctly processes the transaction, and that the confirmation is sent to the user.

- Acceptance testing: it focuses on the behaviour and capabilities of a complete system or product with the aim of establishing confidence in its quality and verifying its correct functioning.
  Among the most common acceptance tests, we recognise the User Acceptance Test (UAT), which focuses on validating the suitability of the system for use by the intended users, either in a real or simulated environment, with the aim of assessing that users can use the system to meet their needs with a minimum of difficulty, cost, and risk.
  In general, it uses business processes, user or business requirements, and system requirements as the basis for testing. As far as test objects are concerned, these include business processes, configuration data, production data or reports that may involve defects

such as business rules that have not been implemented correctly or systems that do not meet requirements.

1.1.3 Different test types and techniques

Each of the test levels just described can be associated with a test type; test type means a group of test activities with the purpose of testing specific characteristics of a software system, or part of a system, based on specific test objectives.
Specifically, the indicated levels of testing can be classified into functional and non-functional testing:

- Functional testing of a system includes tests that evaluate the functions the system is expected to perform. Functional requirements may be described in work products such as business requirements specifications, epic, user story, use case or functional specifications, or they may not be documented. Functions represent "what" the system should do.
- Non-functional testing of a system evaluates the characteristics of software and systems, such as usability, performance efficiency or security. It can be considered as the testing of "how well" the system performs. Contrary to common misperceptions, non-functional testing can, and often should, be done at all levels of testing and as early as possible. Late discovery of non-functional defects can be extremely dangerous to the success of a project.

In addition, for each of the levels and types of tests identified, we point out the existence of three main testing techniques: black-box, white box and techniques based on experience. The purpose of a testing technique, including those being discussed in this section, is to help identify test conditions, test cases, and test data. The choice of test techniques to be used depends on a number of factors, which include the complexity of the component/system, regulatory standards, contractual or customer requirements, tester skills, timing and budget.

Black-box testing techniques, also called behavioural or behaviour-based techniques, are based on an analysis of the appropriate test base such as formal requirements documents, specifications, use cases, user stories or business processes. These techniques are applicable to both functional and non-functional testing. Black-box testing techniques focus on the inputs and outputs of the test object, without reference to its internal structure.
As real-case example, in mobile application testing black-box testing involves checking the functionality of the mobile application without knowing the source code. For example, the tester can verify that all buttons work properly, that notifications are sent in a timely manner, and that the application is stable during use.

White-box testing techniques, also called structural or structure-based techniques, are based on an analysis of the architecture, detailed design, internal structure, or code of the test object. Unlike black-box testing techniques, white-box testing techniques focus on the structure and internal processing of the test object.

As an example, in software testing, white-box testing consists of verifying the quality of software code; the tester can verify that the code is well written, meets requirements, and is error-free. In a system testing, white-box testing consists of verifying the quality of the system from a technical perspective; the tester may verify that the system meets performance requirements and that it is stable.

Experience-based testing techniques leverage the experience of developers, testers, and users to design, implement, and execute tests. These techniques are often combined with black-box and white-box testing techniques.

Black-box, white-box and experience-based testing in turn encompass several techniques.

Black box testing includes:

1. Equivalence partitioning: it divides data into partitions based on the assumption that all elements in a given partition are processed equally. Partitions identify valid values, which should be accepted by the component or system, and invalid values, which should be rejected by the system. Two partitions are thus identified: Valid Equivalence Partition and Invalid Equivalence Partition. Each partition can be divided into sub-partitions, but each value can belong to only one of them. To achieve 100% coverage with this technique, test cases must cover all identified partitions, including invalid partitions. Coverage is measured as the number of equivalence partitions tested by at least one value, divided by the total number of equivalence partitions identified, usually expressed as a percentage. Equivalence partitioning is applicable to all levels of testing.
2. Boundary value analysis (BVA): is an extension of equivalence partitioning but can be used only when the partition is ordered, consisting of numeric or sequential data. The minimum and maximum values, or first and last values, of a partition are its boundary values and are more likely to misbehave than internal values. Boundary value coverage is measured as the number of boundary values tested, divided by the total number of identified boundary values, normally expressed as a percentage.
3. Decision table testing: it is a viable approach to describing complex business rules that a system must implement. In creating decision tables, the tester identifies the conditions, then the inputs, and the resulting actions, the outputs, of the system. These elements form the rows of the table, usually with the conditions at the top and the actions at the bottom of the table. Each column corresponds to a decision rule, defined by a unique combination of conditions that determine the execution of the actions associated with that rule. The values of the conditions and actions are normally given as Boolean values or discrete values.
   The typical standard minimum coverage for decision table testing is to have at least one test case for each decision rule in the table.
4. State transition testing: it is used when a system responds differently to the same input depending on current conditions and previous history. The previous history of the system is called the state, while an event is an occurrence that can result in a transition. A state transition

diagram or state transition table is a representation of the system's states and events that are causing transitions between states. A state transitions table shows all valid and potentially invalid transitions between states, as well as events, and actions resulting from valid transitions. Coverage is commonly measured as the number of states or transitions identified and tested divided by the total number of states or transitions identified in the test object, usually expressed as a percentage.

5. Use Case testing: Tests can be derived from use cases, which are a specific way of designing interactions with software elements and report requirements for software functions. Each use case specifies the behaviour of one or more actors and how they interact with each other.
   Tests are designed to practice the behaviours defined in the use case. Coverage can be measured as the number of use case behaviours tested, divided by the total number of use case behaviours, usually expressed as a percentage.

White-box testing include:

1. Statement coverage: it exercises the potentially executable instructions in the code; it aims to find a set of tests such that every instruction within the software is guaranteed to be executed at least once. Coverage is measured as the number of instructions executed by the tests divided by the total number of executable instructions in the test object, usually expressed as a percentage
2. Decision or branch coverage: Decision testing exercises the decisions in the code and tests the code that is executed based on the decision outcomes. To do this, test cases follow the control flow that starts from a decision point. Coverage is measured as the number of decision outcomes performed by the tests divided by the total number of decision outcomes in the test object, usually expressed as a percentage.

Experience-based testing include:

1. Error Guessing: Error guessing is a technique used to anticipate the occurrence of errors, defects, and failures, based on the tester's knowledge, which takes into consideration, for example, how the application has worked in the past and what types of errors tend to occur.
   One approach to the error guessing technique is to create a list of possible errors, defects, and failures, and to design the tests that will generate those failures and the defects that caused them.
2. Exploratory testing: are non-predefined tests that are designed, executed, recorded, and evaluated during test execution. Test results are used to learn more about the component or system, and to create tests related to areas that may need more attention.
3. Checklist-based testing: testers design, implement, and execute tests to cover the test conditions reported in a checklist. This checklist can just have been created, or an already existing one can also be used.

1.1.4 Overview of testing process activities

As mentioned before, one of the main misinterpretations about testing is that it just consists of execution; instead, the testing process comprises several activities, and test execution is just one of them.

There is no single universal software testing process, but there are common sets of testing activities without which testing will be less likely to achieve its stated goals. These sets of testing activities constitute a testing process.

Various factors influence the testing process, including, for example, product and project risks, internal and external standards required by the customer; operational constraints such as budget, resources, time, complexity, contractual and regulatory requirements; or even the software development lifecycle model.

A testing process consists of the following main groups of activities [Figure 1.1]: test planning, test monitoring and control, test analysis, test design, test execution, test completion.

The individual activities consist of several sub-activities, which may vary according to the project in question. In general, however, they can be described as follows.

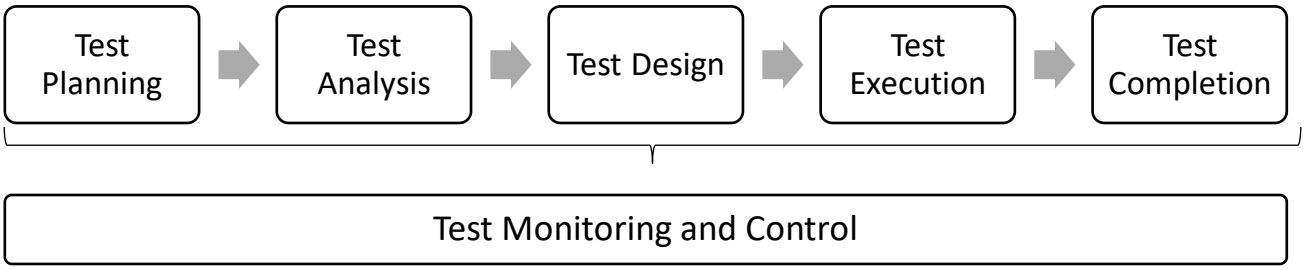

Figure 1.1 Testing process activities

Test planning defines the objectives of the test and the approach to meet those objectives, within the constraints imposed by the context; for example, by specifying suitable test techniques and tasks and scheduling tests to meet a deadline.

The product of the planning activity is the so-called Test Plan; moreover, since it is an ongoing or continuous activity that is performed throughout the product life cycle, feedback from the test activities should be used to recognise changes in risks and modify the planning appropriately.

The next phase is test monitoring and control, which involves the continuous comparison of actual progress against planned progress, using the test monitoring metrics defined in the Test Plan, and the implementation of the necessary actions to meet the targets.

Test progress against the planned is communicated to stakeholders in the Test Progress Reports, including any deviations from the plan. In addition to the Test Progress Report, the Test Manager also prepares the so-called Test Summary Report at the end of the test activities basing on the last progress report and any other relevant information.

During test analysis, the test base is analysed to identify testable functionality, in other words to determine “what to test”.
Among the main sub-activities of test analysis, it is important to highlight some of them, such as:

- Analysing the appropriate test base for the considered test level, e.g., requirement specifications (whether business or functional), information about the component or system implementation such as code or interfaces.
- Evaluate the test basis and test elements to identify any defects, such as contradictions, ambiguities, or inaccuracies.
- Identify the functionalities and sets of functionalities to be tested.

The design phase answers the question "how to test". Test cases are designed and prioritised, the data required to support the test conditions are identified and the test environment is designed.

During the implementation of the tests, it is ensured that everything is set up correctly to guarantee the execution of the tests: the test suites are created according to the schedule in order to achieve efficient test execution, the test environment is created and, finally, the test cases are loaded into the test environment.

Reaching the execution phase, the test suites are executed according to the planned schedule. The execution phase involves executing tests manually or by means of execution tools, but also comparing actual results with expected results, reporting defects based on observed failures, re-executing test activities following corrective action of an anomaly.

The last stage of the testing process is the test completion, during which all test activity data is collected to reinforce the experience and knowledge gained during the process. In addition, the closure of all defect reports is verified, change requests are created for any unresolved defects, and the Test Summary Report is drawn up to be submitted to stakeholders [3].

At this point in the discussion, it is evident that there are two leading figures within the testing process: the Test Manager and the Tester.
The Test Manager has overall responsibility for the testing process and for successful leadership of testing activities. The main activity envisaged by this role is to plan testing activities considering the context, objectives and risks involved; in addition, the Test Manager draws up and updates the Test Plan, as well as the Test Progress Report and the Test Summary Report.
As a manager, his or her role also goes beyond project activities, being responsible for developing the skills and careers of testers.

The tester, on the other hand, plays a key role from the test analysis phase onwards, although he or she may also be involved in the drafting of the Test Plan. The main activities performed by the tester are to design, implement and execute test cases.

The tester's activities will be analysed more in detail in Chapter 2, since it is the role I held during the internship I did at Alten Italia Test Factory.

1.2 Testing as a risk mitigation activity

Risk is the possibility that an event will have negative consequences in the future. The level of risk is determined by the probability of the event and the impact, i.e., the damage, of the event.
In the context of testing, risk plays a special role since testing itself is a risk mitigation activity, used to reduce the probability and impact of a negative event.

Risk is classified into product risk and project risk.

Product risk consists of the possibility that a work product, such as a specification, component, system, or test, may fail to meet the legitimate needs of its users and/or stakeholders.
Examples of product risk include, for example, the incorrect execution of expected functions in a software programme.

Project risk consists of situations that, should they occur, could have a negative effect on the project's ability to achieve its objectives.
Project risks can be classified into:

- Project problems, such as delays in the release or in the completion of activities.
- Organisational problems, such as conflicts or problems within the staff concerned.
- Political problems, such as inadequate communication of test results by testers.
- Technical problems, which may include a test environment that is not ready on time or requirements that are not defined well enough.
- Supplier problems, such as the failure to release a product/service necessary for the activities to continue.

A risk-based testing approach offers valuable opportunities to reduce product risk levels. It consists of product risk analysis, which includes identifying product risks and assessing the likelihood and impact of each risk. The resulting product risk information is used to guide the entire testing process, and thus test planning, specification, test case preparation and execution, and test monitoring and control.
Analysing product risk early contributes to the success of a project.

One of the objectives of testing is to find defects, so defects found during testing should be tracked from their discovery to their resolution in order to minimise the related risk.
This is the Defect Management process: an organisation should establish a process that includes a workflow and rules for classifying defects.

1.3 Quality in the test process

We may define quality in many ways. Most people have a conceptual understanding of quality as relating to one or more desirable characteristics that a product or service should possess. Although this conceptual understanding is certainly a useful starting point, we will give a more precise and useful definition. Quality has become one of the most important consumer decision factors in the selection among competing products and services. The phenomenon is widespread, regardless of whether the consumer is an individual, an industrial organization, a retail store, a bank or financial institution, or a military defence program. Consequently, understanding and improving quality are key factors leading to business success, growth, and enhanced competitiveness [4].
Quality can be defined as *"the degree to which the product/process or service is able to satisfy stated or implied needs".*

Emphasising how the focus on meeting quality requirements has become a competitive issue, we can state that Quality Assurance has a crucial role in ensuring an organisation's results.
The term Quality Assurance is often used when referring to testing. However, these aspects, although certainly related, are not the same thing. They are linked by a broader concept: Quality Management, which encompasses all activities that direct and control an organisation from a quality perspective.
Among other activities, Quality Management includes both Quality Assurance and Quality Control [Figure 1.2]. Quality Assurance is typically focused on compliance with appropriate processes in order to provide confidence that adequate levels of quality are achieved.
Quality Control involves various activities, including testing activities which are considered an integral part of overall software development and maintenance process, that support the achievement of adequate levels of quality.

Figure 1.2 QM activities

Since Quality Assurance is concerned with the correct execution of the entire process, we can deduce that it supports adequate testing, which as described in the previous paragraphs contributes to the achievement of quality in various ways.

1.4 The role of Alten Italia

At this point in the paper, it became clear how testing has become increasingly important in today's market environment. It plays a major role in ensuring the quality of a system, which translates into a satisfied customer and a good image of the firm.

Testing activities are often outsourced. We define outsourcing as the operation of shifting a transaction or activities to an external supplier through a long-term contract and involving the transfer of staff to the vendor [5]; it is a business organisation practice in which the steps of a company's production process are contracted out to an external company.

This explains the importance which the role of the consultant has gained today, and in particular how the IT consultancy firm takes shape. IT consultancy is a form, branch or sector of consultancy that consists of the professional services of one or more persons with expertise in the field of business information technology, who provide advice to a company on how best to use information technology (ICT) to achieve certain business goals or objectives.
The role of consultants and IT consultancy firms has risen in recent years due to the increasing importance of technology in businesses. Companies are turning to these firms for expertise in areas such as digital transformation, data analytics, and cybersecurity. The use of consultants also allows companies to bring in specialized skills on a project basis, rather than hiring full-time staff. Additionally, as technology continues to evolve rapidly, companies may find it more cost-effective to work with consultants who are up to date on the latest developments.

Outsourcing testing activities to a consultancy firm can be an effective way for companies to ensure the quality and reliability of their products and services. Consultancy firms typically have a team of experienced testers who are skilled in various testing methodologies and tools.
Outsourcing testing activities to a consultancy firm can also provide cost savings for companies, as they do not have to invest in expensive testing tools and resources. It also allows companies to focus on their core business activities and leave the testing activities to the experts [6].

The ALTEN Group, Europe's leading consultancy for advanced engineering and ICT technologies, operates in this context.
ALTEN is an international consulting company specialising in technological and engineering innovation founded in France in 1988 with 34,000 employees and listed on the Paris Stock Exchange since 1998.

ALTEN serves the market with services related to engineering, information technology, telecommunications, and life sciences, with several centres of excellence such as: digital, business intelligence, testing, ITSM, IoT, HW, embedded SW, quality, and CSV.
In the IT sector, ALTEN gives its support in different areas such as Banking, Insurance, Industry & Retail, Service Companies [B].

Alten's Test Factory, located in Turin, is involved in insurance and banking projects for major credit institutions, where the activities required are mainly of functional and quality assurance nature for banking-related software or applications.

During my internship I had the opportunity to work in the IT area for the Banking sector as a tester, and, in particular, within the Test Factory Credit Recovery team.

Chapter 2. Testing in the field of Credit Recovery, PGR – Upload project

In this chapter, the company project under discussion is presented. Being a project carried out within the credit recovery team, an introduction to the world of credit recovery is initially given to shed light on the requirements that led the client to the development of the PGR software. Then, the functionalities of the Upload integrative application are described in order to identify the elements that are to be tested and the test cases subject to execution; thus, describing in more detail the role of the tester in the testing process.

2.1 Introduction to Credit Recovery

For credit institutions, the quality of their portfolios is of utmost importance as it influences many aspects of normal management.

A "healthy" portfolio allows intermediaries greater freedom to grant credit, a higher level of operations and greater risk tolerance. Knowing how the behaviour of users can influence the creditworthiness attributed to them, precisely as a function of certain internal classifications within the bank, makes the relationship between the two parties more conscious.

The classification of credit situations allow creditors to classify their debtors and, depending on the seriousness of the debt situation and risk exposure, the bank will have to implement certain strategies to protect itself: set aside capital to cover exposures; change its credit granting policies; vary, making them more rigid, the economic conditions accompanying loan agreements; activate functional instruments to recover outstanding debts, and this comes with an obvious increase in costs.

Below are the most important classifications of impaired loans, provided by the Bank of Italy [7]:

1. Non-performing loan: when the bank has reason to believe that the loan is unrecoverable. Nonperforming loans, in fact, are commonly understood to be a status of persistent and not transitory asset and financial instability likely to hinder the recovery of the loan.

2. UTP, Unlikely To Pay: credit exposures for which the bank considers that the debtor is unlikely to meet its credit obligations in full, in principal and/or interest, without recourse to actions such as the enforcement of collateral.

3. Impaired past-due and/or overdrawn exposures: this category includes exposures, other than those classified as non-performing or likely to default, that are past due or in arrears for more than 90 days.

4. Forborne loans: loans, performing or impaired, that are subject to concessions by the bank. Forbearance measures are modifications to the original contractual terms of the line of credit that the bank grants to the corporate client. For example, the bank may grant the client a reduction in the interest rate of the loan or may arrange for an extension of the term of the loan.

Such forbearance measures may affect performing customers in financial difficulty or customers classified as impaired.

Credit recovery is the set of activities aimed at recovering a sum of money that a debtor owes to another party, namely the creditor. The purpose is to obtain what is due from the relationship between the parties. It is a phase of "credit management" which is the service of managing, administering and controlling a loan. In the banking sector, the amounts to be recovered from banks relate to credits and loans that have not been repaid by a customer.

To achieve this goal, many credits collection companies and agencies have emerged in Italy, offering protection and assistance on behalf of unsatisfied creditors.
Depending on the agreement made with the creditor, these can be "delegated" to collect the debt that remains in the name of a third party or become the actual owners of the outstanding debt to be collected. Credit recovery companies therefore can operate either under a general assignment or by acquiring the rights by assignment of the receivable.

This seemingly simple process is actually a path that starts with an initial set of reminder actions in an amicable manner that fall under what is known as extrajudicial credit recovery, the purpose of which is to invite the debtor to make a "spontaneous" payment.
Once attempts to obtain extrajudicial credit recovery have failed, it is necessary to proceed to the so-called credit recovery in court proceedings, in order to obtain an enforcement order, i.e., an order from the Judge for payment, addressed to the debtor. This title will then allow the creditor to proceed with enforcement on the debtor's property.
Within these two types of interventions, there are a number of possible actions, which help to delineate a specific path that can be described as follows:

- Phone Collection: a telephone reminder that aims to get in touch with the debtor and solicit in an amicable manner the payment of what is owed
- Written reminder: necessary to achieve greater effectiveness and to formalize the request for payment
- Home Collection: the intervention of specialized officials that often allows for the concretization of credit collection, reducing the distance between creditor and debtor.

During these stages of debt collection, it is possible to negotiate a debt repayment plan, that is, a deferred payment plan that allows the debtor to repay his debt with periodic payments. This solution can help when the debtor is in financial difficulty, allowing him or her to meet his or her debt and the charge of any additional interest accrued as a result of failure to meet payment terms. If a creditor and debtor agree on a repayment plan, it is essential that the agreed-upon deadlines are always monitored and met on time, payments should be verified by the creditor company, and any missed payments should be reported to the debtor immediately [C].

For many banks impaired loans and credit recovery activities represent a thorn in their side. And that is precisely why our client, who for reasons of confidentiality agreements will always be kept anonymous in the development of the work, has set up a system to recover unpaid instalments. This involves both in-house (Gestione Specialistica, GS) and outsourced management (Gestione Esternalizzata, GE):

- the former is usually used for the most important customers, who are therefore managed by figures within the bank.
- In the second case, instead, the bank relies on private companies, namely outsources.

About 80% of our client's customers are managed by outsourcers, who take a commission for each customer recovered.

2.2 PGR, Portale Gestione Retail: Management of impaired loan positions

In the previous section, we pointed out that impaired loan positions and the resulting credit collection activities play a major role in the banking world. Therefore, it is crucial for banks to identify and manage impaired positions in the best possible way.

Our client's solution was the implementation of a new portal called PGR, Portale Gestione Retail. PGR is an application used for the management of loans of retail customers, with reference to bad loans. The portal allows the recognition of impaired loan positions net of non-performing loans and enables their management through operational and monitoring functions.

The PGR project started in 2019, when the client opted for a complete revamp of its credit recovery application. The portal is still now evolving toward what was the initial business requirement through different implementations that bring more and more functions within the application.

Broadly speaking, we can consider PGR as a large registry of impaired positions, and through various functions several actions can be performed on them. A simple example is the "Avvia intervista" function, through which various questions will be asked to determine whether the individual is able to sustain the instalment payments against the bank and any other creditors, which results in one of the first actions defined as amicable in the previous section. At the end of the interview, the software will determine whether or not the stated instalment is sustainable [a].

As specified at the beginning of the paper, the subject of this discussion will be the implementation of the Upload functionality in the PGR portal, for which I have been able to follow the testing activities myself. The functionalities of the PGR portal are numerous and have already been tested

in previous implementations. Therefore, the following are some screenshots of the PGR layout without going into too much detail [Figure 2.1].

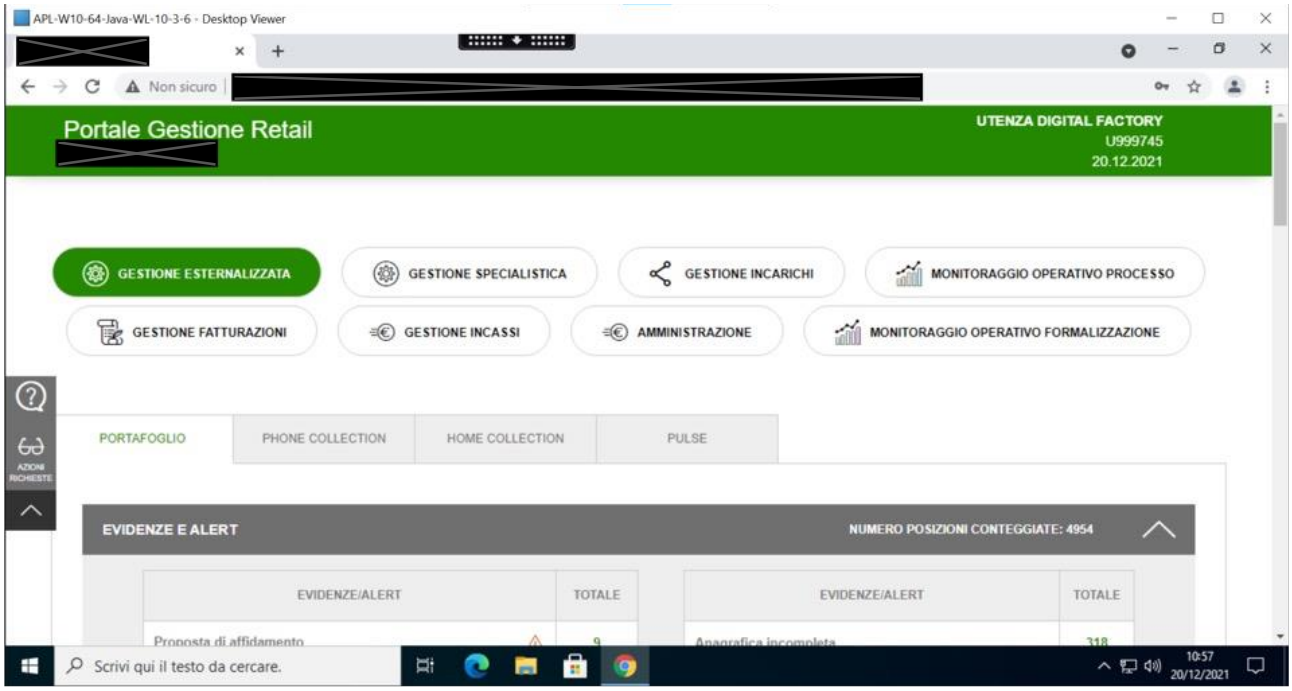

Figure 2.1 PGR - Homepage Layout. As soon as the PGR user logs in, this is the page to which he/she is redirected. Each tab has different functionalities, necessary for the bank to keep track of different impaired positions.

Please remember that PGR is a tool that support the bank in the management of impaired credit positions, as demonstrated by the first two Tabs on the Homepage, which allow the search and

processing of a position according to its assignment [Figure 2.2], GE or GI, in accordance with the corporate strategy presented in the previous paragraph.

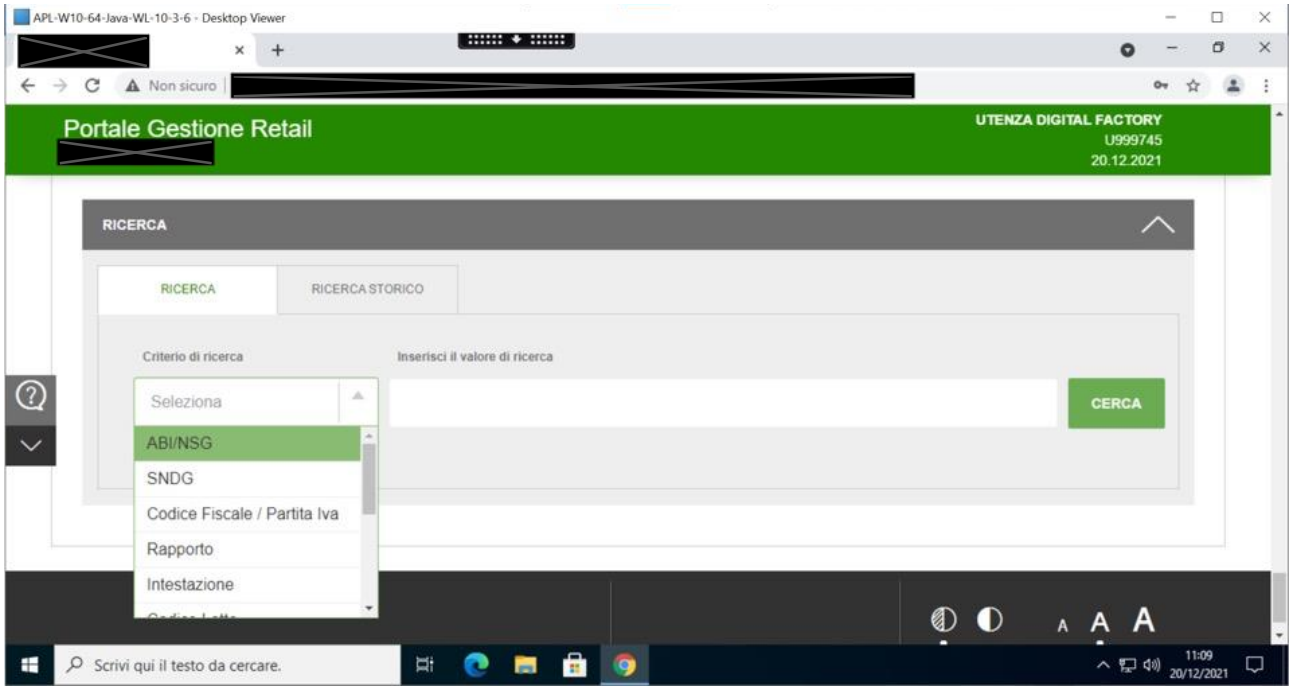

Figure 2.2 PGR – Search section. In this section, the PGR user can search for different position according to different criteria: ABI, NSG, Codice fiscale, Intestazione, Rapporto. Codice Lotto, Fase di Gestione. Once one research criteria have been selected, clicking on the "Cerca" button, the user will be re-directed to a page with a table containing only those positions which meet the criteria.

In the following paragraphs, on the other hand, the functionalities connected with the implementation of the new Upload application will be analysed in more detail.

By the end of 2023 we plan to close all the planned interventions to make this portal work properly.

2.3 Description of the new Upload applicative functions

As a tester, it was my responsibility to identify and analyse the new functions of the application in order to define the test objects.
This is done by means of a document called Functional Analysis, or more simply AFU, received from the client upstream of the testing process.
This document makes it possible to identify and circumscribe the processes that constitute an information system.
Thanks to this activity, it is then possible to define all the characteristics and technical specifications of the software components to be implemented as part of the system under analysis.

The new Upload application aims to create a function that can be reused by all NPL applications that need to carry out massive actions; by uploading certain requests, in fact, the user will be able to make changes to some of the positions registered in PGR. On the Upload application, the process and monitoring of the different requests is centralised.

The functioning of the entire Upload application is now described, starting with login, the uploading of a request and its processing, and ending with the verification of the correct changes on PGR.

Access to the application is via PGR, but will not be available to all user types, but only to a “Super-utente” type. By entering super-utente credentials [Figure 2.3], username and password, the user accesses the PGR homepage [Figure 2.4] where now in the Tab section there will be the “Upload” tab which allows access to the homepage of the new application [Figure 2.5].

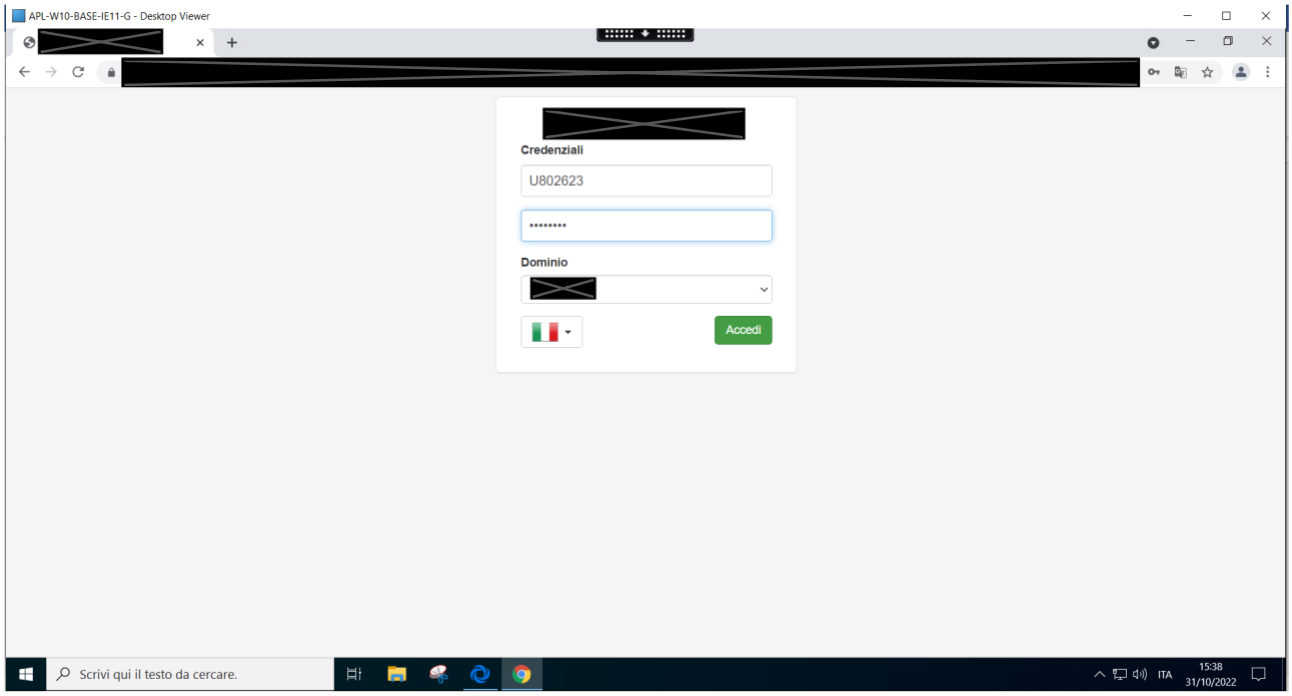

Figure 2.3 Sequence of actions to log in into Upload, Credentials (1)

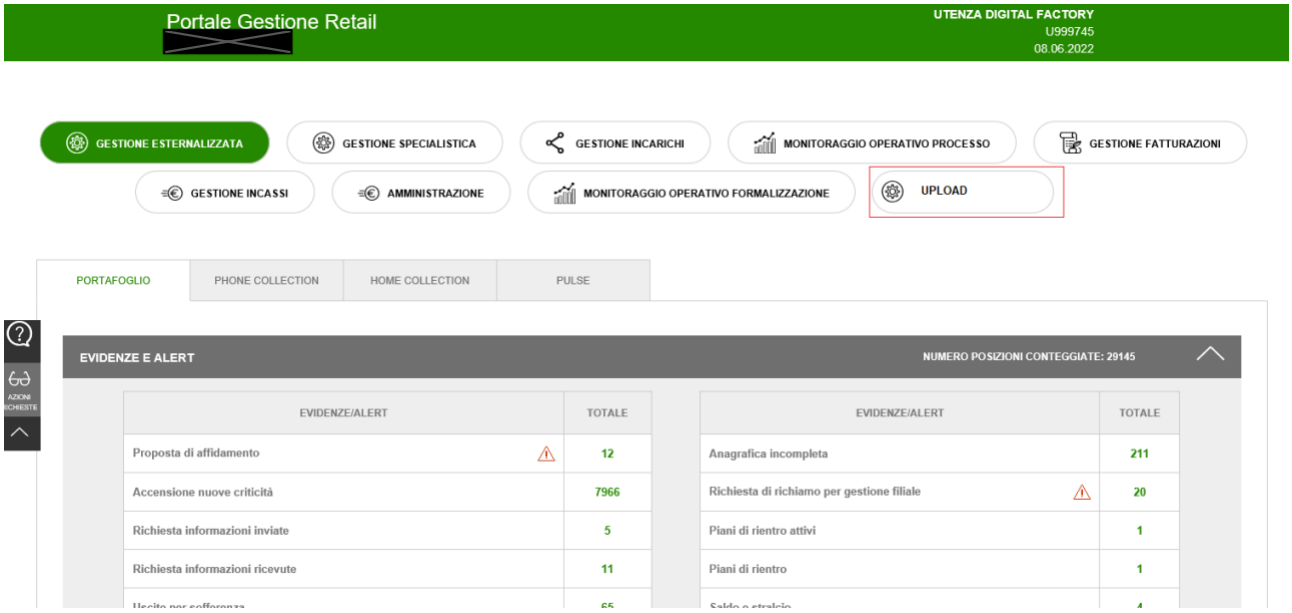

Figure 2.4 Sequence of actions to log in into Upload, PGR – Homepage new Tab (2)

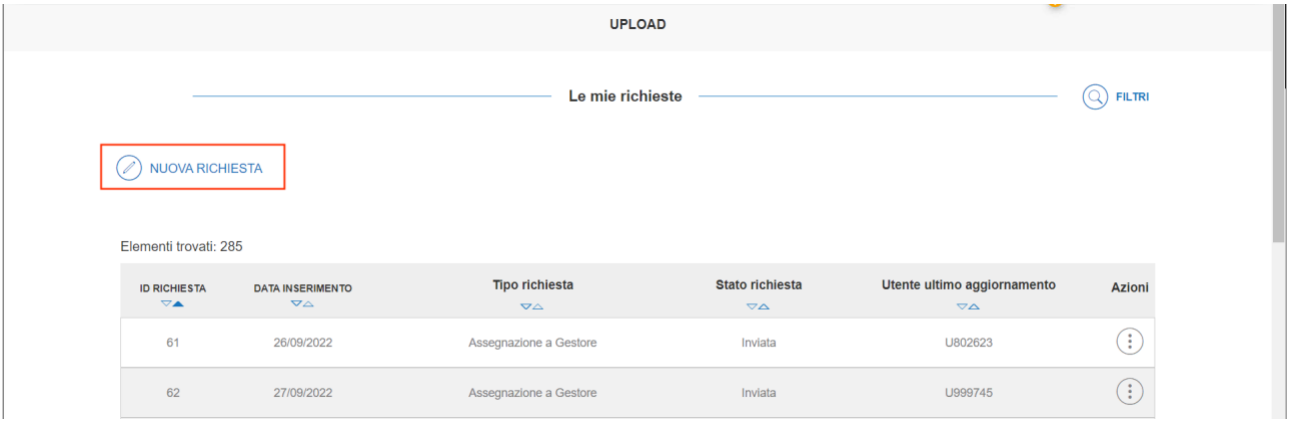

Figure 2.5 Sequence of actions to log in into Upload, Upload – Homepage (3)

Upon entering the page, a table with details of the requests already executed by the same user is displayed.

When clicking on the "Nuova richiesta" button which is highlighted in Figure 2.5, the user lands in the request section [Figure 2.6]. When the page opens, only a drop-down menu is enabled through which the type of request to be entered can be selected.

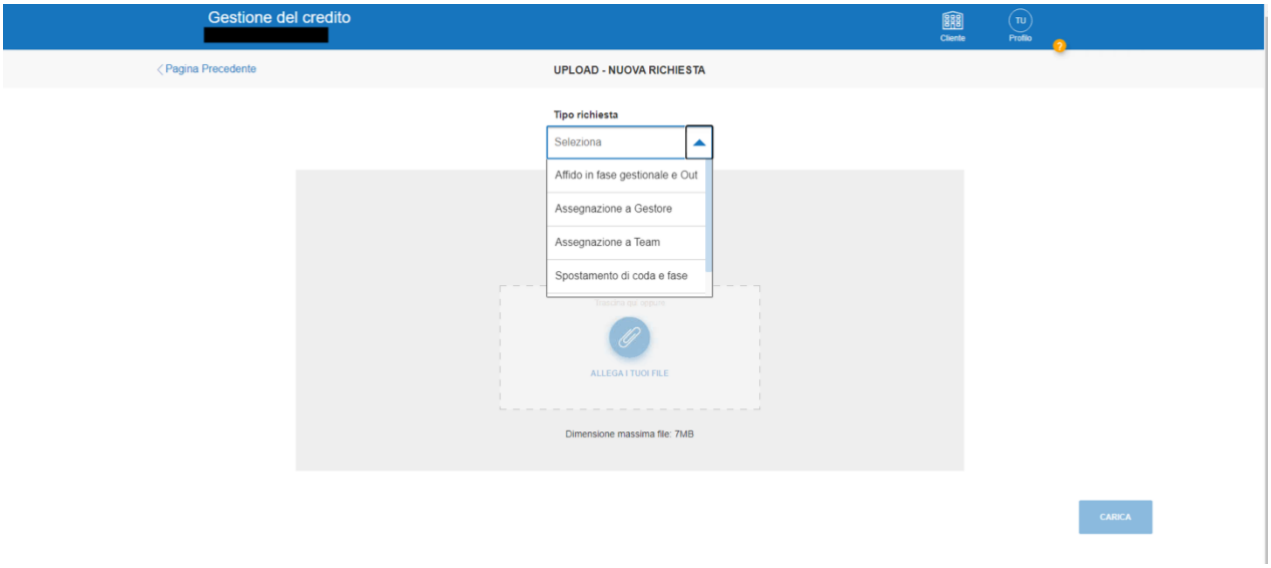

Figure 2.6 Uploading of a new request

The PGR actions that are operated by the UPLOAD web up are as follows:

1) Richiamo da affido esterno: the action allows a list of positions to be recalled from their current fostering.
2) Spostamento di fase e coda: the action allows moving the queue of a list of positions to another queue.
3) Assegnazione a Gestore: the action allows a list of clients to be assigned to specific managers.
4) Assegnazione a Team: the action allows a list of positions to be assigned to certain teams.

5) Affido in fase gestionale e Outsourcer: the action allows specific outsourcers to be assigned to all positions in the list.

After selecting the request type via the drop-down menu at the top you enable the section below, where you can download an excel template to fill in with the positions for which the request is intended and upload the completed file. In this section there is a “SCARICA TEMPLATE” button that allows you to download the excel file. This file allows multiple requests of the same type to be uploaded in a single operation. Each type of request has its own template.
The template for the Richiamo da affido esterno action is reported below as an example [Figure 2.7].

	A	B	C
1	SNDG	ATTIVITA	NOTE
2	0000000073951818	Valutazione specialistica	System test Alten 1
3	0000000068699010	Valutazione specialistica	System test Alten 2
4	0000000084821412	Non affidate da gestire	System test Alten 3

Figure 2.7 Richiamo da affido esterno excel template: It consists of three fields to be filled in: SNDG, ATTIVITÀ, NOTE; an already filled template is shown, in this case the request was made for three different positions.

Below this button is a section in which to upload the compiled file, either by directly dragging the excel or by selecting it from a folder using the “Allega il tuo file” button. The file to be uploaded cannot be larger than 7 MB, otherwise it will not be inserted.

At the bottom of the page there is the "Carica" button, at the click of which a modal informing that the operation has been successfully completed is displayed, the request is uploaded with the status "Inserita"; now the formal checks start, at the end of which the request assumes the status "Controlli terminati" if they are successful or "Controlli falliti" if there is at least one record of the file in KO. Formal checks verify that the template has been filled in correctly for each field.
The user is then taken back to the homepage where he can see the newly created request with the updated status in the table.

Once the formal checks have been successfully completed, the request assumes the status of “Controlli terminati”. At this stage, the user must validate the file that will be sent to the legacy. From the homepage, the “Valida flusso” action is available [Figure 2.8].

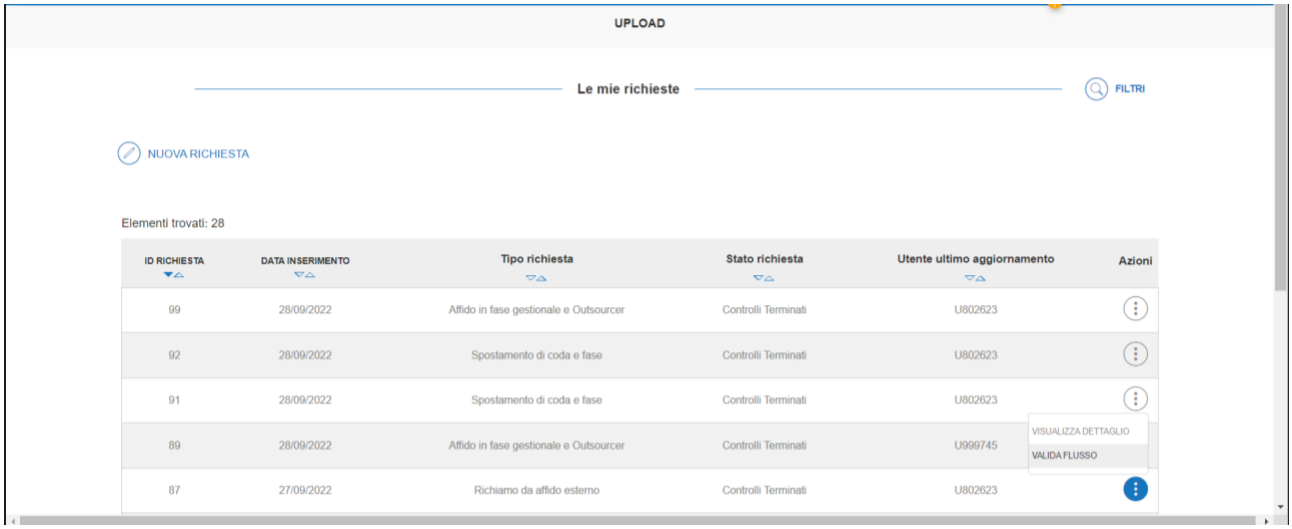

Figure 2.8 Upload – Azioni. By selecting the "Azioni" button the user can access to the page in which the request can be validated to be sent to the legacy.

Clicking on the action lands on a detail page that contains the table with all the records entered in the request [Figure 2.9]. At the bottom of the page are buttons to validate or not validate the request. If the user clicks on "Conferma richiesta", the request assumes the status of "Validata" [Figure 2.10] and is sent to the appropriate legacy. On the other hand, if the user clicks on "Annulla richiesta", the request assumes the status of "Annullata" and is not sent to the legacy.

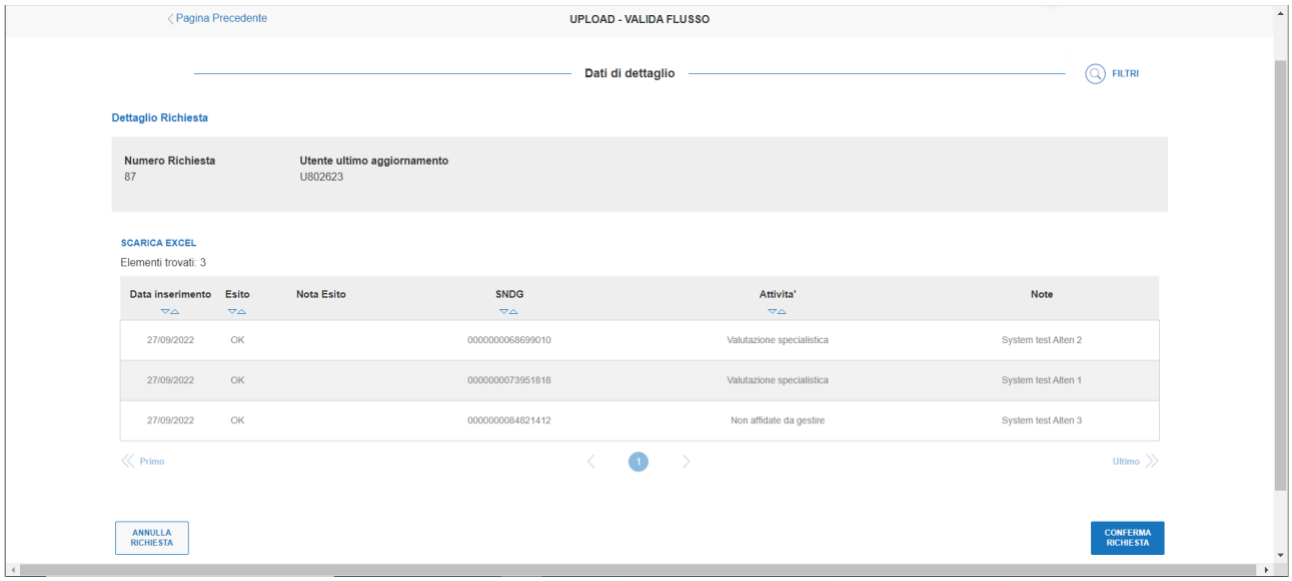

Figure 2.9 Upload – Valida flusso. In the table the detail of the request is given; by clicking on one of the two buttons at the bottom you can either validate or cancel the request.

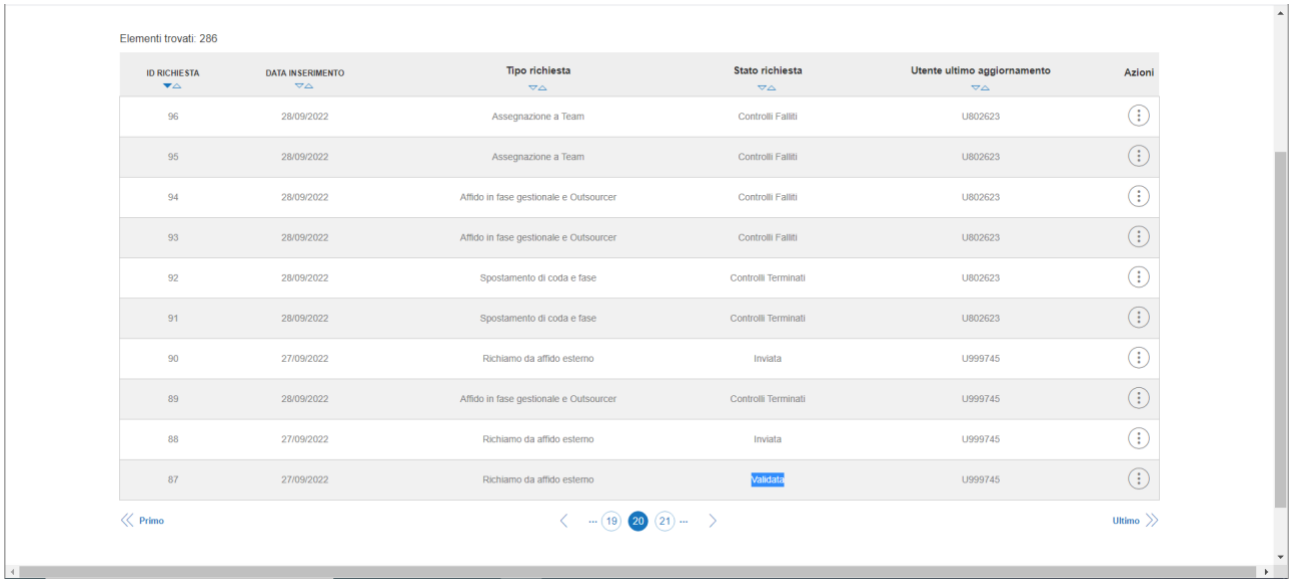

Figure 2.10 Upload – Homepage: status updated. Once the request has been validated, the request status is updated to Validata.

Periodically, a scheduled batch starts that takes all requests in the “Validata” state, groups them by request type, and sends them to each relevant legacy via a flow. The requests thus take on one of the following states:

- “Invio in Corso”: when sending the request to the legacy is started.
- “Inviata”: in the case of successful sending.
- “Invio fallito”: in the event that the first sending attempt fails, with rescheduling of the request.
- “Rinvio in Corso”: in case the first sending attempt fails and the second sending of the request to the legacy is started.
- “Annulllata”: in the event that the second attempt also fails.

When the Legacy receives the request, it processes it and sends to Upload a stream containing the outcomes for each row of each request; the request now assumes the status of "Processata" [Figure 2.11].

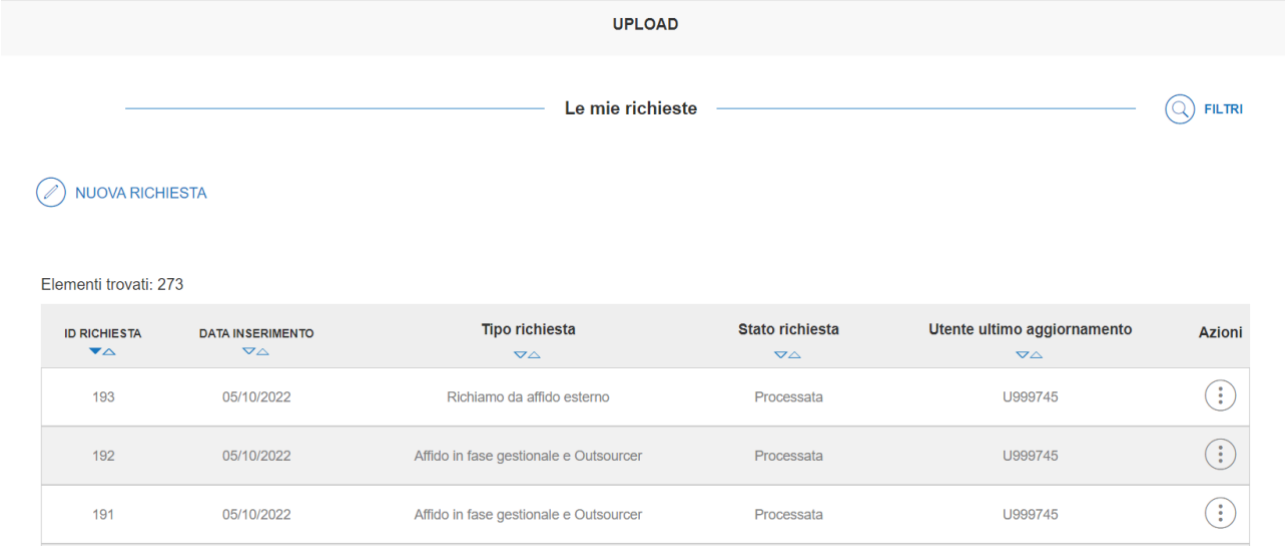

Figure 2.11 Upload – Homepage: status updated. Once the request has been processed by the legacy, it assumes the status "Processata".

Legacy obviously follows some rules when processing the request. These rules differ depending on the type of request received and influence the outcome of each row or position contained within the request.
These rules apply for the action of Richiamo da affido esterno:

1. Verify that the position is open and in PGR perimeter:
- if open/in PGR perimeter proceed with processing,
- if not open/in perimeter there is the return of a KO outcome in the output stream to Upload with the following message, "La posizione non è in perimetro PGR."

2. Verifies that the position has at least one report with open criticality:
- if the report is present with open criticality, proceed with processing the action,
- if the report does not have open criticality, return a KO outcome in the output stream to Upload with the following message: " La posizione non può essere lavorata in quanto è senza criticità per rapporto".

3. Verifies that the position is assigned:
- in case of assigned customer proceed with processing,
- in case of non-assigned customer return a KO result in the output stream to Upload and the following message: "La posizione non può essere lavorata in quanto non affidata".

At this stage the user from the homepage can use the "Visualizza dettaglio" action to view the outcome for each record entered [Figure 2.12].

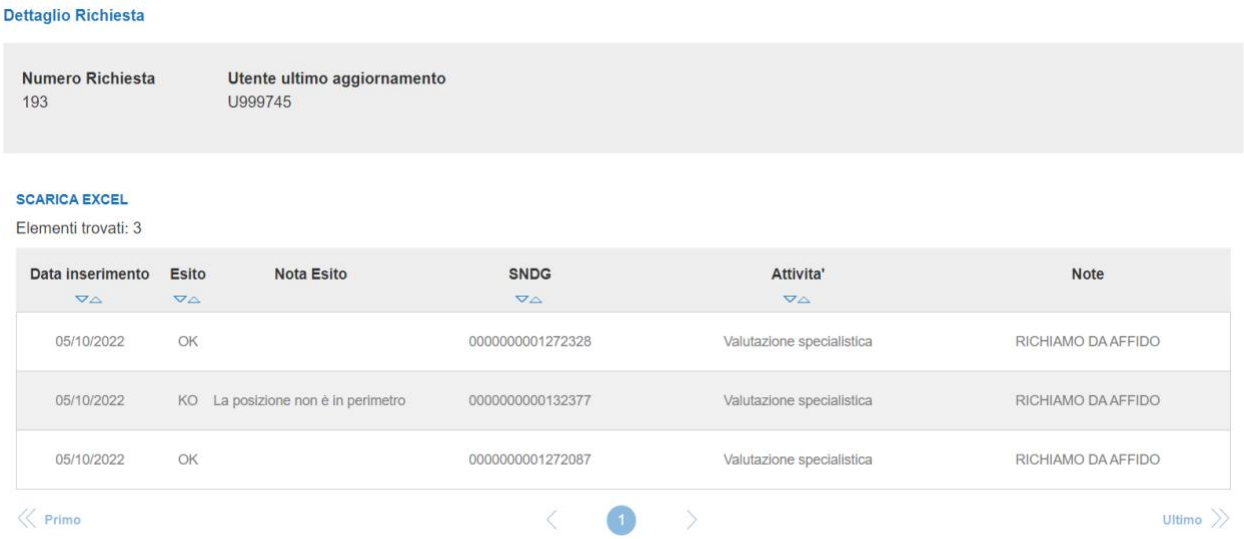

Figure 2.12 Request detail: it is reported a table showing the detail of the request; in this case, for two positions the request was successful, while for the third one it was not, since rule number one was not met.

Thus, Upload is an application in support of the PGR portal through which actions or changes can be made on PGR registered positions. Consequently, once the request uploaded via Upload has been successfully processed, the results of the action must also be visible and updated on the portal [b]. To make this clearer, I will give a simple example: following a request of type “Spostamento di coda e fase” aiming to move a position from one queue to another, e.g., in the “Clienti deceduti da gestire” queue, once the request is successfully processed, in PGR the position must actually be in the new queue.

2.4 Defining test objects and test cases

Once the functionalities of the new Upload application have been analysed, it is time to identify the test objects and test cases to be carried out, so the test design phase starts.
In this phase, the tester proceeds with the drafting of the so-called Testbook (TB), an excel document with a SCART (Strumento per il Caricamento di Requisiti e Test) tool template in which all the tests to be performed to ensure adequate software quality are listed [c]. The tool allows the necessary tests to be loaded directly into ALM (Application Lifecycle Management), the software used for the actual System Test execution phase.

The Testbook is structured as follows [Figure 2.13 and Figure 2.14]:

- “Test/Step”: “Test” is used to add a new test case, “Step” to add a new step to the current test case.
- “Nome del test”: for simplicity and order, it is our convention to keep the same format for all the test names. MO-MPAP0_M004_F001_R052_T001_ALL_Upload_Header_Layout is an example of test case name, where:
  - MO-MPAP0: represents the acronym of the running project,

- M004_F001_R052: represents the so-called "alberatura", which means the folder and suite in which the test will be saved in ALM.
- T001: represents the number of the test
- The text, on the other hand, represents a brief description of the test we are going to perform, in this example we observe a front-end test to verify that the on-page Header layout is displayed and structured as expected.

- "Descrizione": brief description of the test, followed by the PREREQUISITES needed to allow the test to run correctly.
- "Cartella": it contains the path of the folder where the test will be saved when it is loaded into ALM.
- "Autore": user of the tester.
- "Tipo test": there are 4 possible choices:
  - P: positive Front-end test
  - N: negative Front-end test
  - BP: positive Back-end test
  - BN: negative Back-end test
- "Importanza": it can be High, Medium, or Low. Low level is used typically for layout tests, while medium or high for functional tests.
- "Status": there are two possible states; "02. Pronto" in the case of test writing completion, "01. In Lavorazione" if changes still need to be made.

TEST/STE	NOME DEL TEST	DESCRIZIONE	CARTELLA	AUTORE	TIPO TEST	IMPORTANZA	STATUS
Test	MO-MPAP0_M004_F001_R052_T001_ALL_Upload_Header_Layout	Verify the correct display of the homepage provided for the "Upload" applicative. PREREQUISITE: User enabled for upload access and with visibility to several requests entered via the application.	Progetto\Tests\MO/PGR_Portale Gestione Recupero\M004_PULSE\F001_Implementazioni di Front-End\R052 - Upload\01. Homepage	U0I8331	P	Media	02. Pronto

Figure 2.13 Test case example (1)

- "Commenti": this column contains the prerequisites for running the test case. These are reported also in the "Descrizione" column.
- "Nome step":
  - 10: for a new test case
  - 20: for the first step of the current test case, and so on for 30, 40. etc.
- "Descrizione step": it describes, in order, he different steps to be performed for the test execution.

- “Esito atteso”: it contains the expected result as a consequence of the steps identified in the previous step.
- Caratteristiche dati input: as the “Descrizione” and “Commenti” columns, the prerequisites for the test run are reported.

COMMENTI	NOME STEP	DESCRIZIONE	ESITO ATTESO	CARATTERISTICHE DATI INPUT
PREREQUISITE: User enabled for upload access and with visibility to several requests entered via the application.	10	1. Log-in to the homepage of "Upload" as an enabled user. 2. Analyse the top of the page.	Verify that the header is structured as follows: - Label "Credit Workbench" on the left on a blue blackground - "xx" bank logo on a blue blackgeround - "Profile" icon on the right on a blue blackground - Label "Upload" in the center on gray blackground	PREREQUISITE: User enabled for upload access and with visibility to several requests entered via the application.

Figure 2.14 Test case example (2)

Once all aspects described in AFU are covered by the tests in the Testbook, it will be sent to the client for approval.
At this point, the ball is in the customer's court: analyse the TB and communicate any adjustments or changes to be made to it.
By now, the TB can be considered finished and ready for uploading into ALM, in which the tests are organised into suites that define the order in which these tests are to be executed.

Moreover, the TB needs to be analysed carefully especially by the development team. That’s because before the execution phase can start, they must implement the data preparation.
During the data preparation phase, they prepare and send all the necessary data for the tests to be carried out such as users or specific positions, which are visible in the prerequisites section of the individual tests. In the example reported in Figure 2.13 and Figure 2.14, for the test to be correctly executed an Upload user with visibility on some requests previously uploaded in the system is needed.

2.4.1 Some Upload FE test cases

Being Upload an entirely new application, all its functionalities must be tested to verify that they are properly implemented, that they work correctly, that they meet the customer's requirements and thus to ensure a good level of software quality.
The test cases consequently concern the entire application, from the layout and thus the graphics, labels, tables; to the functions, and thus the correct uploading of a request, the updating of its status

following an action, the correct display of error messages, the correct updating of the position status in PGR following an upload action, and so on all the functionalities described in Section 2.3.
Below are now reported some of the test cases identified for the testing process of Upload project [d].

1. MO-MPAP0_M004_F001_R052_T003_ALL_Upload_Homepage_Tabella_Layout.

The first test case example reported is a layout test [Figure 2.15], whose aim is to verify the correct structure of the table in the homepage.

TEST/STEP	NOME DEL TEST	DESCRIZIONE
Test	MO-MPAP0_M004_F001_R052_T003_ALL_Upload_Homepage_Tabella_Layout	Verify the correct display of the homepage provided for the "Upload" applicative. PREREQUISITE: User enabled for upload access and with visibility to several requests entered via the application.

Figure 2.15a Table layout test

COMMENTI	NOME STEP	DESCRIZIONE	ESITO ATTESO	CARATTERISTICHE DATI INPUT
PREREQUISITE: User enabled for upload access and with visibility to several requests entered via the application.	10	1. Log-in to the homepage of "Upload" as an enabled user. 2. Analyse the table on the page.	Verify that the table consists of the following columns: - Id richiesta - Data inserimento - Tipo richiesta - Stato richiesta - Utente inserimento - Azioni	PREREQUISITE: User enabled for upload access and with visibility to several requests entered via the application.

Figure 2.15bTable layout test

According to AFU, the table should present the following columns: Id richiesta, Data inserimento, Tipo richiesta, Stato richiesta Utente ultimo inserimento and Azioni. The aim of the test case is to check whether in system the table is actually structured as expected.

2. MO-MPAP0_M004_F001_R052_T007_ALL_Upload_Homepage_Tabella_Filtri_Applica

This is instead a function test. Moreover, this is an example of a multi-step test [Figure 2.16].
In the applicative, each of the table displayed reports a filtering system according to different fields which are provided by AFU. The filtering section by itself involves the implementation of several

tests: checking that the layout is as expected, that the table on page is filterable for the expected fields, and that the application/cancellation of filters works properly.
The test case at issue aims at verifying the latter: it checks whether the application, step 10, and cancellation, step 20, of filters works properly for all the different fields. Of course, by applying one or more filtering criteria we expect the table on page to be updated showing just the requests that meet the imposed criteria.

TEST/STEP	NOME DEL TEST	DESCRIZIONE
Test	MO-MPAP0_M004_F001_R052_T007_ALL_Upload_Homepage_Tabella_Filtri_Applica	Verify the correct display of the homepage provided for the "Upload" applicative. PREREQUISITE: User enabled for upload access and with visibility to several requests entered via the application.
Step		

Figure 2.16a Filters application test

COMMENTI	NOME STEP	DESCRIZIONE	ESITO ATTESO	CARATTERISTICHE DATI INPUT
PREREQUISITES: User enabled for upload access and with visibility to several requests entered via the application	10	1. Log-in into the homepage of "Upload" as an enabled user 2. Select the "Filtri" icon 3. Set one or more filtering criteria 4. Select the APPLICA button	Verify the correct resetting of the table based on the set of criteria If the application of filters does not yield any result, the label "Nessun risultato trovato" will be displayed	PREREQUISITES: User enabled for upload access and with visibility to several requests entered via the application
	20	Select the CANCELLA FILTRI button	Verify the removal of the filters set in the previous step and the visualization of the original table (no filters set)	

Figure 2.16b Filters application test

3. MO-MPAP0_M004_F001_R052_T070_ALL_Upload_Nuova richiesta_Caricamento documento_Carica
This is another functional test. As outlined earlier, via the “Nuova richiesta” page, once the excel template has been downloaded and filled in, it will be attached in the appropriate section to make the request. For the sequence of actions just described, several test objects and test cases can be identified: verify that the template is generated correctly, verify that the generated template comprises the expected fields for that particular type of request, and finally verify that the upload of the request was successful. This last test case is shown in Figure 2.17.

TEST/STEP	NOME DEL TEST	DESCRIZIONE
Test	MO-MPAP0_M004_F001_R052_T070_ALL_Upload_Nuova richiesta_Caricamento documento_Carica	Verify the correct implementation of the functionality that allows the entry of a new request within the "Upload" functionality. PREREQUISITE: User enabled to access upload and with the ability to enter new requests.

Figure 2.17a Test case uploading a request

COMMENTI	NOME STEP	DESCRIZIONE	ESITO ATTESO	CARATTERISTICHE DATI INPUT
PREREQUISITE: User enabled to access upload and with the ability to enter new requests.	10	1. Log-in to the homepage of Upload as the user specified in the prerequisites 2. Select the NUOVA RICHIESTA button 3. Valorize the "Tipo di richiesta" drop-down menu with one of the available options 4. Select the SCARICA TEMPLATE button 5. Correctly fill out the excel file just generated 6. Attach the file via link or by drag and drop (the file must be less than 7MB) 7. Select the CARICA button	Verify that there is an occurence in the table on the Homepage related to the newly uploaded request and that it has the STATO RICHIESTA column vlued as "Inserita".	PREREQUISITE: User enabled to access upload and with the ability to enter new requests.

Figure 2.17b Test case uploading a request

The purpose of the test is therefore to check that after clicking on the CARICA button the request made is actually taken over and that it is therefore correctly displayed in the table on the Upload homepage where all requests performed are visible.

4. MO-MPAP0_M004_F001_R052_T115_ALL_Upload_Valida flusso_Conferma Richiesta_Conferma

This is also a functional and multi-step test. We have seen how, through the "Valida flusso" page, upon clicking "Conferma richiesta" the request should update its status to "Validata" and will be sent to the appropriate legacy.

The purpose of the test, which is reported in Figure 2.18, is precisely to verify the above, and therefore that the validation action is being processed correctly, step 10, and that the status of the request is appropriately updated, step 20.

TEST/STEP	NOME DEL TEST	DESCRIZIONE
Test	MO-MPAP0_M004_F001_R052_T115_ALL_Upload_Valida flusso_Conferma Richiesta_Conferma	Verify the correct implementation of the functionality that allows the validation of a request within the "Upload" application. PREREQUISITE: User enabled for upload access and with the ability to validate requests; There is at least one request with status "Controlli terminati".
Step		

Figure 2.18a Test case validating request

COMMENTI	NOME STEP	DESCRIZIONE	ESITO ATTESO	CARATTERISTICHE DATI INPUT
PREREQUISITE: User enabled for upload access and with the ability to validate requests; There is at least one request with status "Controlli terminati".	10	1. Log-in to the "Upload" application as the user specified in the prerequisites 2. Select the AZIONI button of an occurrence related to request with status "Controlli terminati" 3. Select the VALIDA FLUSSO action 4. Select the CONFERMA RICHIESTA button 5. Select the CONFERMA button	Verify that the action is processed correctly	PREREQUISITE: User enabled for upload access and with the ability to validate requests; There is at least one request with status "Controlli terminati".
	20	1. Return to the Homepage 2. Research the request for which validation was confirmed in the previous step	Verify that the request being tested has the STATO RICHIESTA valued as "Validata"	

Figure 2.18b Test case validating request

It is important to look at the prerequisites section; in fact, as well as the previously illustrated cases, the test case requires an upload user with visibility into several requests, but in addition now it is necessary that the user has validation enablement and that the application has a request with status "Controlli terminati", necessary conditions for a request to be validated.

2.4.2 Some Upload BE test cases

It is important to mention that the test cases identified, however, do not concern the front-end only. In software development, equally important is the back-end part, which refers to those parts of a computer application or a program's code that allow it to operate and that cannot be accessed by a user.

From a tester's point of view, BE tests might be more challenging than the execution of FE tests. That is because in execution, we cannot proceed independently, but must proceed step by step with the developers as they are the only ones who can verify what is happening in the back-end.

In our case study, the object of back-end tests are all the data flows that Upload and PGR interchange during the execution of certain actions. Therefore, the task of the tests is to verify that after the execution of these actions, the interchanged flows are the correct ones and that they are structured as expected.

1. MO-MPAB0_M004_F002_R048_T148_Upload_Flusso input_PGR_Invio richiesta_Richiamo da affido esterno

This is an example of test case concerning the back-end data flow [Figure 2.19]
Whenever a request is validated, the PGR application receives and acquires an input stream from the external Upload application containing the list of clients on which one of the possible actions has been performed and information about the action implemented. So, the structure and data reported by the flow are different for each type of request. The test case example at issue concerns the Richiamo da affido esterno action.

TEST/STEP	NOME DEL TEST	DESCRIZIONE
Test	MO-MPAB0_M004_F002_R048_T148_Upload_Flusso input PGR_Invio richiesta_Richiamo da affido esterno	Verifify the correct exchange of operational flows between PGR and Upload, and viceversa PREREQUISITE: Correct processing of the "Invio richiesta" flow for "Richiamo da affido esterno" action sent from Upload to PGR.

Figure 2.19a BE test – Input stream request flow

COMMENTI	NOME STEP	DESCRIZIONE	ESITO ATTESO	CARATTERISTICHE DATI INPUT
PREREQUISITE: Correct processing of the "Invio richiesta" flow for "Richiamo da affido esterno" action sent from Upload to PGR.	10	Analyse the flow	Verify that the flow consists of the following fields: - PROG_RICHIESTA - DATA_FLUSSO - ID_RIGA - SNDG - ATTIVITA' - NOTE - MATR_UTENTE	PREREQUISITE: Correct processing of the "Invio richiesta" flow for "Richiamo da affido esterno" action sent from Upload to PGR.

Figure 2.19b BE test – Input stream request flow

The purpose of the test is to verify that the above-mentioned flow contains the required information, i.e. PROG_RICHIESTA, the identification number of the request or requests whose data are contained in the flow; DATA_FLUSSO, the date on which that request was made; ID_RIGA, id of the request record; SNDG, unique code to identify the customer relating to that record; ATTIVITÀ,

type of activity; NOTE, a free field in which the user can enter any notes; MATR_UTENTE, the user of the person who made the request.
For the sake of clarity, the flow that was attached as evidence to the test case at the execution stage is shown [Figure 2.20].

	A	B	C	D	E	F	G
1	180	20221005	1221	4561406	COVS	SGR	U999745
2	180	20221005	1222	6980806	NOAC	SGR	U999745
3	180	20221005	1223	8441395	COVS	SGR	U999745
4	180	20221005	1224	14096258	NOAC	SGR	U999745
5	180	20221005	1225	24439624	NOAC	SGR	U999745
6	182	20221005	1232	4561406	COVS	SGR	U999745
7	182	20221005	1233	6980806	NOAC	SGR	U999745
8	182	20221005	1234	8441395	COVS	SGR	U999745
9	182	20221005	1235	14096258	NOAC	SGR	U999745
10	182	20221005	1236	24439624	NOAC	SGR	U999745
11	193	20221005	1255	1272087	COVS	RICHIAMO DA AFFIDO	U999745
12	193	20221005	1256	1272328	COVS	RICHIAMO DA AFFIDO	U999745
13	193	20221005	1257	132377	COVS	RICHIAMO DA AFFIDO	U999745

Figure 2.20 Input stream flow attached evidence

The test in question therefore involves a twofold check. In addition to checking that the flow presents the correct information, it is also necessary to check that the data for each of the requests in the flow exactly match those actually uploaded to Upload.
This means, for example, checking that request ID 193 was actually made for records 00000001272087, 00000001272328 and 00000001132377 as stated in the flow in Figure 2.20.

2. MO-MPAB0_M004_F002_R048_T180_Upload_Flusso output PGR_Ricezione esiti_Richiamo da affido esterno

We just saw that once a new request is validated on UPLOAD, it sends the input stream to PGR, which will perform the due checks.
Now, the PGR application sends to the external Upload application an output stream containing the list of the outcomes of the processing of the action for each position processed and, in case of KO, the related error message, displayed in system.
The example for this test case is once again given for the action Richiamo da affido esterno [Figure 2.21]

TEST/STEP	NOME DEL TEST	DESCRIZIONE
Test	MO-MPAB0_M004_F002_R048_T180_Upload_Flusso output PGR_Ricezione esiti_Richiamo da affido esterno	Verify the correct exchange of operational flows between PGR and Upload, and viceversa. PREREQUISITE: Correct processing of the "Ricezione esiti" flow for "Richiamo da affido esterno" action sent from PGR to Upload.

Figure 2.21a BE test – Output stream from PGR

COMMENTI	NOME STEP	DESCRIZIONE	ESITO ATTESO	CARATTERISTICHE DATI INPUT
PREREQUISITE: Correct processing of the "Ricezione esiti" flow for "Richiamo da affido esterno" action sent from PGR to Upload.	10	Analyse the flow	Verify that the flow consists of the followinf fields: - PROG_RICHIESTA - DATA_FLUSSO - ID_RIGA - NOTA_ESITO	PREREQUISITE: Correct processing of the "Ricezione esiti" flow for "Richiamo da affido esterno" action sent from PGR to Upload.

Figure 2.21b BE test – Output stream from PGR

The purpose of the test is the same as the previous one, i.e., to check the structure and that the data contained is correct, but this time it concerns a flow sent from PGR to Upload and not vice versa. Again, evidence of the attached flow is shown [Figure 2.22].

Figure 2.22 Output stream flow attached evidence

It is important to note that in this case the last field NOTA_ESITO is correctly not filled in; it is in fact dedicated to the error message for a record that went KO after the checks, while in this case all three records show OK results.

2.5 Upload System test execution

The release and thus the start of the tests were scheduled to start on 21/09/2022.

As mentioned above, it is usual for the execution phase to follow the schedule imposed by the suites created in the ALM test environment. For the case study, two main suites were planned: Front-end, which provides for the execution of 126 tests, and Back-end, which provides for the execution of 52 tests.
The two main suites are in turn divided into sub-suites.

The FE was subdivided as follows:

- PGR: where all tests to be run on the PGR portal are loaded. In turn, this suite is divided into Superutente, User GS, User GE, Pulse, Outsourcer depending on the type of user that is to be used to run the test.
- UPLOAD: where all the tests to be run on the Upload application are uploaded. This in turn is divided into the four main pages, i.e., Homepage, Inserimento richiesta, Visualizza dettaglio, Valida flusso, depending on where the function under test is performed.

The BE, on the other hand, consists of a single Back-end suite where all tests relating to the flow exchange between PGR and Upload are loaded.

The execution plan was to execute the FE first, and then the BE. This is mainly because, as mentioned, the execution of the BE tests requires full support from development parties.
Below is a breakdown of the test suites directly from ALM [Figure 2.23]

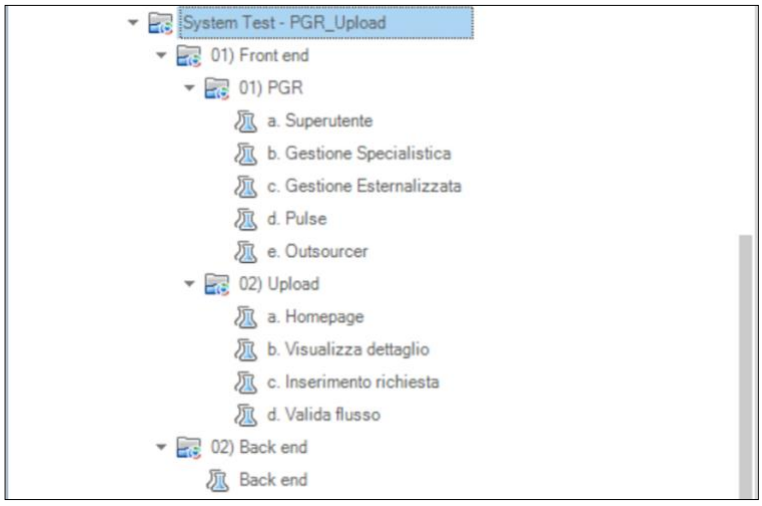

Figure 2.23 ALM – System test's suites classification

Eleven days were estimated for the system test activities. However, a flawed data preparation and an imperfect test environment that resulted in numerous defects being opened in the first few days caused the end of activities to be postponed to 07/10/2022, two days later than planned.
Moreover, only 171 out of 179 initial tests were found to be executable during the execution phase, following the client's instruction. As to the remaining eight tests:

- one test was eliminated from the execution plan because it was not executable in the test environment;
- seven tests were executed as N/A, as the result of an error occurred during the creation of the test case in the data preparation.

In addition to the seven N/A tests, seventy-one defects were detected and reported during execution. This brings the total number of errors found to seventy-eight; despite this, the seventy-one functional defects were corrected prior to the production release.
However, results obtained during the process will be analysed in more detail in Chapter 3.

2.5.1 Tools supporting execution phase in the testing process

In this section, a brief guide to the tool used for execution is given.

As mentioned earlier, the execution phase is carried out through ALM, Application Lifecycle Management.

The data of the individual tests obviously match those of the TB, which, as mentioned, once completed, is directly loaded into ALM thanks to the SCART template.

An example of ALM visualization of the test to be executed is shown in Figure 2.24 and Figure 2.25.

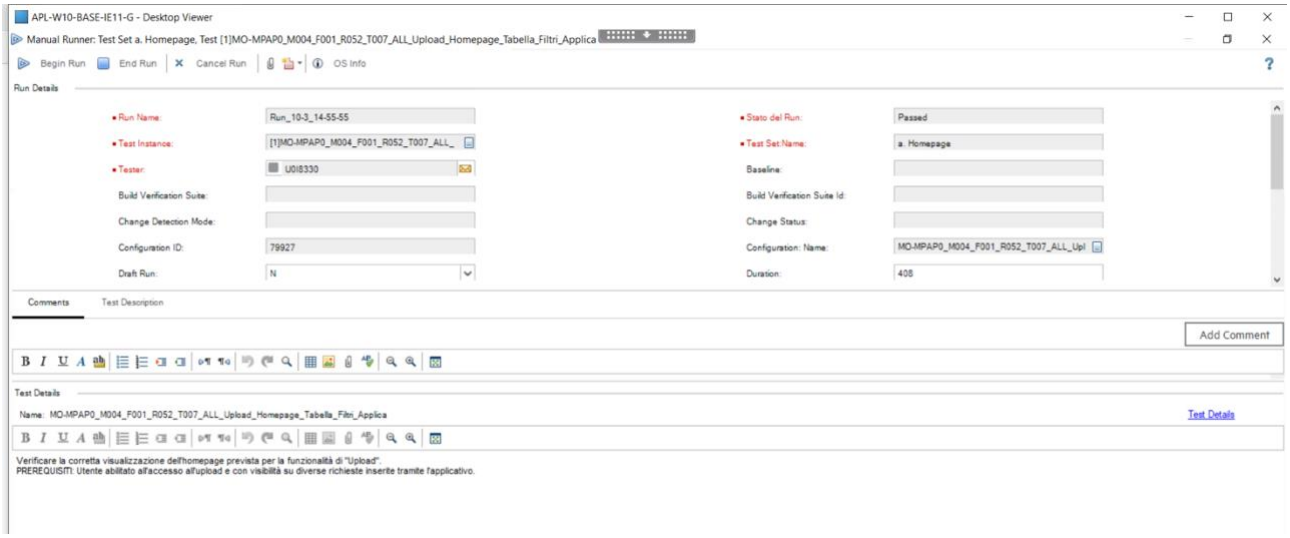

Figure 2.24 Representation of test launching

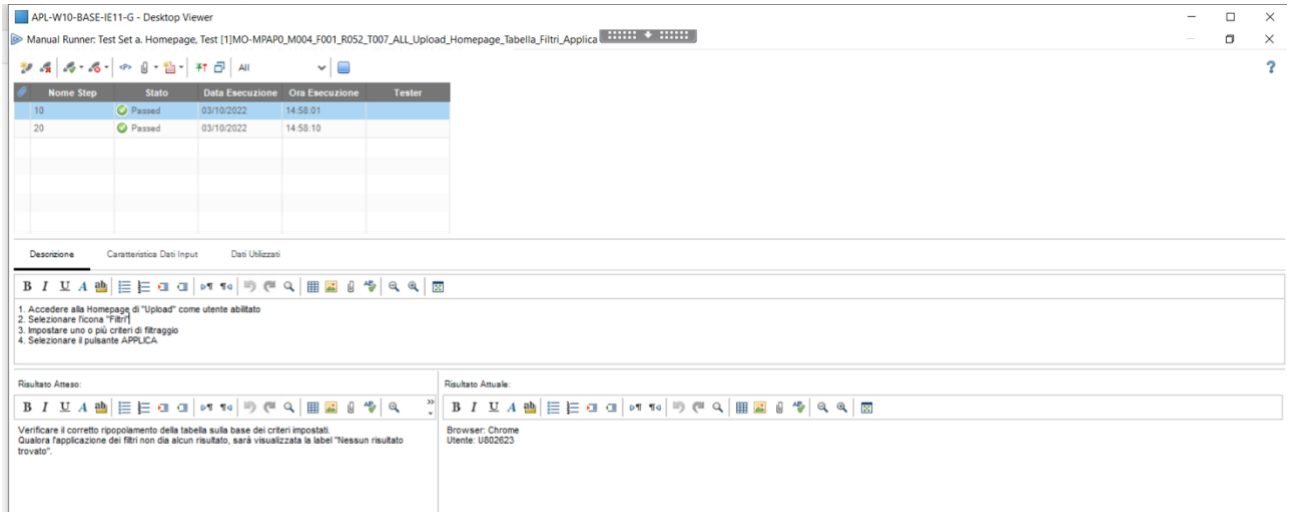

Figure 2.25 Running test details

The first representation [Figure 2.24] gives a general overview of the test, including information such as the test instance, the run name, the status of the test and the suite in which the test was entered. From a tester's point of view, the most relevant part is the Test Details section, in which a first brief description of the test and the prerequisites necessary for its execution are given.

Using the “Begin run” button, the tester lands in the actual test execution section [Figure 2.25]. Here, the information required to run the test is shown, such as the steps to be performed and the expected result.
At the top is the field for the test result. A single test has six possible outcomes:

- *Passed*: when the expected result, “Risultato atteso”, of the test exactly matches what we actually see in system.
- *Passed with minor*: when it can be considered almost as passed because of some imperfection.
- *Not completed*: this is a temporary state; when the test has been started but there is a need for time to determine the outcome.
- *N/A*: the test is found to be non-executable. There may be various reasons for this, such as inadequate data preparation so the necessary test case data was not provided.
- *Blocked*: similar to N/A, usually used when multiple tests cannot be run for the same reason.
- *Failed*: the expected outcome does not match what we see in system. When the status of a test is in failed, it is the tester's job to report the anomaly (or defect) to the customer.

The “Risultato attuale” section is a free-entry field to be filled in by the tester. In it, some information concerning the execution of the test is provided: the browser used to perform the test (usually Chrome), the user and the position used, when needed.
When running a test, it is always recommended to attach one or more pieces of evidence concerning the outcome, such as a file or a screenshot of the application. This is done via the 'Attach to run' section of ALM.

We have already seen how the system's objective is to guarantee a high level of product quality whatever the product. To do this, the tester aims to identify defects to be notified, so that they are corrected in time before release for production.
When a new defect is detected, the test result inevitably becomes “Failed”. The procedure for opening a new defect is now explained.

In ALM select the "New defect" button. Through the section shown in Figure 2.26, it will be possible to explain the reason for the failed test with attached evidence of the failure. In addition, the defect will be assigned to the appropriate party so that the failure can be resolved: the software will directly send them an email once the field “Assegnato a” has been filled with the pertinent user.
It is customary to give already in the title an indication of the type of defect detected, the page and the section where it was found.
Once resolved, the development party will change the status of the defect to "Corretto".

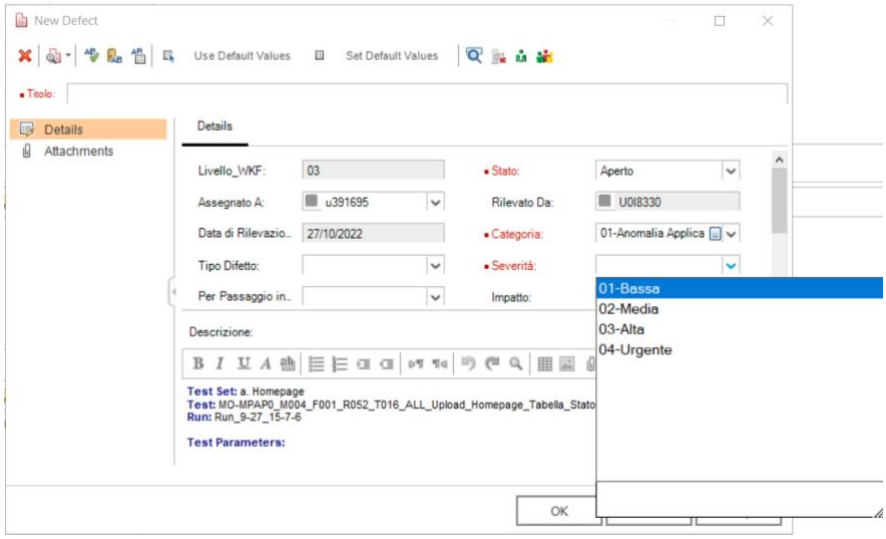

Figure 2.26 ALM – New defect section

At this point, the tester can retest to verify that the defect has indeed been corrected. There are two options:

1. the anomaly has been corrected and, therefore, the test results "Passed”, and the status of the defect will be changed to "Chiuso".
2. the anomaly persists, or a new one occurs, so the defect is reassigned or a new one is opened.

Chapter 3. Testing monitoring and analysis of results

In this chapter, a complete overview of the system test process is given.
In Chapter 2, we examined the new functionalities that led to the definition and the execution of the various test cases.
Now, we try to give a complete view of the process, analysing its progress and the results obtained, and therefore reporting some of the detected anomalies. This type of information is contained within the reports drawn up during the process. Therefore, an outline of the reporting process is also given.

3.1 Test Reports: content and purpose

Monitoring and reporting of system test activities is an important aspect of software development and testing. It involves tracking the progress of the testing activities, identifying any issues or defects, and providing regular updates to stakeholders on the status of the testing.
The latter is a very important aspect of monitoring and reporting ST activities; regular progress reports and status updates should be provided to stakeholders, such as project managers and development teams, to keep them informed on the progress of testing and any issues that have been identified. Additionally, stakeholders should be provided with detailed information on the test results, including the number of defects found, their severity, and any resolutions that have been implemented.

Two documents are drawn up for this purpose: Test Progress Report and Test Summary Report.

The former is prepared during the testing activities. A Test Progress Report typically provides an overview of the testing activities that have been completed, the progress made, and any issues or blockers that have been encountered. It includes information such as the number of test cases executed, the number of defects found, the test coverage achieved, and the overall status of the testing. The report is usually prepared on a regular basis, such as monthly, weekly, or even daily and is intended to keep stakeholders informed of the testing progress and to identify any issues that need to be addressed.

The latter, instead, is prepared at the end of the testing phase. A Test Summary Report is a document that provides a final summary of the testing activities. It includes information such as the number of test cases executed, the number of defects found, the test coverage achieved, and the overall status of the testing. It also includes a summary of the test results, including information about what happened during the test period and any departures from original plan. This report is intended to provide stakeholders with a comprehensive overview of the testing process and results.

During the testing activities for the Upload project a Daily Progress Report was drawn up, thanks to which stakeholders were informed of the current progress of the activities and provided with a list of all the currently open defects in ALM.

3.2 Analysis of results obtained during Upload execution

During the execution phase of the system test, as already mentioned, a progress report is prepared at the end of each working day to be sent to all stakeholders to keep them updated.

An example of the daily report is shown below in Figure 3.1. This report was drafted during the execution phase of PGR – Upload on 28/09/2022, when the percentage of progress was approximately 67%.

	A	D	E	F	G	H	I	J	K	L	M	N	O	P
1	System Test - PGR_Upload	TestSet	Totali	Passed	Passed with Minor	Failed	No Run	Not Completed	Blocked	N/A	%Eseguiti	%OK	%KO / Eseguiti	%OK / Eseguiti
2														
3														
4	Root\2022\PRJ466597-02\MDC_PRJ466597-02_STEP145\System Test - PGR_Upload\01) Front end\01) PGR	a. Superutente	8	3	0	0	5	0	0	0	37,50%	37,50%	0,00%	100,00%
5		b. Gestione Specialistica	1	1	0	0	0	0	0	0	100,00%	100,00%	0,00%	100,00%
6		c. Gestione Esternalizzata	1	1	0	0	0	0	0	0	100,00%	100,00%	0,00%	100,00%
7		d. Pulse	1	1	0	0	0	0	0	0	100,00%	100,00%	0,00%	100,00%
8		e. Outsourcer	1	1	0	0	0	0	0	0	100,00%	100,00%	0,00%	100,00%
9														
10		Totali cartella	12	7	0	0	5	0	0	0	58,33%	58,33%	0,00%	100,00%
11														
12	Root\2022\PRJ466597-02\MDC_PRJ466597-02_STEP145\System Test - PGR_Upload\01) Front end\02) Upload	a. Homepage	22	18	0	1	3	0	0	0	86,36%	81,82%	5,26%	94,74%
13		b. Visualizza dettaglio	37	37	0	0	0	0	0	0	100,00%	100,00%	0,00%	100,00%
14		c. Inserimento richiesta	13	11	0	2	0	0	0	0	100,00%	84,62%	15,38%	84,62%
15		d. Valida flusso	43	43	0	0	0	0	0	0	100,00%	100,00%	0,00%	100,00%
16														
17		Totali cartella	115	109	0	3	3	0	0	0	97,39%	94,78%	2,68%	97,32%
18														
19	Root\2022\PRJ466597-02\MDC_PRJ466597-02_STEP145\System Test - PGR_Upload\02) Back end	Back end	52	0	0	0	52	0	0	0	0,00%	0,00%	0,00%	0,00%
20														
21		Totali cartella	52	0	0	0	52	0	0	0	0,00%	0,00%	0,00%	0,00%
22														
23														
24		TOTALI GENERALI	179	116	0	3	60	0	0	0	66,48%	64,80%	2,52%	97,48%
25														
26		Delta report precedente	0	11	0	-9	-2	0	0	0	1,12%	6,14%	-7,74%	7,74%

Figure 3.1. Daily progress report

This kind of report is structured as follows:

- Column A: it represents the system activities at issue.
- Column D: It divides the different test cases according to their suites.
- Column E to L: they list all the different possible test outcomes and are filled in with the number of tests that have been so executed.
- Column M to P: They list respectively the percentage of executed tests compared to the total number of tests to be performed, the percentage of passed tests compared to the total, the percentage of failed tests compared to executed tests, the percentage of passed tests compared to executed tests.

The last row represents the delta progress from the previous day.

Along with the progress report, stakeholders are also provided daily with the list of defects open in ALM at the time.

The example is shown in the Figure 3.2:

	A	B	C	D	E	F	G	H	I	J
1	Defect ID	Stato	Titolo	Severità	Priorità	Rilevato Da	Rilevato nel Cycle	Assegnato A	Data di Rilevazione	Data Stima Correzione
2	17112	InLavorazione	System test - UPLOAD - Nuova richiesta - Scarica template - Affido in fase gestionale e Outsourcer - Colonna fase di lavorazione formattata erroneamente	01-Bassa	01-Bassa	U0I8330	System Test - PGR_Upload	u0i3308	27/09/22	30/09/22
3	17122	InLavorazione	System test - UPLOAD - Nuova richiesta - Affido in fase gestionale e outsourcer - Anomalia KO per fase di lavorazione "AUTOMATICO"	02-Media	02-Media	U0I8330	System Test - PGR_Upload	u0i3308	28/09/22	29/09/22
4	17124	InLavorazione	System test - UPLOAD - Homepage - Stato richiesta - Anomalia cambio utente dopo conferma o annulla richiesta	02-Media	02-Media	u0i8330	System Test - PGR_Upload	u0k6112	28/09/22	03/10/22

Figure 3.2 Defects list report

- The first column represents the defect ID.
- The second column represents the actual status of the defect, which in the current example is 'In Lavorazione, i.e., it has been taken over by the development team.
- Column C is filled in with the title given to the defect.
- Column D and E represent respectively its severity and priority, whose concepts will be introduced in the next section.
- Column F and H shows two different users: respectively the one who detected the anomaly and the one to whom the anomaly has been assigned to be corrected.
- Column G reports the project on which we are working on.
- Column I and J report two date: the one on which the defect was detected and the one on which it is estimated to correct it.

To monitor and have a complete view of the entire process the data from the daily report are reported in a spreadsheet by the Test Manager, which is reported in Figure 3.3.

			Giorni	TC TOT	TC/gg	Effort											
Pulse/PGR - Funzionalità Upload	inizio System Test: 21/09/2022		11	179	16,27273	0,813636											
Check	TC previsti	gg	DATA		Totali	Passed	Passed With Minor	Failed	No Run	Not Completed	Blocked	N/A	%Eseguiti	%OK	%KO / Eseguiti	%OK / Eseguiti	DEFECT
OK	16	1	21/09/22	TOTALI GENERALI	179	24	0	16	139	0	0	0	22,35%	13,41%	40,00%	60,00%	16
OK	33	2	22/09/22	TOTALI GENERALI	179	53	0	15	111	0	0	0	37,99%	29,61%	22,06%	77,94%	18
OK	49	3	23/09/22	TOTALI GENERALI	179	68	0	17	94	0	0	0	47,49%	37,99%	20,00%	80,00%	15
OK	65	4	26/09/22	TOTALI GENERALI	179	86	0	18	75	0	0	0	58,10%	48,04%	17,31%	82,69%	16
OK	81	5	27/09/22	TOTALI GENERALI	179	105	0	12	62	0	0	0	65,36%	58,66%	10,26%	89,74%	8
OK	98	6	28/09/22	TOTALI GENERALI	179	116	0	3	60	0	0	0	66,48%	64,80%	2,52%	97,48%	3
OK	114	7	29/09/22	TOTALI GENERALI	179	118	0	1	60	0	0	0	66,48%	65,92%	0,84%	99,16%	1
KO	130	8	30/09/22	TOTALI GENERALI	179	118	0	1	60	0	0	0	66,48%	65,92%	0,84%	99,16%	1
KO	146	9	03/10/22	TOTALI GENERALI	179	120	0	1	58	0	0	0	67,60%	67,04%	0,83%	99,17%	1
KO	162	10	04/10/22	TOTALI GENERALI	178	127	0	0	51	0	0	0	71,35%	71,35%	0,00%	100,00%	0
KO	178	11	05/10/22	TOTALI GENERALI	178	146	0	0	32	0	0	0	82,02%	82,02%	0,00%	100,00%	0
OK	178	12	06/10/22	TOTALI GENERALI	178	170	0	1	0	0	0	7	100,00%	99,42%	0,58%	99,42%	1
OK	178	13	07/10/22	TOTALI GENERALI	178	171	0	0	0	0	0	7	100,00%	100,00%	0,00%	100,00%	0

Figure 3.3 ST activity Monitoring

According to schedule, an average of about sixteen tests per day was required to successfully complete the task in the allotted time. As shown in the image above [Figure 3.3], the activity proceeded as estimated for the first week.

Moreover, we can see that the distribution of failed tests is concentrated in the early days of the project. Failed tests mean that defects have been reported. It was the reporting of these defects that caused a slowdown in the execution of the tests, this was due to the fact that many anomalies were occurring in different parts of the application making it effectively impossible to proceed with testing activities until they were corrected. On 29/09/2022 and 30/09/2022, in fact, no progress was recorded.

During these days, the planned average number of daily tests could not be sustained, this has led to a two-day delay in the completion of activities.
The “Check” column intervenes to confirm this: this column shows a positive result, OK, if the test cases scheduled up to that day are less than or equal to the total number of tests performed. For example, on 30/09/2022 at least 130 tests should have been performed, whereas 119 tests were executed.

Therefore, for the above-mentioned reasons, the ST went through four nearly deadlocked days in which it was not possible to keep up the planned average number of daily tests.
A summary of the ST's progress just described is given in the table [Table 3.1] and graph below [Figure 3.4]:

DATA	Passed	Failed	Executed	Total	Planned
21/09/22	24	16	40	179	16,27
22/09/22	53	15	28	179	32,55
23/09/22	68	17	17	179	48,82
26/09/22	86	18	19	179	65,09
27/09/22	105	12	13	179	81,36
28/09/22	116	3	2	179	97,64
29/09/22	118	1	0	179	113,91
30/09/22	118	1	0	179	130,18
03/10/22	120	1	2	179	146,45
04/10/22	127	0	7	178	161,82
05/10/22	146	0	19	178	178,00
06/10/22	170	1	32	178	178,00
07/10/22	171	0	0	178	178,00

Table 3.1. ST daily progress

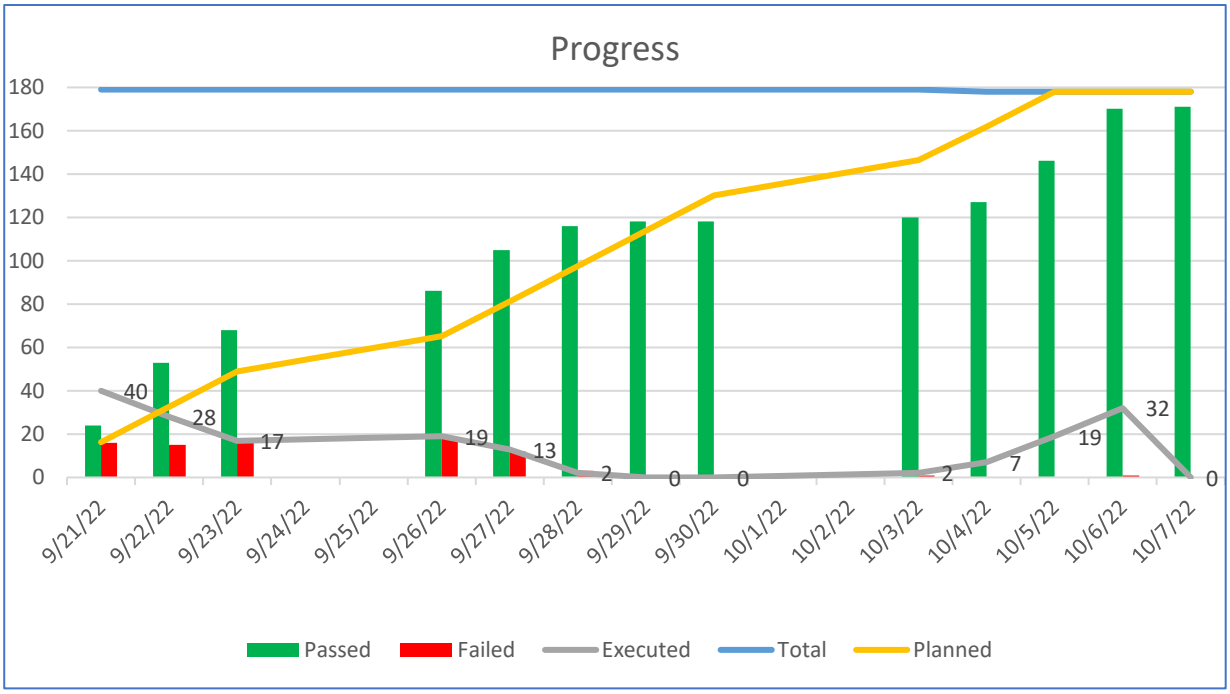

Figure 3.4. ST progress graph

3.3 Analysis of defects detected

As specified at the end of Chapter 2, a total of seventy-eight defects were identified during the testing activity.

Before carrying out a defects analysis trying to identify their main causes, some notions must first be clarified.

A person may make an error that can lead to the introduction of a defect into software code or another related work product.

Hence, an error leads to the introduction of a defect in a work product and may be the trigger for another error that leads to the introduction of a defect in a related work product. For example, an error in requirements identification may lead to a defect in requirements, resulting in a programming error that will lead to a defect in the code.

The latter are obviously among the most typical when we talk about software testing: a defect in the code if executed can cause a failure.

Having defined errors, defects, and failures, we can give another important definition: root causes. The root causes of defects are the first actions or conditions that contributed to the creation of the defects. Defects can be analysed to identify their root causes to reduce similar defects occurring in the future. By focusing on the most important root causes, root cause analysis can lead to process improvements that prevent the introduction of a significant number of future defects [3].

It is the responsibility of the test team to identify and report defects. Identifying a defect precedes the action of opening tasks describing defects; that is why the term "opening a defect" is used in tester jargon.

In order to manage all defects until they are resolved, an organization should establish a Defect Management process that includes rules for classification. One method for defect classification involves ranking them according to priority and severity parameters.
Severity and priority are terms that are often used interchangeably, but this is obviously a misinterpretation; in fact, severity is a degree of impact a bug or a defect has on the software application under test, a higher effect of bug/defect on system functionality will lead to a higher severity level; priority is defined as the order in which a defect should be fixed, higher the priority the sooner the defect should be resolved.

Priority can be classified as:

1. Urgent: a bug with an immediate priority must be resolved as fast as possible, usually within twenty-four hours, in particular a bug is defined as urgent when it is 'blocking', i.e., when it blocks other functionalities, thus making it impossible to continue testing.
2. High and Medium priority: high and medium priority are generally associated with functional problems of the products, in the specific case of an Internet banking software, they are linked to the impossibility, for example, of performing some kind of function or even log-in problems. The choice between high and medium depends essentially on the type of function in question, whether it is a characteristic function of the product or not. Medium priority bugs should only be resolved after the developers have dealt with the higher priority levels.
3. Low: a defect with low priority may concern an 'aesthetic defect', i.e., a layout that does not correspond to what is shown in any mock-up of the project. A low priority bug might concern improvements in product design.

The same classification also applies for severity.

With particular reference to the PGR - Upload project, the main causes of defects found during execution are reported:

- Functionality failure or not available: this is the case for those anomalies that concern dispositive actions. For example, defect ID 17017 falls into this category as it concerns the malfunctioning of the download excel functionality.
- Lack of data on which to carry out the tests: this is the cause of the failure of the seven back-end tests, which were rated as N/A. During the data preparation, some positions that were necessary to carry out the mentioned tests were not possible to create.
- Problems in writing code at graphic and design level: with reference to layout problems, characterised for example by an incorrect location on the page or by an incorrect label and therefore placed in the low severity section.

As examples, two anomalies identified in Upload activities are now reported:

1. Defect ID 17075

The defect is related to the test MO-MPAP0_M004_F001_R052_T080_ALL_Upload_Valida flusso_Tabella_Richiamo da affido esterno_Filtri_Applica.

This test is intended to verify that the application of filters, for the expected fields, works properly; a similar test was reported at point 2 in Section 2.4.1.
In particular, an anomaly was found in the filter setting according to the "Attività" field. As can be seen in the evidence shown in Figure 3.5 and Figure 3.6, once the filtering criteria is set, upon clicking the apply button, the table shows no results even though positions meeting that criteria were present.

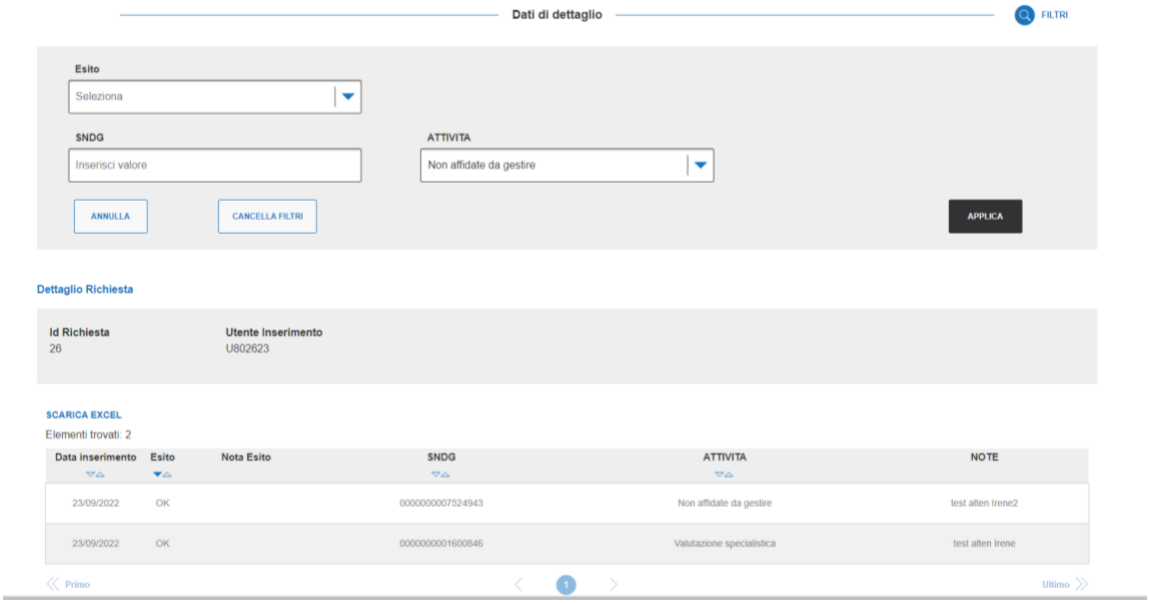

Figure 3.5 Defect's evidence. The table shown is the original one, with no filters applied.

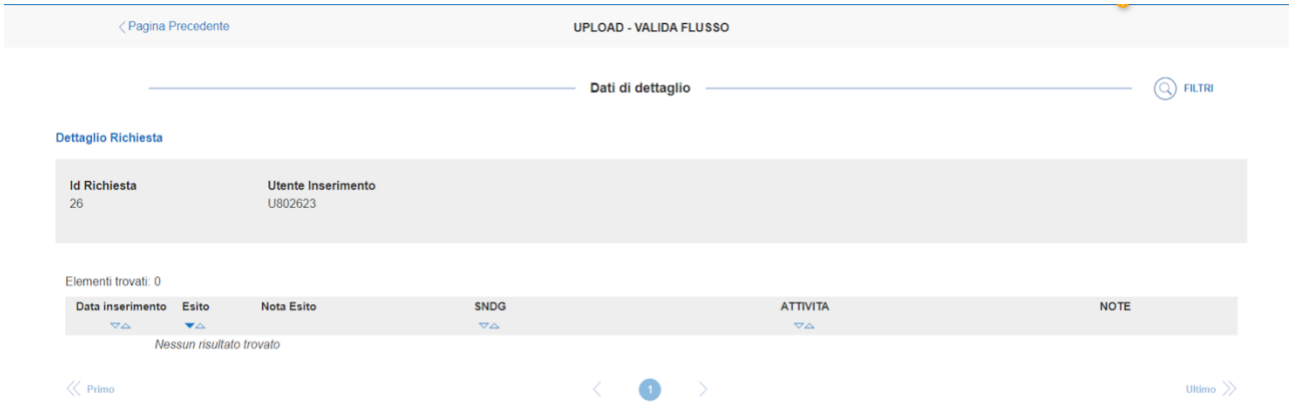

Figure 3.6 Defect's evidence. After imposing the "Attività" filter, the table is updated showing no result. We would have expected a table showing only one position: 0000000007524943.

The defect was assigned a medium priority, being a functional test. It was taken over and corrected by the developers; below is the defect's section in ALM [Figure 3.7].

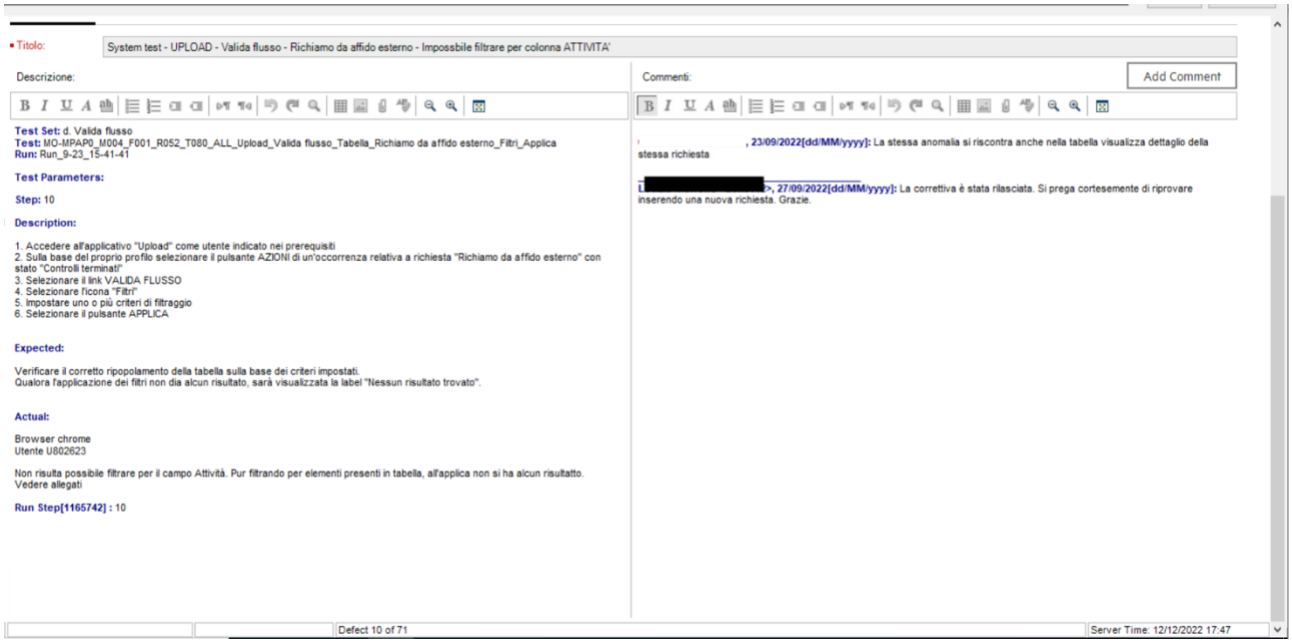

Figure 3.7 ALM - Defect ID 17075

2. Defect ID 17160

The defect is related to the test MO-MPAB0_M004_F002_R048_T166_Upload_Flusso input PGR_Invio richiesta_Assegnazione a Team_Posizione NON aperta.

The anomaly found concerns post-validation checks, when the uploaded request is processed by the legacy: the indicated positions went KO after Legacy validation. However, it was not possible to view the KO result note at the front-end, as it was not possible to read the entire message when mouseover over the field.

Furthermore, the KO message was not consistent with what was reported in the AFU for that particular position: according to AFU, the message should be “La posizione non è in perimetro PGR”; as we can see in Figure 3.8 in system the message reported was “RECORD SCARTATO: POSIZIONE NON PRESENTE IN PGR PER SNDG 9999999999999999”.

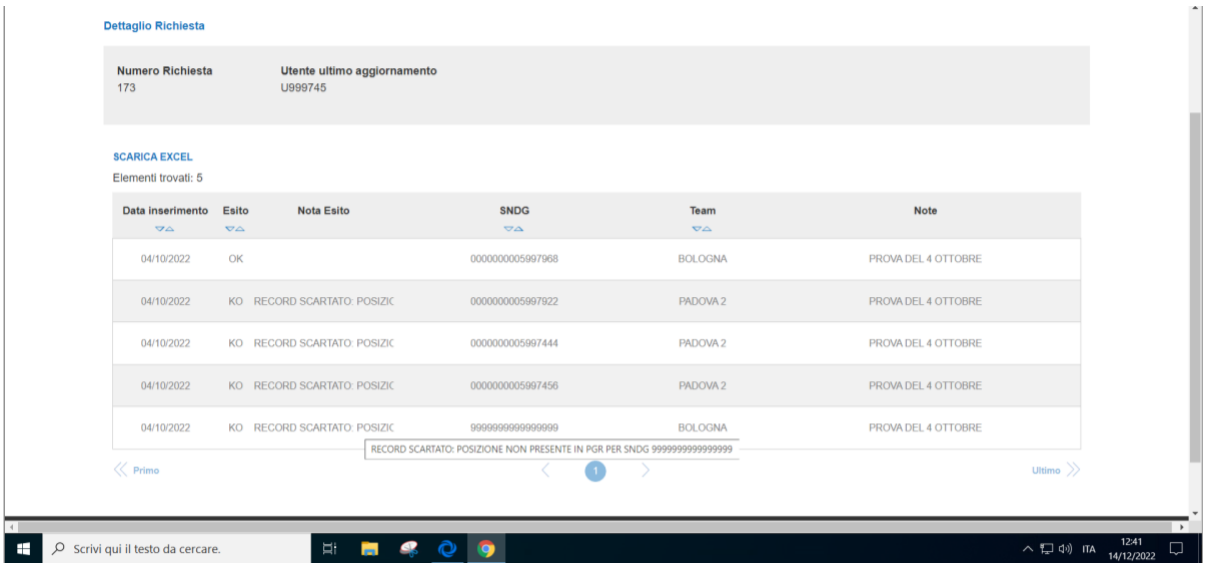

Figure 3.8. KO message - Error evidence

The defect was reported with high priority and severity, this was decided essentially for two reasons: the first is that the related test is a functional test that underlies the entire communication process between PGR and Upload. The second was also a question of timing: a delay in the delivery of the data preparation required for the test to be carried out led to the anomaly being detected the same day as the scheduled end of the system test.

Chapter 4. Software Testing project KPIs

The fourth chapter of the paper aims at a deeper analysis of the results obtained during Upload execution, outlined in the previous chapter, by means of the computation of critical KPIs for testing activities.
In order to do this, the first part will be dedicated to the introduction of the concept of measurement and to the theory of scale of measurement which lay the foundation to the concept of indicator, enabling the comparison of actual results with expected results.

4.1 KPIs and their impact on overall organizational performance

In order to get and keep competitive advantage over other market players in the same industry the manufacturing organizations must produce the quality products at lower cost. These are the few among many valuable objectives of the organizations. To differentiate oneself in the market means above all to be able to measure the quality of a performance; quality is in fact the ability to satisfy needs of any kind, in measurable terms.
In order to get confirmations regarding the fulfilling of their objectives and goals organizations have to keep check over their performance. To achieve these purposes organizations must have to use the performance management systems.
Simply the performance management is done by the organizations to confirm that either they are going in right direction or not. For measuring, managing, and comparing the performance the organizations are required to know about the performance indicators.
There is, however, an important criticality because companies tend to “become what they measure”, i.e., performance measures could become a factor influencing an organisation's behaviour, in this general framework it is easy to understand how the process of selecting the right performance indicators becomes extremely delicate [8].

Underlying the concept of indicators is the one of measurement. One of the most important pioneers of measurement theorists is probably the Harvard psychologist Stanley Smith Stevens, who defined measurement *“as the assignment of numerals to objects or events according to rules […] The fact that numerals can be assigned under different rules leads to different kinds of scales and different kinds of measurement*”.
According to S.S. Stevens and his Theory of Scale of Measurement, four types of measurement scales can be identified: nominal, ordinal, interval, and ratio scale. Before defining them in detail [Table 4.1], however, it should be pointed out that each type of scale is associated with a basic empirical operation, a permissible scale transformation, and a permissible statistic. The transformations that can be defined as "admissible" for a scale are those that leave the structure of the scale unchanged, i.e., if the relations of the numerical system do not change their properties following the application of the transformation that is being applied.

Scale type	**Basic empirical operation**	**Permissible scale transformation**	**Permissible statistic**
Nominal	Equivalence ($=$ or $\neq$)	Permutation (one-to-one substitution)	Mode, chi square
Ordinal	Order ($<$ or $>$)	Monotonic increasing function	Median, percentiles
Interval	Distance ($+$ or $-$)	Linear function $x' = ax + b$	Mean, standard deviation, correlation, regression, analysis of variance
Ratio	Ratio ($\times$ or $\div$)	Similarity $x' = ax$	Geometric mean, harmonic mean, coefficient of variation, logarithm

Table 4.1 Definition of measurement scales according to S.S. Stevens

The columns listing the basic operations and the permissible statistics are cumulative: to an operation/statistic listed must be added all those operations/statistics preceding it. Conversely, the column about the mathematical transformations which leave the scale-form invariant is "inversely" cumulative: each transformation in the column is contained in the transformation immediately above it.

Some measurement theorists, thanks to Stevens' work, have formulated a modern theory of measurement; it is known as the Representational theory of Measurement and it defines measurement as "*the assignment of numbers to properties of objects or events in the real world by means of an objective and empirical operation, in such a way as to describe them*" [9]. In other words, measurement is an operation of objective description of reality: different measurements of the same entity in the same operating conditions should result in the same output, independently from subjects.

Thus, as mentioned, measurement theory underlies the indicator concept which now can be considered as a mapping from an empirical system, the "real world", into a symbolic system, e.g., a numeric system. The indicator then becomes a way of observing the system and based on which decisions can be made. Most of the indicators of a generic process concern the following aspects [10]:

- Effectiveness: it indicates the ability to achieve a set of goals ("Are we doing the right things?");
- Efficiency: it assesses the ability to do so while employing the minimum essential resources ("Are we doing things right?");
- Customer care: the degree of satisfaction of process users.

Measuring is essential for the process-performance control and improvement. Performance measurement takes the form of a comparison between what may be permissible ranges of values in a given situation and the values actually examined.
The result of a performance measurement is a performance indicator, which is generally expressed by a number and a unit of measurement.
The performance indicators can be defined as the physical values which are used to measure, compare, and manage the overall organizational performance. They are a method for communicating the status of a product, a service, an organisation and more, to the outside world.
These indicators are not perfect measures, without errors or problem of definition and interpretation, but they are important pointers to the functioning of the system and keeping track of them is one aspect of quality control.
Moreover, choosing the wrong indicators could also be damaging for the organization; performance indicators will have an impact, there will be reactions both logical and emotional to the use of indicators, and these need to be considered in advanced and watched carefully in practice.

So, constructing and implementing a measurement system is easier said than done; the crucial point is to identify the “right” indicators to properly represent the process: i.e., the so-called Key Performance Indicators or more simply KPIs [Petersen et al. 2009].
KPIs have the clear purpose of certifying the achievement of the company or project goal as they can identify, with a certain degree of precision, the general picture that arises at the end of a project or after the realisation of a product.

4.2 Metrics used in testing process

We have seen how within the testing process monitoring activities play a major role. The use of appropriate KPIs is one of the ways through which the entire process can be monitored.
A KPI is a metric used to assess whether an organization is achieving its goals. By definition, not all metrics can be "key"; therefore, KPIs are a select group of metrics deemed essential to the achievement of business goals. These indicators help teams focus on the areas that generate the most value and have the greatest impact on business results.
IT metrics support KPIs by monitoring cost, performance, and output for IT processes. To be effective, they are often compared to established benchmarks, which provide context for the value of KPIs. In this way, IT metrics help determine differences between current and desired performance, track progress over time, and demonstrate how process improvements affect performance.

In Testing, metrics can be collected during and at the end of testing activities. The main purpose of metrics used in testing is to evaluate:

1. Progress with respect to the planned schedule and budget; remember that any deviation from the schedule must be justified and reported, as well as used to update the Test Plan.

2. The current quality of the test object; it seems obvious to say that an imperfect design of the work product being tested could cause slowdowns in testing activities.
3. The suitability of the testing approach; delays or deviations from the original schedule can be caused by using testing processes that are not appropriate for the current project.
4. The effectiveness of testing activities with respect to objectives; as with the testing approach, testing activities must also be appropriate with respect to objectives in order to avoid deviations from planning.

Common testing metrics include:

1. Percentage of test cases implemented compared to planned
2. Percentage of work done in preparing the test environment compared to planned
3. Execution of test cases, and thus number of test cases executed/not executed, test cases passed/failed, and/or test conditions passed/failed
4. Defect information, such as number of defects detected, or number of defects corrected
5. Test coverage in terms of requirements, user story, acceptance criteria, risk, or code
6. Completion of activities, allocation and utilization of resources, and related effort
7. Cost of testing, such as the cost versus benefit in detecting a new defect

4.3 System Test KPIs' construction

KPIs or Key Performance Indicators in software testing are some measurable values calculated to evaluate the efficiency and effectiveness of the software testing process as a whole.

Although it may seem natural to use KPIs to measure efficiency, there has been a dispute within the testing community over the years about the use of KPIs in the industry. Indeed, some argue that KPIs for software testing cannot help achieving the much-needed balance between time, cost, and quality, while others swear by their effectiveness.

Organizations have different products, so the development methodology and the testing processes are different too.
So, what a QA individual or manager needs is to:

- Understand the process
- Review the KPIs that can be measured
- Figure out the KPIs for quality that should be measured for maximum effectiveness according to your project.

Referring to system testing activities we will now go on to define some of the key parameters to be kept under control; then we will then be able to define some KPIs for testing, and again we will go on to identify and calculate those critical for the Upload project according to the project outcomes outlined in the previous chapter.
Among the parameters we identify:

- Total number of errors detected
- Number of errors detected before testing
- Number of errors detected during testing
- Number of errors detected before release
- Number of errors detected after release
- Number of errors fixed before release
- Number of errors fixed after release
- Number of total fixed errors
- Number of unsolved errors
- Test duration
- Duration of errors resolution
- Number of test cases per suite
- Total number of test cases per phase
- Total number of blocked tests
- Number of hours worked

The most common KPIs that are measured in the software testing industry [D]:

- Active defects: all defects that have yet to be closed are called active defects. There may be new defects open or resolved but not verified. The test manager must decide on a threshold value above which immediate action will need to be taken on how to proceed to reduce the number of functional defects. The general rule is that the lower the number of functional defects, the better the quality of the product at a given time. The presence of excessively large number of active defects could result in problems or delays in the execution of testing activities, evidence of which is the case study under discussion, as we saw in Section 3.2.

- Authored test: it measures the number of test cases designed during a defined time interval. This also helps measure the test cases against requirements.

- Covered requirements: This KPI is used to measure the mapping of test cases to requirements. A test manager is required to ensure that all the requirements have corresponding test cases, and action should be taken on any requirements that could not be mapped to any test case and vice versa. The goal is to keep the mapping of requirements to test cases to 100%.

- Passed tests: it is useful to understand the effectiveness of the test case design process, where test cases passed means the design is practical and vice versa.

- Rejected defects: this KPI measures the percentage of rejected defects compared to the total defects reported. If the percentage is higher than the set threshold value, the underlying issue must be identified and acted upon. This could mean more training for software testers or better requirement documentation.

- Severe defects: it aims to keep the number of severe defects in an application at a time under a limit; if there are more severe defects, then immediate action is needed. But before using this, the testing team needs to be properly trained to identify severe defects correctly.

- Defects fixed per day: This is used to measure the effectiveness of the development. However, it is subjective as some bugs could be more challenging to fix than others.

- Time schedule and constraints: This KPI is used to measure the average time of test execution. This helps provide testing time estimates during release planning or the development and testing plans to the project managers.

As mentioned above, we try now to identify some critical KPIs for the Upload project.
The scale of measurement referred to for the following KPIs is the ratio scale.

1. Passed test case coverage.
   It simply measures the percentage of passed test cases.

$$\frac{Number\ of\ passed\ tests}{Total\ number\ of\ tests\ executed}$$

2. Defect removal efficiency (DRE) or Fixed defects percentage before release.
   Indicator that can be placed in the set of quality indices. It provides a measure of the development team's ability to remove various defects from the software, prior to its release or implementation, by computing the percentage of errors resolved with respect to the total errors detected prior to release.
   Values between 0 and 1 can be found, of course the ideal value would be 1 as this would mean that all errors found prior to product delivery have been resolved.
   In some cases it is difficult to reach the maximum value, so values ≥ 0.9 are considered acceptable, although it is desirable that, especially in simpler projects, all errors are resolved.

$$\frac{Number\ of\ defects\ resolved\ by\ the\ development\ team}{Total\ number\ of\ detected\ defects}$$

3. Percentage of errors detected in the production environment and Defect Leakage.
   These indicators see themselves positioned among the quality indices. The percentage of errors detected in the production environment is useful as it gives an indication of the amount of errors detected after the final delivery of the software, when the acceptance tests take place.
   Defect leakage is used by software testers to review the efficiency of the testing process before the product's user acceptance testing (UAT). If any defects are left undetected by the team and are found by the user, it is known as defect leakage or bug leakage.

With reference to the project under consideration, they are not calculated as the term at the nominator, i.e., the errors identified after the final delivery of the product, is calculated by the customer itself, who together with the UAT team takes charge of the acceptance tests.

The KPIs can take values between 0 and 1, obviously the optimal value is 0, this would mean that no errors are detected after delivery of the software, ergo that the work of the tester has been carried out correctly, i.e., so that all errors are identified and corrected before release. It is clear that the optimal value always remains somewhat utopian; therefore, an acceptable value around ≤ 0.2 is identified.

$$\frac{Errors\ detected\ during\ UAT}{Total\ detected\ errors}$$

$$\frac{Errors\ detected\ during\ UAT}{Errors\ detected\ before\ UAT}$$

4. Percentage of unresolved errors.

   Again, a quality indicator, which determines the percentage of accepted errors, i.e., correctly detected by the tester, but not solved, compared to the total accepted errors. There are several possible reasons for which the client may decide not to resolve some defects: resolution would be too costly and time-consuming, the ratio of importance of the functionality in which the error was found and the cost is too unbalanced, the error concerns a functionality that is not inherent to the main purpose of the product and could therefore be eliminated.

   The KPI can take values between 0 and 1, obviously the optimum value is 0, this would mean that all errors detected have been resolved. An acceptable value is considered to be ≤ 0.1.

$$\frac{Total\ number\ of\ unresolved\ defects}{Total\ number\ of\ detected\ errors}$$

5. Average time to detect an error.

   The first KPI with its own unit of measurement, i.e., minutes/error and it is placed within the category of service indicators. It is also very useful for the purpose of improving the learning effect because it gives an indication of the average time of detection of an error, which may decrease as the tester gain confidence with the software. The values that can be found are certainly positive, but low values are preferred, which would mean that errors are detected in a very short time.

   As for the values that could be considered optimal, they may vary depending on the project.

$$\frac{Total\ duration\ of\ test\ phase}{Total\ number\ of\ detected\ errors}$$

6. Average defects resolution time.
   This indicator is also in the category of service indicators and from this we obtain information about the average time to solve an error. The total resolution time is calculated as the sum of the resolution times of the individual errors. Error resolution in the Upload project is delegated to the development team, but in any case, it is not the responsibility of the tester.

$$\frac{Total\ duration\ to\ solve\ an\ error}{Solved\ errors}$$

4.4 Calculation of KPIs based on the results obtained

After having defined some parameters, metrics and KPIs which can be useful in monitoring software testing activities; we now calculate some of the identified KPIs by referring to the results obtained during the execution of the PGR-Upload project [Table 4.2].

PARAMETER	FRONT-END	BACK-END	RESULT
Total number of test cases	126	52	178
Total number of detected errors	69	9	78
Total number of detected errors before testing	-	-	-
Total number of detected errors during testing	69	9	78
Total number of errors detected before release	69	9	78
Total number of errors detected after release	-	-	-
Total number of fixed errors	69	2	71
Total number of unresolved errors	0	7	7
Number of test cases per suite	Variable	Variable	Variable
Tests passed	126	45	171
Total duration test phase (days)	-	-	13
Working hours of the day	8	8	8

Table 4.2 Parameters obtained in execution phase

KPIs' computations:

1. Passed test case coverage.

   FRONT-END:

$$\frac{Number\ of\ passed\ tests}{Total\ number\ of\ tests\ executed} = \frac{126}{126} = 100\%$$

   BACK-END:

$$\frac{Number\ of\ passed\ tests}{Total\ number\ of\ tests\ executed} = \frac{45}{52} = 86.54\%$$

   TOTAL:

$$\frac{Number\ of\ passed\ tests}{Total\ number\ of\ tests\ executed} = \frac{171}{178} = 96.07\%$$

   Looking at the results, we can affirm the good outcome of the executions, which justifies the successful release of the software into production.

2. DRE, Defects Removal Efficiency.

   FRONT-END:

$$\frac{Number\ of\ defects\ resolved\ by\ the\ development\ team}{Total\ number\ of\ detected\ defects} = \frac{69}{69} = 100\%$$

   BACK-END:

$$\frac{Number\ of\ defects\ resolved\ by\ the\ development\ team}{Total\ number\ of\ detected\ defects} = \frac{2}{9} = 22.22\%$$

   TOTAL:

$$\frac{Number\ of\ defects\ resolved\ by\ the\ development\ team}{Total\ number\ of\ detected\ defects} = \frac{71}{78} = 91.03\%$$

   Regarding the indicator of unresolved defects, for FE we note a reassuring 100%, in fact as previously specified all defects related to the FE suite have been resolved.
   Not so reassuring, instead, is the 22.22% related to the BE suite. However, we have to think that the seven unresolved errors are not related to the implementation of the new software or its operations and functions, but they are errors that fall under the category of "Lack of data" concerning the data preparation carried out by the development team that did not allow the proper execution of the tests. In addition, looking at the total indicator, the value falls within the acceptable values specified in the previous paragraph.

3. Average time to detect an error.
   TOTAL:

$$\frac{Total\ duration\ of\ test\ phase}{Total\ number\ of\ detected\ errors} = \frac{13}{78} = 0.1667\ days/defect = 80\ min/defect$$

   Which also means that in average were detected 6 defects/day.
   The distinction into FE and BE suites was not made for this indicator. This was decided because the initial scheduling of first performing FE and then BE tests was possible to maintain only during the first few days of activity. The opening of numerous defects in the early days, which as already mentioned caused a two-day delay, caused the two phases of execution to overlap.

4.5 Properties of Indicators

The main problem when dealing with Indicators is the identification of the right indicators for relatively complex system. A practical tool for the identification, selection and evaluation of performance indicators are their properties, which help understand how to interpret and use them more effectively.

Indicators' properties can be classified into four main categories, which are summarized in the table below [Table 4.3] [10]:

Category	**Properties**	**Short description**
General properties	Consistency with the representation-target	It should properly operationalize the representation-target
	Level of detail	It should not provide more information than necessary
	Non-counter productivity	It should not create incentives for counter-productive acts
	Economic impact	It should be defined considering the expenses to collect the information needed.
	Simplicity of use	It should be easy to understand and use.
Properties of sets of indicators:	Exhaustiveness	Indicators of the set must treat all parts or aspects of the

A set or family of indicators is composed by the indicators selected to represent a generic process		system of interest, without omission.
	Non-redundancy	Indicators set should not include redundant indicators.
Properties of derived indicators	Monotony	The increase/decrease of one of the sub-indicators should be associated to a corresponding increase/decrease of the derived indicator.
	Compensation	Changes of different aggregated indicators may compensate each other, without making the derived indicator change.
Accessory properties	Long-term goals	Indicators should encourage the achievement of process' long-term goals.
	Impact on stakeholders/Customer satisfaction	For each indicator the impact on process stakeholders should be carefully analysed.

Table 4.3 Properties of indicators

Indicator properties are an important component of software testing process evaluation and improvement. Indicators are used to measure the performance and quality of the process and results, and indicator properties can help you determine whether an indicator is appropriate for your needs.

For example, the monotony property of an indicator means that the indicator will increase or decrease consistently in relation to changes in the process.

These properties are very useful because they can help determine whether an indicator is actually reflecting the true performance of the process and whether any improvements made in the process are actually reflected in the results. In addition, indicator properties can also be used to decide which indicators to use in a software testing process and how to use these indicators to continuously monitor and improve the process.

In summary, the use of indicator properties is critical to ensure that decisions made on the basis of indicator results are valid and reliable, and to provide a sound basis for continuous improvement of the software testing process.

To ensure an adequate understanding of what has been said so far, a more detailed analysis of the derived DRE indicator, introduced in the previous paragraphs, is given.

- Monotony.

The derived indicator should respond to variations in on or more sub-indicators, which mean we would expect the derived indicator to change following a variation of one or more sub-indicators.

$$DRE = \frac{Number\ of\ defects\ resolved\ by\ the\ development\ team}{Total\ number\ of\ detected\ defects} = \frac{DR}{TD}$$

Trying to verify monotony for DRE with respect to both of the sub-indicators DR and TD, we suppose a variation so that:

$$DR' = DR + \Delta(DR)$$
$$TD' = TD + \Delta(TD)$$

Now let us consider three scenarios realized during the execution of the ST PGR-Upload.

1. Scenario 1, 22/09/2022: at the end of the second day of execution, 34 defects have been detected but just 16 have been corrected.

$$DRE_1 = \frac{16}{34} = 0.47$$

2. Scenario 2, 23/09/2022: at the end of the third execution day, 19 new defects have been detected but, in total, 38 have been corrected by the development team.

$$DRE_2 = \frac{38}{53} = 0.71$$

3. Scenario 3, 26/09/2022: at the end of the fourth day, 5 new defects have been identified and 4 more defects have been corrected.

$$DRE_3 = \frac{42}{58} = 0.72$$

By analysing the scenarios presented in the early days of ST execution, it is easy to understand that the value of DRE over the course of a ST execution could increase or decrease depending on the performance of the development team.

In confirmation of this we can assume a new Scenario 4, in which ten new defects are identified but not corrected, so the value of DRE decreases from $DRE_3 = 0.72$ to $DRE_4 = 0.62$.

However, analysing its trend can be very useful because this property indicates whether the effectiveness in removing defects is improving or worsening over time. If DRE is monotonically increasing, it means that the amount of defects that are detected and fixed during the testing process is decreasing, which is a positive sign for software quality. On the other hand, if DRE is monotonically decreasing, it means that the number of defects is increasing, which may be a sign of problems in the testing process or software development.

It is also very useful to ensure that we do not exceed the open defect threshold value established in the planning phase. This way, if it is noticed a decreasing trend in the indicator, it may mean that the development is experiencing difficulties in resolving previously open defects. So, it could be necessary to slow down the executions for a moment and avoid signalling overloading.

In summary, we can say that during the execution phases the goal should be to keep the indicator monotonically increasing, as this is interpreted as an improvement in software quality.

- Compensation.

Given the derived indicator DRE of two sub-indicators, DR and TD, if:

- A variation $\Delta(TD)$ always causes a variation $\Delta(DRE)$, and if
- A variation $\Delta(DR)$ compensates the previous $\Delta(DRE)$, so that $DRE' = DRE$, then
- DRE fulfils the property of compensation and a substitution rate between the two sub-indicators can be univocally determined.

For the case study, imposing $DRE' = DRE$, which means $\Delta(DRE) = 0$, we obtain:

$$DRE' = \frac{DR + \Delta(DR)}{TD + \Delta(TD)} = DRE = \frac{DR}{TD}$$

From which we obtain the substitution rate:

$$\Delta(DR) = \frac{\Delta(TD)}{TD} DR$$

This data can prove to be very important especially for the development team. In this way, they can regularly set work goals based on the new number of defects identified and immediately identify how many to fix to maintain an adequate level of software quality.

Chapter 5. Statistical control methodologies

Now statistical control methodologies are analysed and proposed based on the results obtained.
Just as with the test cases, the control is carried out on a software application, and the purpose is thus to verify the proper functioning of the implemented functions.
The application of these methodologies goes to confirm how even software products are subject to strict quality control to this day.
Within factories two typed of quality control can be carried out: *Control during production* and *Acceptance control;* the latter is described in more details in Appendix A.

5.1 Single Sampling Plan

A simple sampling plan for the results of the ST Upload phase is now applied.
For this purpose, the totality of tests is considered as the population, with no subdivisions between FE and BE.
This assumption, allows the identification of approximately thirty-five sample of size five:

$$N = 178$$

$$n = 5$$

$$c = 1$$

For the construction of the simple sampling plan, the conventional method was not chosen.
The choice to use a small sample size and an acceptance number of one was not made at random but was dictated by very important considerations related to the quality of the software we are testing. This software is intended for a critical business management application for the banking institution, so quality is of utmost importance to ensure adequate support for the bank.
For this reason, it was decided to take a more rigorous approach to software testing and establish parameters upstream in the process to ensure that any identified defects are corrected before the software is delivered and acceptance testing begins. The acceptance number of one means that even one identified defect in the sample represents a problem for the software, and therefore must be corrected, ensuring that the software meets stringent quality requirements. The small sample size allows us to perform checks on a limited number of units, ensuring that each unit undergoes rigorous evaluation.
However, since this is an ST, the total population N consists of 178 test cases each of which plays a key role in ensuring a good level of quality and proper functioning of the software. Consequently, applying a sampling plan, in which decisions on the entire batch are made following inspection of a sample, increases the risk of making decisions by neglecting critical software functions. This risk was intentionally highlighted by choosing a small sample size for analysis of five.

For the selected values, the approximation of the hypergeometric distribution to the binomial is valid, so for the calculation of the probability of acceptance:

$$p = \frac{7}{178} = 0.04$$

$$Pa \cong 98.6\%$$

Looking at the results obtained, we can say that the customer would have positive feedback, given the small defectiveness and the resulting high probability of acceptance.

Moreover, another important measure is the total amount of inspection required by the sampling program in order to accept or reject the lot, ATI. This is a very important measure since it reflects the inspection cost:

$$ATI \propto inspection\ cost$$

If the lot quality is $0 < p < 1$, the average amount of inspection per lot will vary between the sample size n and the lot size N. If the lot is of quality p and the probability of lot acceptance is Pa, then the average total inspection per lot will be:

$$ATI = n + (1 - Pa)(N - n)$$

For the built-up case, it results: $ATI = 7.42$

If the lots contain no defective items, no lots will be rejected, and the amount of inspection per lot will be the sample size n. If the items are all defective, every lot will be submitted to 100% inspection, and the amount of inspection per lot will be the lot size N.
This is one of the reasons why we opted for the construction of a single sampling plan for the entire ST, without resorting to a split between the FE and BE phases; in addition to the overlap of the two illustrated in the previous chapters.
In fact, by splitting up the two phases, ATI would become:

$$ATI\ (FE) = n = 5$$
$$ATI\ (BE) \cong 11.6$$

ATI is strongly dependent on N and Pa, and consequently on p. The lower the defect rate, as in the case of FE, the more likely it is that batches will be accepted and the need for rectification will not arise. Conversely, the higher the defectiveness, the more likely it is that a 100% inspection will be required.

Therefore, it is evident that the inspection cost would be higher in the latter case. Applying acceptance sampling plan to ST activities, ATI becomes a very important measure since it would avoid budget waste.

In any case, using sampling plans for system testing activities of a software application may not be the wisest choice; just remember that each test case corresponds to a functionality, and therefore plays a key role in the proper implementation of software requirements. This makes the random choice of the sample much more critical. This aspect will be analysed in more detail in Section 5.3.

5.1.1 The OC curves

An important measure of the performance of an acceptance-sampling plan is the operating-characteristic (OC) curve. It displays the discriminatory power of the sampling plan by plotting the probability of accepting the lot versus the lot fraction defective.
A more in-depth analysis on this topic can be found in Appendix A.

The OC curve for the Upload project is now reported [Figure 5.1], based on the data given in Table 5.1.

p	n	Pa	Pa(%)
0	5	1	100,00%
0,01	5	0,9990	99,90%
0,02	5	0,9962	99,62%
0,03	5	0,9915	99,15%
0,04	5	0,9852	98,52%
0,05	5	0,9774	97,74%
0,06	5	0,9681	96,81%
0,07	5	0,9575	95,75%
0,08	5	0,9456	94,56%
0,09	5	0,9326	93,26%
0,1	5	0,9185	91,85%
0,2	5	0,7373	73,73%
0,3	5	0,5282	52,82%
0,4	5	0,3370	33,70%
0,5	5	0,1875	18,75%
0,6	5	0,0870	8,70%
0,7	5	0,0308	3,08%
0,8	5	0,0067	0,67%
0,9	5	0,0005	0,05%
1	5	0	0,00%

Table 5.1 OC Curve data construction

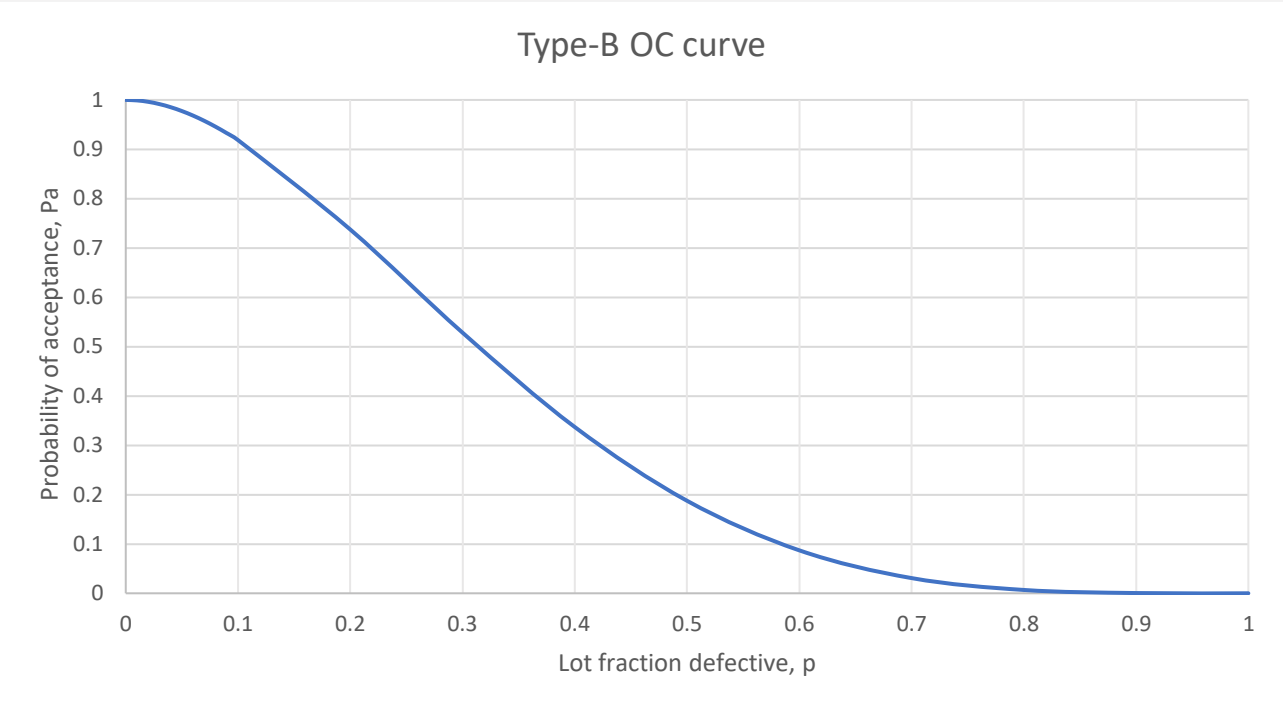

Figure 5.1 PGR – Upload project OC curve

So, in applying an acceptance sampling plan to ST activities we can consider the Test Factory as the supplier and the banking institution as the customer. The supplier, and thus the TF, is interested in the level of product quality that corresponds to a high probability of acceptance; the customer, on the other hand, is certainly interested in a good level of quality, but making sure that it does not result in an overly lenient probability of acceptance.

These values correspond to the fraction of defective AQL and LTPD, which are introduced in the theoretical section of this chapter in Appendix A; we have seen that AQL and LTPD are associated with an α and β risk level of the supplier and customer, respectively. In the case study, they represent:

- α: the risk that some of the functionalities that according to the TF are correctly implemented are rejected by the client.
- β: the risk that the banking institution accept some functionalities that although have been tested are not fully compliant with the initial requirements of the software.

Risk estimation is done at the planning stage by the Test Manager; however, for system test activities $\alpha = 10\%$ and $\beta = 12\%$ can be considered as an overall estimate.

All of the above outlines the conventional method for constructing a sampling plan that involves constructing a Type-B OC curve passing through two points; to determine the two points, the equation below needs to be solved so that the requirements of both the client and the supplier are met.

$$\begin{cases} 1-\alpha \;=\; \sum_{i\,=\,0}^{c} \binom{n}{i}\, AQL^{i}\,(1-AQL)^{n-i} \\ \beta \;=\; \sum_{i\,=\,0}^{c} \binom{n}{i}\, LTPD^{i}\,(1-LTPD)^{n-i} \end{cases}$$

To solve these equations, the Larsson's Nomograph is used, through which the value of n and c are determined.

5.2 Double Sampling Plan

DSP is procedure in which, under certain circumstances, a second sample is required before the lot can be sentenced. It is defined by four parameters:

$$n_1 = sample\ size\ on\ the\ first\ sample$$
$$c_1 = acceptance\ number\ of\ the\ first\ sample$$
$$n_2 = sample\ size\ on\ the\ second\ sample$$
$$c_2 = acceptance\ number\ of\ the\ second\ sample$$

Comparing the double sampling plan with the single sampling plan, one of the possible advantages is that it may reduce the total amount of required inspection; when the first sample taken under a double sampling plan is smaller than the sample that would be required using a single sampling plan that offers the consumer the same protection, then in the case in which a lot is accepted or rejected on the first sample, the cost of inspection will be lower for double sampling than it would be for single sampling. So, in DSP it is possible to reject a lot without complete inspection of the second sample. Consequently, the use of double sampling can often result in lower total inspection costs.
In addition, in some situations, a double-sampling plan has the psychological advantage of giving a lot a second chance.

However, this also falls under one of the possible disadvantages. Unless curtailment comes into play in the second sample, the double sampling plan may require more total inspection than the single sampling plan that offers the same protection. Therefore, unless it is used carefully, the potential economic advantage of the double sampling plan may be lost.

Now, just as it was for the single sampling plan, we will go on to apply the double sampling plan for the ST under study.
The values used for the construction of the double sampling plan are:

$$n_1 = 5$$
$$c_1 = 1$$
$$n_2 = 10$$
$$c_2 = 3$$

The choice of parameters follows a reasoning equivalent to that made for the construction of the simple sampling plan. It is apparent that the risk of excluding software-critical functionality is reduced if the inspection continues to the second sample; however, while this offers greater protection, it does not guarantee 100 percent coverage of requirements, which we have seen to be critical for a ST activity.

In DSPs, the total probability of acceptance Pa is thus given by the sum of the probability of acceptance on the first sample, Pa^I, and the probability of acceptance on the second sample, Pa^{II}.

$$Pa = Pa^I + Pa^{II}$$

Therefore, in our case study Pa results:

$$Pa = \underbrace{P^I(0) + P^I(1)}_{Pa^I} + \underbrace{[P^I(2)] * [P^{II}(0) + P^{II}(1)] + [P^I(3)] * [P^{II}(0)]}_{Pa^{II}}$$

Of which:

- $P^I(0) + P^I(1) = 98{,}6\%$ since it represents the probability of acceptance of the SSP that has been built up in section 5.2.

- $[P^I(2)] = \binom{5}{2} * \left(\frac{7}{178}\right)^2 * \left(1 - \frac{7}{178}\right)^3 = 0.0137$

- $P^{II}(0) = \binom{10}{0} * \left(\frac{7}{178}\right)^0 * \left(1 - \frac{7}{178}\right)^{10} = 0.6695$

- $P^{II}(1) = \binom{10}{1} * \left(\frac{7}{178}\right)^1 * \left(1 - \frac{7}{178}\right)^9 = 0.2741$

- $P^I(3) = \binom{5}{3} * \left(\frac{7}{178}\right)^3 * \left(1 - \frac{7}{178}\right)^2 = 5.613 * 10^{-4}$

- $Pa = P^I(0) + P^I(1) + [P^I(2)] * [P^{II}(0) + P^{II}(1)] + [P^I(3)] * [P^{II}(0)] = 99.8\%$

The disadvantage here is that the DSP is more administratively complex, which could lead to errors in inspection. In addition, the first sample has the same size as the SSP, $n_1 = n_{SSP} = 5$, consequently leading to an increase in the inspection cost in case the first sample does not discriminate the lot; in turn, it provides a higher level of protection against accepting lots with a high number of defects.

5.3 Observations on constructed acceptance sampling plan

The application of sampling plans to the PGR-Upload project was therefore proposed as a possible alternative to performing all the planned test cases, so as to avoid wasting resources, time and budget.
Indeed, as we have seen, the sampling plan could reduce execution time and inspection costs; however, care must be taken because these reductions are not synonymous with greater efficiency.

The main problem is that the decision is made by considering only part of the population, the one contained in the sample. Being in the context of software testing this becomes a not minor problem: in the Upload project each individual test case has the goal of going to evaluate the correct implementation of the functionality provided by the application, so it is not possible to evaluate the quality of the software based on a smaller number of tests.
This problem is even more emphasized for the plans constructed in the paper, as n was deliberately chosen small so as to propose a borderline case that does not allow enough information to be obtained, certainly leading to neglect critical functions for the client.

Because of what has just been said, therefore, it is clear that the SSP does not appear to be an appropriate method to replace the execution of individual test cases, as the client would incur too much risk.

The same, of course, applies to DSP, which also has another aggravating factor: although it provides a higher level of protection, which in any case is not enough in the field of software testing, the cost of inspection would be higher than the one for SSP, for the reasons pointed out in the previous paragraph.

However, there might be some ST activities where sampling plans can play the role of an alternative to running all test cases. For example, in STs that involve the implementation of the same functions in multiple applications, or the same features in multiple pages of the same application, then acceptance sampling plans could be used as an alternative to testing to avoid redundancy and drastically reduce the number of test cases to be executed and thus reduce execution time, resources, and budget.

Looking at the results obtained with the two sampling plans, we can see that in both cases the wastes obtained are really minimal.
However, in case we want to take action to reduce them even more, the following solutions are proposed:

- Build a more selective acceptance sampling plan, by increasing n or decreasing c, since a larger sample size increases the chances of finding a representative sample that meets the acceptance criteria. In particular, increasing n has a double positive effect:
    - For relatively low value of p, Pa tends to increase;
    - For relatively high value of p, Pa tends to decrease.

- Modify the value of the risks an α and β, which would mean the banking institution commissioning the tests should assume a higher risk of accepting a lot that does not comply with the requirements.

Chapter 6. Control charts for PGR-Upload project

We already stated that within factories two typed of quality control can be carried out: *Control during production* and *Acceptance control.* The latter has been described and applied in the previous chapter; now, we focus our effort in the first one and in particular to the role and application of Control Charts to the case study.
CC analysis is described in more detail in Appendix B.

6.1 Construction of Upload Control Chart

CC for attributes is used for the specific case study. The most widely used attribute control chart is definitely the p chart, which is employed to monitor and evaluate the quality of a process using the percentage of defective/non-conforming units, i.e., the ones considered to be noncompliant with the requirements for one or more quality characteristics.

In this context, it is important to highlight the difference between the concept of defects and the one of defectives. The former refers to any deviation from the desired or specified characteristic of a product or service. In other words, a defect is any attribute of a product or service that does not meet the required standards or specifications. The latter, on the other hand, refer to products or services that contain one or more defects.

The defect in the case of the project under consideration, with particular reference to the activity of testing, is considered the bug or error found in the functionality to which the test case refers. The adjective defective, on the other hand, refers to the test case that has at least one defect.

The p-chart is based on analysis of the distribution of the variable $\hat{p}_i$: the probability of having x number of defectives in the sample follows a binomial distribution.

$$\hat{\mathrm{p}}_i \sim B\left(p, \frac{p(1-p)}{n}\right)$$

When the process fraction nonconforming p is not known, then it must be estimated from observed data. The usual procedure is to select m preliminary samples, each of size n.
Then if there are Di nonconforming units in sample i, we compute the fraction nonconforming in the i_{th} sample as:

$$\widehat{p_i} = \frac{D_i}{n} \qquad i = 1,2,\ldots,m$$

From which the calculation of the average of these individual sample fractions nonconforming is obtained:

$$\bar{p} = \frac{\sum_{i=1}^{m} \widehat{p_i}}{m}$$

The statistics $\bar{p}$ estimates the unknown fraction of nonconforming.

The centre line and control limits of a p chart are computed as follows:

$$UCL_p = \bar{p} + L\sqrt{\frac{\bar{p}(1-\bar{p})}{n}}$$

$$CL_p = \bar{p}$$

$$LCL_p = \bar{p} + L\sqrt{\frac{\bar{p}(1-\bar{p})}{n}}$$

Depending on the values of p and n, sometimes the lower control limit $LCL < 0$. In these cases, we customarily set $LCL = 0$; since p is a probability, negative values are senseless. For this reason, control limits of p-chart can often be asymmetrical.

Now the construction of a p-chart for the Upload project is proposed. The procedure used to construct a control chart for attributes, as well as that for variables, involves the extraction of samples at regular time intervals, these k samples are used initially for the construction of the control chart; the samples extracted later, on the other hand, are used to monitor the process.
In this specific case the samples are considered of equal size to those constructed for the sampling plans so as to allow a more balanced comparison between the two methods.
Once the samples have been extracted, the defective units contained within each of them are identified; these will in turn allow calculation of the defective fraction defined as the ratio of the number of defective units in the sample to the sample's size.
Twenty-five samples of size five were considered for the construction of the control chart; the results obtained are shown in the table below [Table 6.1].

Sample	Sample size	Number of defective elements
1	5	3
2	5	4
3	5	3
4	5	3
5	5	2
6	5	3
7	5	1
8	5	2
9	5	1
10	5	2
11	5	1
12	5	1
13	5	2
14	5	3
15	5	0
16	5	3
17	5	1
18	5	2
19	5	1
20	5	3
21	5	1
22	5	0
23	5	0
24	5	0
25	5	0

Table 6.1 Data for CC construction

Using the data collected in the table, we go on to calculate the center line and control limits for the p chart:

$$\bar{p} = CL = \frac{42}{125} = 0.336$$

$$UCL = 0.336 + 3 * \sqrt{\frac{0.336 * (1 - 0.336)}{5}} = 0.9697$$

$$LCL = 0.336 - 3 * \sqrt{\frac{0.336 * (1 - 0.336)}{5}} = -0.2977 \rightarrow LCL = 0$$

This is an approximate representation of the control limits, since the binomial distribution cannot be perfectly approximated by the normal distribution in all cases. In this case, the approximation

may not be adequate and more specific methods may be needed to calculate more specific control limits of the p chart.

The control chart is shown below graphically [Figure 6.1]:

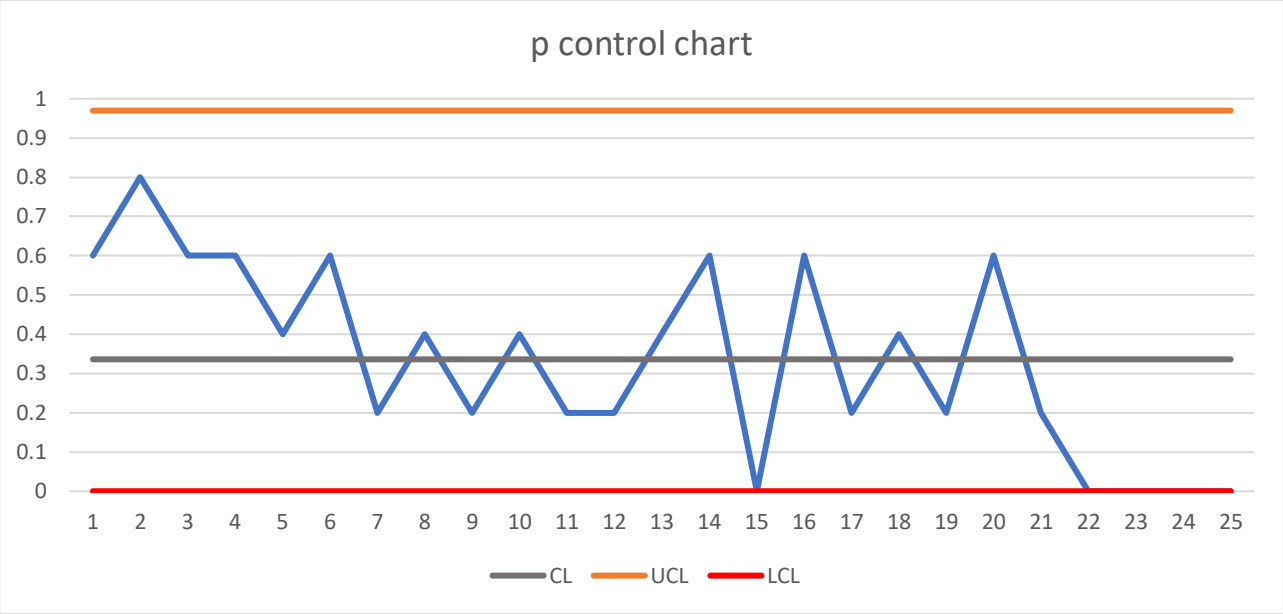

Figure 6.1 CC, p-chart for the ST activity

Indeed, it is important to specify that the choice of sample size n = 5 was made to keep related to the sampling plan constructed in the previous chapter; the choice of such a small size is not to be taken lightly, as it has several consequences on the control chart. Using a small sample size in a p control chart can result in unreliable results, which can lead to incorrect conclusions about the process stability and quality. In particular, it can result in problems such as:

- Insufficient data to accurately represent the process; a small sample size provides limited information about the process and can lead to false conclusions or incorrect decisions.
- Increased variability; a small sample size increases the variability of the sample statistics, which in turn can result in unstable control limits.
- Decreased statistical power; with a smaller sample size, it is more difficult to detect small shifts in the process mean, and therefore, the control chart may have less power to detect real process changes.
- Inaccurate control limits; control limits are based on the sample statistics, so a small sample size can result in inaccurate control limits that do not truly represent the process variation.

6.2 Analysis of CC results

A CC may indicate an out-of-control condition when one or more points fall beyond the control limits. By looking at the graphic representation of the p chart in Figure 6.2, the process seems to be in control since every point fall within the control limits.

However, an out-of-control condition may arise also when the plotted points exhibit some non-random pattern. Here, the problem is one of pattern recognition that is recognizing systematic or non-random patterns on the control chart and identifying the reason for this behaviour.
The ability to interpret a particular pattern in terms of assignable causes requires experience and knowledge of the process. That is, we must not only know the statistical principles of control charts, but we must also have a good understanding of the process itself.

This is where the Western Electric Handbook (1956) comes into our support. It suggests a set of decision rules for detecting non-random patterns on control charts. Specifically, it considers the process to be out of control if either [14]:

- One point plots outside the three-sigma control limits,
- Two out of three consecutive points plot beyond the two-sigma warning limits,
- Four out of five consecutive points plot at a distance of one-sigma or beyond from the centre line
- Eight consecutive points plot on one side of the centre line.

In addition to the above Western Electric Rules, Kendall's Turning Points Test is a randomness test which is also proposed for the pattern recognition problem.
This test consists of determining, for a sequence of points, the number of Turning Points (Tp) which is defined as:

$$Tp = \#P + \#V$$

Where, $\#P$ represents the number of local maxima and $\#V$ the number of local minima.
Under the null hypothesis Hp_0 that the sequence is random, it can be demonstrated that Tp is a random variable following a normal distribution.

$$Tp \sim N(\mu_{Tp}, \sigma_{Tp})$$

$$\mu_{Tp} = \frac{2}{3}(m-2)$$

$$\sigma_{Tp}{}^2 = \frac{16m-29}{90}$$

In Kendall's turning point test, the number of turning points in a sequence is used as a measure of randomness. The idea is that a random sequence is expected to have a certain distribution of turning points, which can be modelled and used to calculate a confidence interval for the expected number of turning points in a random sequence of a given length.
If the number of turning points in the sequence under investigation falls within this confidence interval, it is concluded that the sequence is random. In other words, the observed number of

turning points is consistent with what would be expected from a random sequence, and any observed patterns or trends in the data are assumed to be due to chance.
Conversely, if the number of turning points is outside of this range, it may indicate that the sequence is not random and that there is a systematic pattern or trend in the data. This could suggest that there is some underlying factor affecting the values in the sequence that is not accounted for by randomness.

Considering a 95% confidence interval, and a resulting risk $\alpha = 0.05$, we can compute the CI:

$$UCIL = \mu_{Tp} + z_{1-\frac{\alpha}{2}} \sigma_{Tp}$$

$$LCIL = \mu_{Tp} - z_{1-\frac{\alpha}{2}} \sigma_{Tp}$$

For the specific case the interval to be considered is the following:

$$\{11.35; 19.31\}$$

The Turning Point in the p chart reported in Figure 6.2 are:

$$Tp = \#P + \#V = 6 + 7 = 13$$

Because of $Tp \in [11.35; 19.31]$, Hp_0 cannot be rejected, the sequence can be considered as random and the process in control.

6.3 Process capability analysis

Control charts may also be used to estimate the parameters of a production process, and, through this information, to determine process capability. The control chart may also provide information useful in improving the process.

Process capability analysis is the assessment of a process's ability to meet customer specifications or requirements. In a control chart, process capability analysis can be performed by comparing the process performance to the specification limits.

In general, there is no connection between the control limits of the charts and the specification limits; control limits are driven by the natural variability of the process, specification limits are determined externally, by designers, management, or manufacturing engineers.

Any process can be in four different states:

1. Within specification limits and in control (ideal state).

2. Outside specification limits and in control: it represents the worst-case scenario, since the process is governed by natural variability, but the production output does not meet specifications. It is necessary to intervene to improve the process, or even change it in the attempt to reduce its variability; another possible action may be the one of increasing the specification limits, which are too stringent for this level of variability.
3. Within specification limits and out of control: it provides lack of statistical control, but the process does not produce a meaningful number of defectives. A possible corrective action may be undertaken to identify the assignable cause and reduce the process variability.
4. Outside specification limits and out of control: the process is both out of specifications limits and out of control, assignable causes need to be identified to establish control. This scenario is better the one presented at point 2 since there is still room to improve and solve the problem [15].

One way to express in a quantitative way process capability are the so-called Process Capability Ratios. Two leading factors tend to increase the chance to produce scrubs: bad centring and large variability.
Numerous statisticians and quality engineers, such as Kane in 1986, have emphasized research into process capability indices to propose more effective methods of evaluating process potential and performance [16].
The two main indexes are presented, c_p and c_{pk}.

The first one represents a sort of comparison between the specifications and the range of variability. It is defined as

$$c_p = \frac{S}{NT} = \frac{USL - LSL}{6\sigma}$$

And it is based on two assumptions:

- Hp_1: process is governed by only random sources of variability; hence, the process is in control.
- Hp_2: the quality characteristic has a normal distribution; $NT = 6\sigma$.

In general, $c_p > 1$ is considered to be a good value. Nevertheless, when c_p is too large the process may be unnecessarily expensive since $NT \ll S$.
Anyway, a typical reference value is $c_p \geq \frac{4}{3}$.

This index has a drawback: it does not take into account the process centring. To avoid this risk, it is introduced c_{pk}. In this case, two indexes are computed by considering the unilateral specification limits, c_{pu} and c_{pl}.

$$c_{pu} = \frac{USL - \mu}{3\sigma}$$

$$c_{pl} = \frac{\mu - LSL}{3\sigma}$$

c_{pk} is defined as the minimum of the two value just shown:

$$c_{pk} = \min(c_{pu}, c_{pl})$$

6.4 Observations on constructed p chart

In this paper, we used a control chart by attribute to control and monitor a testing process; going to consider only dichotomous values, defective/not defective. A possible alternative may have been to use CC by variables for each individual test case, monitoring as quality characteristics for example the average execution time.

However, using control charts by attributes has the advantage of being able to consider multiple quality characteristics together, and classify the unit as defective if it does not meet specifications for at least one of them.
Another advantage of the CC by attributes is that expensive and time-consuming measurements may sometimes be avoided by attributes inspection; moreover, they are usually characterized by an ease of interpretation and collection of the data needed to construct the control chart.

The disadvantage is that they provide much less information with respect to CC by variables; this leads to the need of sample with a higher size. As already specified for the acceptance sampling plan, this is the crucial problem for the Upload case, and more in general for the testing process.

The main problem that arises when we apply a CC to the execution phase of a testing process is to identify a good method of sampling. Difficulty arises since CC by attributes, which as we saw are the most suitable for the case study, require samples composed of hundreds of elements while in our particular case we only have 178 units to sample.

To give a numerical overview of what has been said so far, we can rely on the Generalised Duncan's Formula.
It is used in the construction of the p-control chart to determine the most suitable sample size for the case study; based on the assumption that the binomial distribution is approximate to the normal distribution.
The Generalised Duncan's Formula assumes the occurrence of a shift in the mean p, and imposes a β probability of not detecting such a shift.

Let's assume that some difficulties or problems occur in the software development during the execution of the tests, so that central line shifts from p to $p' = 0.6$. Since this is a software testing activity, we impose $\beta = 10\%$ because such a shift means that the number of defects has increased

significantly, leading to the need to notice it as soon as possible to ensure a good level of quality in the software.
Based on the above, the formula shows us that:

$$n = \left(\frac{L\sqrt{p(1-p)} - z_{\beta}\sqrt{p'(1-p')}}{p' - p} \right)^2 \cong 59$$

It is evident that with such a sample size, which include more than 33% of the population N, it would not be advisable to construct a control chart for the process under consideration.

The problem we analysed concerning sampling has a critical consequence in the construction of the control chart because control limits are strongly influenced by sample size. Thus, the control limits are evaluated through a small number of sampled values, and they may lead to unreliable results.
A possible alternative may be to use control charts with variable limits, so that a more accurate representation of the process performance can be obtained. However, the costs would become even greater; therefore, we can conclude that the control chart is not advisable as a control method for a testing process characterized by a not too large number of test cases.

Chapter 7. Conclusions

In the course of the paper, light was shed on the new reality that companies are undergoing, which is increasingly focused on the customer and the satisfaction of their needs and requirements. Innovation and digitalization play a primary role in today's world; technology is playing an increasingly important role in the life of every individual and therefore every company.

In this global context, software and applications have been developed and emerged in all areas, such as banking in which Internet Banking may be a prime example.
In addition, proper and functional business management is a crucial aspect of every business, regardless of its size and operating sector. For this reason, it has become essential to provide one's business with ad hoc tools capable of optimizing and simplifying certain business processes. In this perspective, management software is a very powerful tool for improving a company's productivity and competitiveness.

The PGR-Upload project fits perfectly into this framework. We have seen how credit recovery activities have always been a concern for banking institutions, hence the development of what are the specifications and needs of our client: to create a system that would support them in the management and organization of individual positions with potential for impairment, so that they would have support in decision-making and management strategy.

Whatever type of software it is, the primary goal must be to function properly and meet well-defined needs. This is where Outsourced Engineering companies are called in, providing ongoing support to companies of all types and operating in all fields. When we talk about consultancy, we are referring to professional advisors to help companies achieve their goals, supporting the client and managing their needs in order to best implement the required specifications within the product itself.
Our client, specifically, rely on outsourced services to test the functionality this type of applicative needs, and ensure that each one is implemented and works properly before proposing it to users or release it into production.
The relationship between a consultant and a client is built on trust, and it is important for the consultant to maintain this trust by providing quality service, maintaining confidentiality, acting in the best interests of the client, and always keep him updated about the work progress.
In the testing process, monitoring and reporting play a primary role, especially since those commissioning the tests need to be informed of progress and outcomes at regular, even daily, intervals.
This thesis work aims to do exactly that, as well as to show the testing process aimed at ensuring the proper functioning of the software, propose some statistical control methods to support the execution stage and show the results to the customer, confirming the company's strategies of putting the customer at the centre.

So, following an introduction of the application to be tested, two statistical control methodologies are proposed in an attempt to support and in some way facilitate communication with the client, but also within the test team itself, so as to ensure the effectiveness of execution and the quality of the tested product.

The first method proposed is the application of acceptance-sampling plans, which although found to be inadequate as an alternative to the execution of individual test cases, could play an important role in reporting execution.
Carrying out acceptance-sampling plans during executions could provide important data regarding the results, allowing the isolation of those test samples that need more attention due to the presence of a critical number of defects.
As mentioned, however, this method would prove capable of achieving comprehensive and reliable results for ST cases where the number of test cases to be executed is very large.

The second method proposed is the control chart. Control charts are among the most sophisticated methods for checking the stability of a process; however, in our case study it is found to be completely unreliable due to the low size of the total population. Moreover, for a process to be considered stable with certainty it must have limited variability. Controlling and limiting the variability of a process requires the deployment of numerous resources.
Finally, control charts analyse the entire testing process; however, the purpose of performing a test phase is to evaluate the implementation of software functionality, so it would be more appropriate to focus on individual functions than on the entire process.

All the considerations made turn to determine, support and ensure product quality; in our specific case of the PGR-Upload project, it refers to Software Quality Assurance.

As for PGR portal project, the next steps in the development of the application currently include two new integrations: Clienti Exclusive and Torre di controllo.
The former involves the implementation of a new queue “Attività filiale in Corso – Clienti Exclusive”, a new corresponding alert, and implements changes to the routing and workflow engine to handle exclusive customers’ positions.
The latter involves the implementation of a new application, whose access tab is present on the PGR homepage, just like for the Upload application. The goal of the evolutions in the assignment engine area is to revise the structure, integrate the logic and functionality of the control tower, as well as introduce new assignment rules for different kinds of positions, and add new guiding drivers that determine portfolio composition. In addition, a registry section for teams and specialists is added to make the user more autonomous in their census.

In connection with PGR, a new project has also been launched: Tool Gestione Cessioni. Unlike the first two, this is a stand-alone application, but one that also works in cooperation with the portal.

The application has functionality to enable end-to-end process management of portfolio sales transactions.
In the management of impaired positions, banks may indeed often opt to sell these positions; this is done for several reasons, such as risk reduction, as the disposal of these positions allows banks to transfer the associated risk to other entities; to implement an improvement in the current balance sheet or to focus on the core business.
The new application will allow the user to upload portfolios of positions destined for sale via a new type of Upload request, following the process described in this paper. The different portfolios will be uploaded to the new tool in which various changes and actions can be performed via the provided functionalities.

For all three projects, the design phase has been completed, so the data preparation is now needed to start execution on the day scheduled for each.

Appendix A. Acceptance control

In the development of the work, we have defined quality as the degree to which a product/service is conform to its requirements. Every product possesses several characteristics that are critical to quality, which take the name of quality characteristics.
Anyway, most organizations find it difficult and expensive to provide the customer with products that have quality characteristics which are always identical from unit to unit. A major reason for this is variability, no two units of product produced by a process are identical. Some variation is inevitable.
Variability is not totally removable, so some degree of variability may be considered tolerable, or physiological, for a given production process. This type of variability is also referred to by the name "natural variability", which can be defined as "*the process tendency towards producing in regular conditions product with quality characteristics different from target values"*.

Between 1920 and 1945, techniques for statistical control of the output quality were developed thanks to George D. Edwards and Walter A. Shewhart. Control techniques were introduced over the entire production process, no longer being limited to checking products for defects only at the end of the process since sweeping checks on all products was beginning to prove too costly.
To carry out this new type of inspection, there was increasing use of statistical criteria. By examining a few finished products, it was possible to determine, as they were being produced, whether the process had any irregularities or not.

This is where the concept of Statistical Control Methodologies come from: those methods which can be applied to a process to help identify and control sources of variability in the process and make it possible to control quality characteristics during production, in order to maintain the process under-control and to detect and correct possible abnormalities.

Generally, within factories two types of Quality Control are carried out:

1. Control during production: it is a real-time control of the output product, to check whether the process is in control and specifications are satisfied; this is done using control charts.
2. Acceptance control: it consists of the inspection of the product received from the supplier followed by the decision whether to accept it or not. There are three different approaches: 100% inspection, Acceptance sampling and Accept with no inspection.

Acceptance sampling is therefore concerned with inspection and decision-making regarding products, one of the oldest aspects of quality assurance.
It allows a company to estimate the quality of a batch of products by selecting a specified number for testing. The quality of this designated sample will be viewed as the quality level for the entire group of products.
It is important to assess though that acceptance-sampling plans do not provide any direct form of quality control; it simply accepts and rejects lots.

There are several different ways to classify acceptance-sampling plans. One major classification is by variables and attributes. Variables**,** of course, are quality characteristics that are measured on a numerical scale. Variable plans are used as an alternative to attribute plans when measurable variables are available; these allow a broader view of the process being analysed because they provide more information about those processes; however, the disadvantage is that there is a need for a sampling plan for each controlled variable.
Attributes are quality characteristics that are expressed on a "go, no-go" basis, such as identifying a batch within which defective items are then reported [4].
It becomes evident that for our software testing case study it may prove more useful to analyse those by attributes in more detail.

Thus, the construction of a sampling plan involves the extraction of a sample of size n from a batch of size N. The following symbols are adopted:

- N: total number of the units making up the lot, lot size
- D: number of defectives in the lot
- P: percentage of defective elements of the lot, i.e., $P = D/N$
- n: sample size
- c: acceptance number, if there are c or fewer defectives in the sample, accept the lot, otherwise reject it
- d: number of defectives of the sample

Acceptance sampling involves both the producer, or supplier, of materials and the consumer, or buyer. Since this is statistical sampling, on this operation lies the risk that the supplier may see lots rejected that should be accepted and the risk that the customer may see lots accepted that actually should be rejected; therefore, the percentage agreed upon must be such that it meets the specifications imposed by both parties. Consumers need acceptance sampling to limit the risk of rejecting good-quality materials or accepting bad-quality materials. Consequently, the consumer, sometimes in conjunction with the producer through contractual agreements, specifies the parameters of the plan. Any company can be both a producer of goods purchased by another company and a consumer of goods or raw materials supplied by another company [11].
Once the criteria for defining the lot are defined, we move on to the specifications required by the supplier and client.

Two levels of quality are considered in the design of an acceptance sampling plan. The first is the Acceptable Quality Level, AQL. This represents the poorest quality level that a consumer would consider to be acceptable as a process average, and it is specified by the supplier. This value is associated to a risk α to be refused a good lot.
The second level of quality is the Lot Tolerance Proportion Defective, LTPD, or the worst level of quality that the consumer can tolerate. LTPD value is specified by the consumer. This value is associated to a risk β to accept a bad lot.

A sampling plan is a kind of hypothesis testing, where the null hypothesis H_0 is that $p = AQL$:

	H_0 is True, p = AQL	*H_0 is False and p = LTPD*
Probability to accept H_0	1- α	β
Probability to reject H_0	α	1 - β

Table A.1 Sampling plan as a hypothesis testing

The probability that a lot is accepted is denoted by Pa, this probability is defined as the cumulative of the hyper-geometric distribution, i.e., a sample is drawn from a finite population and accepted so if there is a maximum number of elements within that sample (c) [12].

$$Pa\ (N, n, c, p) = \sum_{i=0}^{c} \frac{\binom{N - np}{n - i}\binom{Np}{i}}{\binom{N}{n}}$$

Consequently, the system of equations to construct the sampling plan becomes:

$$\begin{cases} 1 - \alpha\ = \sum_{i\,=\,0}^{c} \dfrac{\binom{N\ -\ n\,*\,AQL}{n\ -\ i}\binom{N\ *\ AQL}{i}}{\binom{N}{n}} \\ \beta\ = \sum_{i\,=\,0}^{c} \dfrac{\binom{N\ -\ n\,*\,LTPD}{n\ -\ i}\binom{N\ *\ LTPD}{i}}{\binom{N}{n}} \end{cases}$$

Whenever $\frac{n}{N} \leq 0.1$ the hyper-geometric distribution can be approximated by a binomial distribution as it is like thinking of drawing the sample from a lot of infinite size: $H \longrightarrow B$

$$Pa(n, c, p) = \binom{n}{i} p^i (1 - p)^{n-i}$$

The two main types of acceptance sampling plans by attribute are now presented: *Simple Sampling Plan* and *Double Sampling Plan*.

1. Single sampling plan: it is a decision rule to accept or reject a lot based on the results of one random sample from the lot. The procedure is to take a random sample of size n and inspect

each item. If the number of defects does not exceed a specified acceptance number c, the consumer accepts the entire lot. If the number of defects in the sample is greater than c, the consumer subjects the entire lot to 100 percent inspection or rejects the entire lot and returns it to the producer.

This procedure is called a single-sampling plan because the lot is sentenced based on the information contained in one sample of size n.

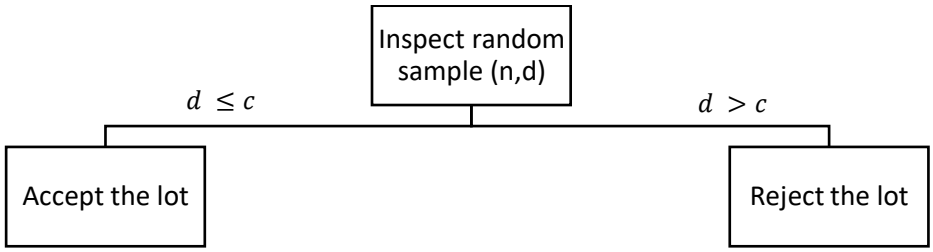

Figure A.1 Single sampling plan decisions' scheme.

2. Double sampling plan: In a double-sampling plan, two sample sizes, n_1 and n_2, and two acceptance numbers, c_1 and c_2, are specified. If the quality of the lot is very good or very bad, the consumer can decide to accept or reject the lot on the basis of the first sample. To use the plan, the consumer takes a random sample of size n_1. If the number of defects is less than or equal to c_1, the consumer accepts the lot. If the number of defects is greater than c_2, the consumer rejects the lot. If the number of defects is between c_1 and c_2, the consumer takes a second sample of size n_2. If the combined number of defects in the two samples is less than or equal to c_2, the consumer accepts the lot. Otherwise, it is rejected.

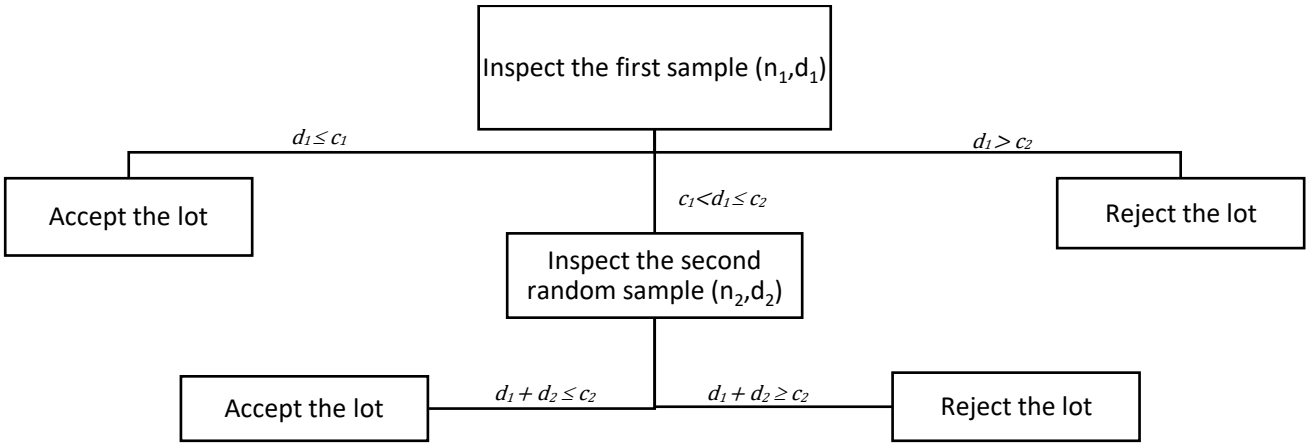

Figure A.2 Double sampling plan decisions' scheme

When lots are rejected, acceptance sampling programs may require corrective actions. This generally takes the form of 100% inspection or screening of rejected lots, with all discovered defective items either removed or return to the supplier. Such sampling programs are called *Rectifying inspection*.

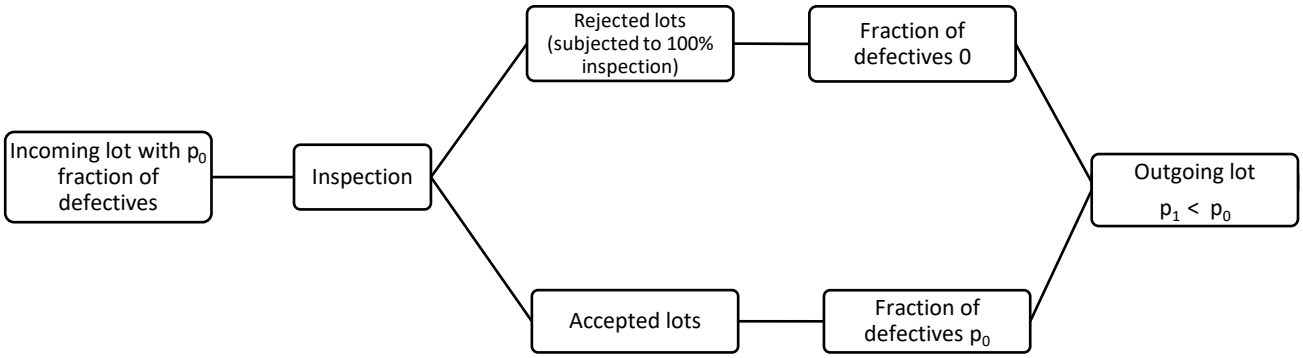

Figure A.3 Rectifying inspection decisions scheme

To measure the performance of a sampling plan, operating- characteristic (OC) curve comes into play. It displays the discriminatory power of the sampling plan by plotting the probability of accepting the lot versus the lot fraction defective.

We can distinguish two type of OC curves:

- Type-A OC curves are used to calculate probabilities of acceptance for an isolated lot of finite size, which means when the sampling distribution of the number of defective items in the sample is the hypergeometric distribution.
- Type-B OC curves [Figure A.4] are used when the binomial distribution is the exact probability distribution for calculating the probability of lot acceptance, which means when samples come from a large lot or that we were sampling from a stream of lots selected at random from a process, and therefore when $\frac{n}{N} \leq 0.1$. When this condition is met the type-A and type-B OC curves are virtually indistinguishable [Figure A.5].

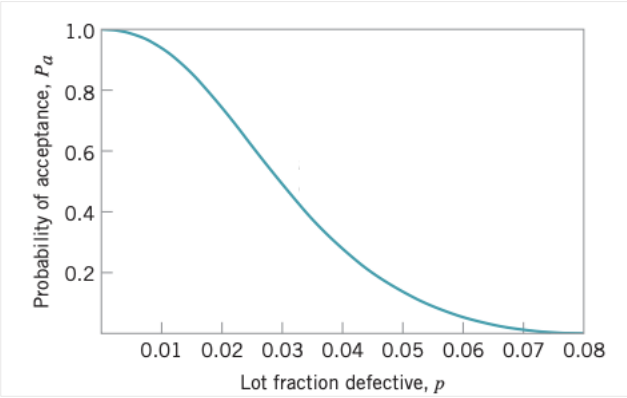

Figure A.4 Type-B OC curve

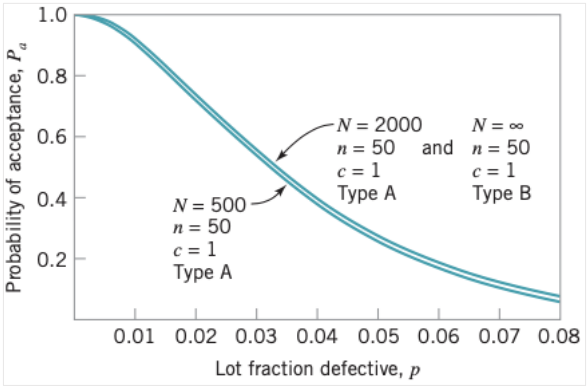

Figure A.5 Type-A and Type-B OC curves when $n \ll N$

A sampling plan that discriminates perfectly between good and bad lots would have an OC curve that looks like Figure A.6, which takes the name of Ideal OC curve.

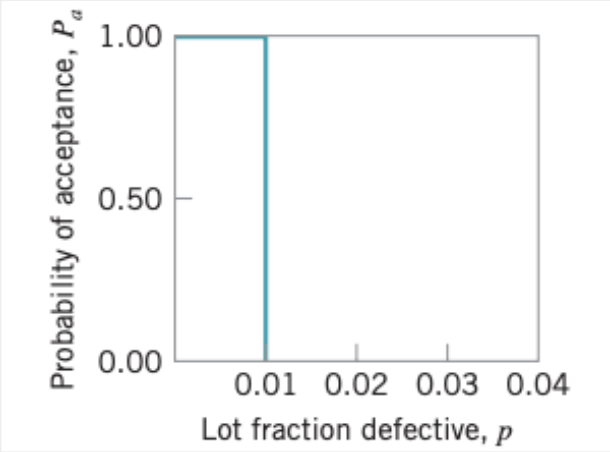

Figure A.6 Ideal OC curves

The Ideal OC curve runs horizontally at a probability of acceptance Pa = 1 until the lower acceptable level of quality is reached, at which point the curve drops vertically to a probability of acceptance Pa = 0, and then the curve runs horizontally again for all lot fraction defectives.
Anyway, the ideal OC curve is very difficult to obtain in practice; but it can be approached by increasing the sample size n [Figure A.7].

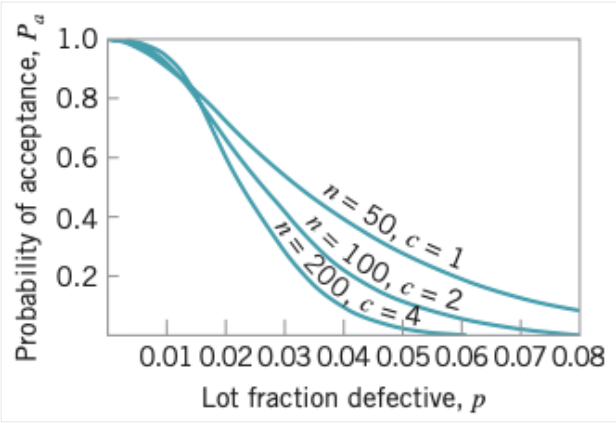

Figure A.7 Sample size effect on OC curve

By decreasing the acceptance number c, the OC curve shifts to the left becoming more selective: plans with smaller value of c provide discrimination at lower levels of defectiveness than do plan with larger values of c [Figure A.8].

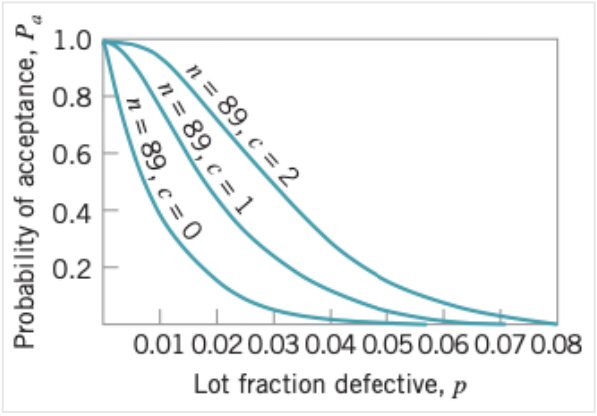

Figure A.8 Acceptance number effects on OC curve

However, it should not be assumed that sampling plans with c = 0 make it easier to detect samples with zero defects.

Plans with zero acceptance numbers have OC curves that have a different shape than the OC curves of sampling plans for which c > 0: they are convex curves and because of this shape, the probability of acceptance drops very rapidly even for small values of the lot fraction defective, which make more likely to reject good quality lots [Figure A.9]. This is extremely hard on the supplier, and in some circumstances, it may be extremely uneconomical for the consumer.

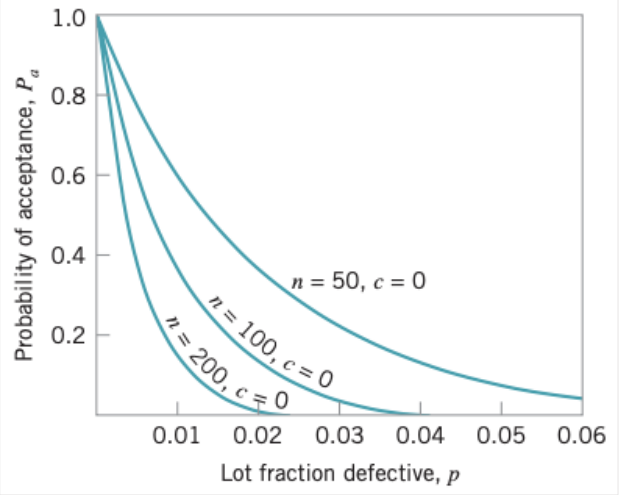

Figure A.9 OC curves with c = 0

Appendix B. Statistical basis of the Control Chart

In the course of the paper, we have repeatedly emphasised how the global context has led to a shift from a focus on manufacturing to a focus centred on customer needs.
Hence the importance of adhering to set specifications and tolerances based on customer needs and preferences.

As we anticipated at the beginning of Appendix A, in any production process, regardless of how well designed or carefully maintained it is, a certain amount of natural variability will always exist.
More precisely, the process must be capable of operating with little variability around the target or nominal dimensions of the product's quality characteristics.
That's why statistical quality control techniques play a paramount role in many manufacturing and service industries; among the SPC tools, the control chart is probably the most technically sophisticated. It was developed and proposed by Walter Shewhart in the 1920s and 1930s.

One of the main purposes of control charts is to distinguish between the variation due to chance causes and the variation due to assignable causes in order to prevent overreaction and underreaction of the process [13].
A process that is operating with only chance causes of variation present is said to be in statistical control. In other words, the chance causes due to natural variability are an inherent part of the process.
As a matter of fact, other kinds of variability may occasionally be present in the output of a process; for example, due to improperly adjusted or controlled machines, operator errors, or defective raw material. We refer to these sources of variability that are not part of the chance cause pattern as assignable causes of variation. A process that is operating in the presence of assignable causes is said to be an out-of-control process.

No process is truly stable forever, and, eventually, assignable causes will occur, seemingly at random, resulting in a shift to an out-of-control state where a larger proportion of the production will fall outside the specification limits, which means that the process output does not conform to requirements.

So, the main objective of Control Charts is to quickly detect the occurrence of assignable causes of process shifts so that investigation of the process and corrective action may be undertaken before many nonconforming units are manufactured.

A typical control chart is shown in Figure B.1; it is a graphical display of a quality characteristic that has been measured or computed from a sample versus the sample number or time.
A Control Chart contains:

- A centre line, CL
- An Upper Control Line, UCL

- A Lower Control Line, LCL

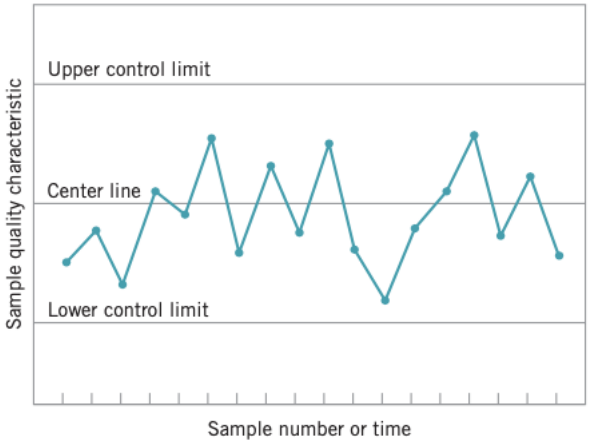

Figure B.1 Typical CC

The basic criteria states that:

- A point that plots within the control limits represents a necessary but not sufficient condition for saying that the process is in control; therefore, that no corrective action is necessary;
- A point that plots outside the control limits is evidence that the process is out of control; investigation and corrective action are required to find and eliminate the assignable cause or causes responsible for this behaviour.

Specifying the control limits is one of the critical decisions that must be made in designing a control chart. By moving the control limits farther from the centre line, we decrease the risk α of type I error. However, widening the control limits will also increase the risk β of type II error. They respectively represent:

- α: the risk of a point falling beyond the control limits, indicating an out-of-control condition when no assignable cause is present; in other words, it represents the probability of a False alarm;
- β: the risk of a point falling between the control limits when the process is really out of control.

It is important to note that CC make it possible to identify when a process is out of control, not out of specifications. The two conditions are not related; a process can be out of control but within specification limits, typically when $NT \ll S$. Just like a process can be in control but out of specification limits, typically when the process natural variability is too large [4].

We may give a general model for a control chart. Let w be a sample statistic that measures some quality characteristic of interest and suppose that the mean of w is μ_w and the standard deviation of w is σ_w. Then the center line, the upper control limit, and the lower control limit become:

$$UCL = \mu_w + L\sigma_w$$
$$CL = \mu_w$$
$$LCL = \mu_w - L\sigma_w$$

Where "L" is the distance between the two limits and the central line in terms of standard deviation units. Generally, these limits are within ± 3 mean square deviations from the statistical measure of interest, since Control chart theory is based on the assumption that the controlled variables follow a normal distribution.

Moreover, there is a close relationship between CC and hypothesis testing. Hp_0 is that the process is in control state; and two possible outcomes are available:

1. Failing to reject Hp_0; and the process is considered in control.
2. Reject Hp_0; the process is out of control and corrective actions are necessary to identify the assignable causes.

There are two main families of control charts:

1. CC for Variables: these are used to monitor continuous data and quality characteristics measured on a continuous numerical value. They include:
   - $\bar{x}$, Mean control chart
   - R, Range control chart
   - s^2, sample variance control chart
   - s, sample standard deviation control chart
   - x_i, control chart for individual measurements

2. CC for Attributes: these are used to monitor discrete data, such as defective/non-defective or conforming/non-conforming. They include:
   - p, control chart for the percentage of defective units
   - np, control chart for the number of defective units
   - u, control chart for the number of defects per unit
   - c, control chart for the number of defects per sample

References

- References to public documents:

[1]. Leopoldo J. Gutierrez-Gutierrez and Vanesa Barrales-Molina, Department of Business Administration, University of Granada, Granada, Spain, and Hale Kaynak. Department of Management, The University of Texas Rio Grande Valley, Edinburg, Texas, USA. (2017). *The role of human resource-related quality management practices in new product development - A dynamic capability perspective.*

[2]. Cristina Evanghelia PAPADIMITRIU. (2022). Rivista di Diritto del Risparmio, *Le TechFin Companies nel mercato finanziario.*

[3]. Certificazione di Tester Syllabus 'Foundation Level', International Software Testing Qualifications Board, Versione 2018 V3.1.

[4]. Montgomery. (2009). *Introduction to Statistical Quality Control 6th edition.*

[5]. Jérôme Barthelemy, Dominique Geyer. (2001). IT outsourcing: Evidence from France and Germany

[6]. Bertrand Que´lin, François Duhamel, HEC School of Management, Jouy-en-Josas. (2003). *Bringing together Strategic Outsourcing and Corporate Strategy: Outsourcing Motives and Risks.*

[7]. Moderari s.r.l. (2015). *Le nuove definizioni di credito deteriorato*.

[8]. Domenico Augusto Maisano. Course of Quality Engineering, *Performance Indicators.*

[9]. L. Finkelstein, M.S. Leaning. (1984). *A review of the fundamental concepts of measurement.*

[10]. Florenzo Franceschini, Maurizio Galetto, Domenico Maisano. (2018). *Designing Performance Measurement Systems - Theory and Practice of Key Performance Indicators*.

[11]. Krajewski, Pearson Global Edition. *Acceptance Sampling Plans,* Supplement G.

[12]. Domenico Augusto Maisano. Course of Quality Engineering, *Theory of sampling inspection: Acceptance Sampling*

[13]. Sajid Ali, Antonio Pievatolo, Rainer Göb. (2016). *An Overview of Control Charts for High-Quality Process,* Review article.

[14]. Western Electric Co., Inc. (1958). *Statistical Quality Control Handbook*.

[15]. Domenico Augusto Maisano. Course of Quality Engineering, *Control Charts.*

[16]. K. S. Chen, M. L. Huang, R.K. Li. (2001). *Process capability analysis for an entire product.*

- Website References:

[A]. «L'evoluzione tecnologica che sta trasformando tutti i settori di mercato porta il software testing in primo piano», https://www.quence.it/

[B]. «Alten Official Website», www.alten.it.

[C]. «Il recupero crediti: cosa sapere», https://www.creditpmi.it/

[D]. «Software Testing Metrics and Indicators», https://www.thinksys.com/qa-testing/software-testing-metrics-kpis/

- Internal company documents:

[a]. PGR – Breve guida introduttiva, 2019.

[b]. AFU_ PRJ466597-02_Upload e azioni massive centralizzate_v7, 2022.

[c]. LINEE GUIDA SCART, 2021.

[d]. Upload project Testbook - Template SCART_PGR - Upload_V2, 2022.