D6.6 Software Quality Assurance measures for AGRICORE

agricore

Deliverable Number	D6.6
Lead Beneficiary	AAT
Authors	AAT, IDE, ALL
Work package	WP6
Delivery Date	M15
Dissemination Level	Public

www.agricore-project.eu

The Agricore project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No. 816078

Document Information

Project title	Agent-based support tool for the development of agriculture policies
Project acronym	AGRICORE
Project call	H2020-RUR-04-2018-2019
Grant number	816078
Project duration	1.09.2019-31.8.2023 (48 months)

Version History

Version	Description	Organisation	Date
0.1	ToC	AAT	04 Sep 2020
0.2	Development of sections	AAT	30 Oct 2020
0.3	Feedback from all partners (IDE, ALL)	IDE, ALL	09 Nov 2020
0.4	Feedback from all partners resolved	AAT	13 Nov 2020
1.0	Final version	AAT	25 Nov 2020

Executive Summary

The document D6.6 "Software Quality Assurance measures for AGRICORE" aims at guiding the development that are involved in the AGRICORE architecture, which includes several modules developed by the different partners of the project. Since all modules interact with each other to build up the AGRICORE ecosystem, the development guide provides a solution to ensure that the integration of all developments is done most straightforwardly, avoiding last-minute integration problems that may affect the project schedule, as well as unify the mechanisms and solutions offered.

To do this, software quality assurance processes will be defined and established in line with the existing standards. In particular, the next list of measures has been defined:

- Development workflow: Guideline that defines how the features are developed and integrated incrementally using git as a configuration management tool.
- Testing guidelines: Definition of how the tests should be implemented to assure a high software quality level, grouped by level of details and interactions with external modules. These guidelines were applied in the definition of tests included in this deliverable.
- Metrics: Software metrics measurements, indicating which tools are going to be used to perform the measurement operations.
- Continuous Integration (CI): Description of how Continuous Integration is applied in the AGRICORE project, allowing to integrate all the previous points in a single workflow.

Abbreviations

Abbreviation	Full name
ABM	Agent-Based Model
AG	AGRICORE
API	Application Programming Interface
ARDIT	Agricultural Research Data Index Tool
CAP	Common Agricultural Policy
CD	Continuous Delivery
CI	Continuous Integration
CSV	Comma-Separated Values
DX	Deliverable X
DWH	Data Warehouse
ETL	Extract, Transform and Load
FR	Functional Requirement
FT	Functional Test
HTML	HyperText Markup Language
ICT	Information and Communications Technology
ID	Identifier
ISQTB	International Software Testing Qualifications Board
IT	Integration Test
JSON	JavaScript Object Notation
KPI	Key Performance Indicators
M15	15th Month
REST	Representational State Transfer
XML	Extensible Markup Language

List of Figures

List of Tables

Table of Contents

1 Introduction

The main purpose of the presented document D6.6 "Software Quality Assurance measures for AGRICORE" is to guide the development of all the individual developments that are involved in the AGRICORE architecture, which includes several modules developed by the different partners of the project. Since all modules interact with each other concluding in the AGRICORE ecosystem, the development guide provides a solution to ensure that the integration of all developments is done most straightforwardly. This will enable avoiding last-minute integration problems that may affect the project schedule, as well as unify the mechanisms and solutions offered.

To do this, software quality assurance processes will be defined and established in line with the existing standards[1]. In particular, the next list of measures has been defined:

- Development workflow: Guideline that defines how the features are developed and integrated incrementally using git as a configuration management tool.
- Testing guidelines: Definition of how the tests should be implemented to assure high software quality levels, grouped by level of details and interactions with external modules.
- Metrics: Software metrics measurements, indicating which tools are going to be used to perform the measurement operations.
- Continuous Integration (CI): Description of how Continuous Integration is applied in the AGRICORE project, allowing to integrate all the previous points in a single workflow.

The final section of this document provides the first version of all the functional and integration tests that will be implemented and executed to ensure the software quality levels expected.

The current document will be used as a basis for any development along the whole lifespan of the project and will be updated according to the detected needs of further tests or procedures not detected in the project at this M15 of the project.

1.1 Document Conventions

- The datasets are independent, so there are no interdependencies among them, and joint operations are not going to be necessary during the ETL process.

The presented document, D6.6 "Software Quality Assurance measures for AGRICORE", has been generated in M15 of the AGRICORE project. At this stage of the project, only a subset of the total number of tests has been defined and included in this document due to not all the AGRICORE project modules has been analysed yet with the granularity level required to define the different test levels purposed. This document is an initial version although it will be continuously updated to include more detailed tests to be done, depending on the developments done for each module.

1.2 Intended Audience

This document is primarily intended for all the partners that are involved in the consortium to have a guideline of how to perform the development tasks to assurance the software quality levels expected for AGRICORE.

The European Commission is also in the scope of the intended audience to report on the progress of the AGRICORE project and meet the project's milestones.

New developers, testers and other stakeholders interested in this project can consult this document to learn about how AGRICORE assurance the quality levels expected, what tests have been defined and their coverage in terms of defined functional requirements.

2 AGRICORE Overall Description

The next section provides a summary about the AGRICORE project and the modules that are composed by. To clarify the technical scope, assumptions and constraints are defined in the next subsections.

2.1 AGRICORE Project Summary

At the current stage of the project, first development tasks have been started to iterate and analyse the bests solutions in terms of software quality assurance and software development workflow. Because the integration of all AGRICORE modules is one of the most critical points that can seriously affect development progress, several points about this should be defined and well known for all the partners involved in the AGRICORE tool development. These modules are defined below:

- D1 ARDIT (Agricultural Research Data Index Tool, formerly referred as European data sources index module). The tool allows users to search for different sources of publicly available data on the Web, categorised by the methodology implemented according the ontology AGRICORE DCAT-AP 2.0 extension.
- D2 DWH: Data Warehouse tool suitable for supporting the analyses contemplated within the AGRICORE project.
- D3 Data extraction Module: Module that extract all the data of interest from multiple datasets considered in the project. Data extraction encompasses the capabilities for accessing different datasets, selection the necessary data and formatting it for further processing.
- D4 Data fusion module: Combine the individualised data with the probability distributions of the variables to generate the joint probability distributions.
- D5 Synthetic populations generator: Module aimed to obtain realistic synthetic population making use of the Synthetic Reconstruction method.
- D6 ABM simulation engine: Instantiate agents for each farmer generated, evaluating its situation and making decisions based on its preferences.
- D7 External interface module: Gateway for the interoperability between the modules to the ABM simulation engine.
- D8 Model interaction modules: Modules that interact with the model generation modules and the external interface module.
- D9 Biophysical model connection module: Provide a biophysical model to the AGRICORE tool.
- D10 Impact assessment module: Provides different modules used to evaluate the KPI's (Key Performance Indicators) related to specific topics (e.g. Environmental / Climate KPI's).
- D11 Policy environment module: Define different policies and translate them into an input for the simulation engine.
- D12 Agricore interface module: Centralise all the interactions of the user with the AGRICORE tool, allowing to see the results of the simulations defined and performed along the simulation process.

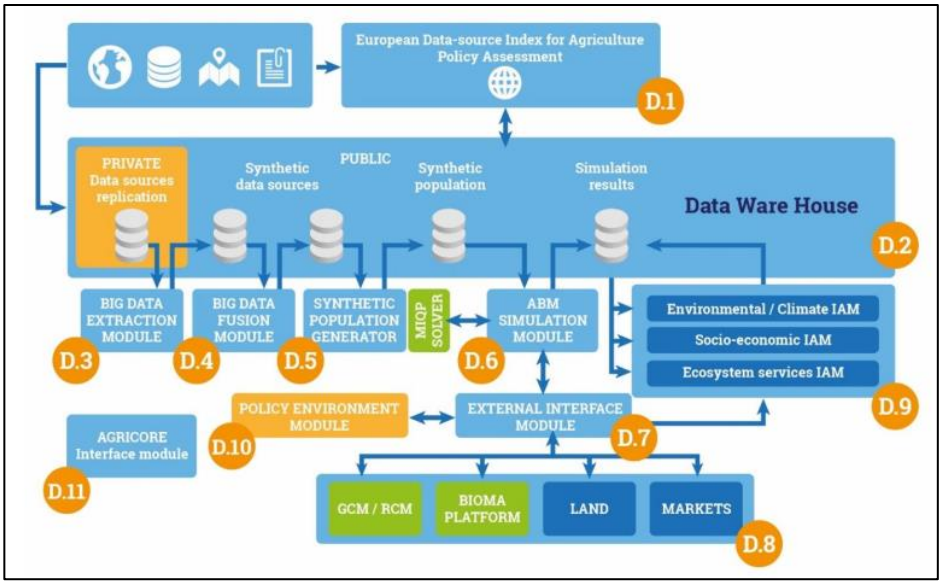

Figure 1 Current version of AGRICORE architecture

2.2 Module communication traceability matrix

As described previously, one of the most critical points in terms of software quality assurance is to get knowledge about how the modules interact with each other. Defining these communications, a list of integration tests could be defined and implemented to ensure the correct interoperability between modules.

To do so, a communication matrix has been defined to register the direct communication between modules. Direct communication has been defined when a module is going to be communicated with another module in terms of physical connection. For example, if a module needs to store information into the DWH, the module will need direct communication with the DWH module. Thanks to this definition, module developers will be able to identify the external services with which they have to interact.

The modules communication traceability matrix is provided below, using the green colour as direct communication:

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
D1: ARDIT	N/A	YES	-	-	-	-	-	-	-	-	-	-
D2: DWH	YES	N/A	YES	YES	YES	YES	YES	-	-	-	-	YES
D3: Data extraction Module	-	YES	N/A	-	-	-	-	-	-	-	-	-
D4: Data fusion module	-	YES	-	N/A	-	-	-	-	-	-	-	-
D5: Synthetic populations generator	-	YES	-	-	N/A	-	-	-	-	-	-	-
D6: ABM simulation engine	-	YES	-	-	-	N/A	YES	-	-	-	-	YES
D7: External interface module	-	YES	-	-	-	YES	N/A	YES	YES	YES	YES	YES
D8: Model interaction modules	-	-	-	-	-	-	YES	N/A	-	-	-	-
D9: Biophysical model connection module	-	-	-	-	-	-	YES	-	N/A	-	-	-
D10: Impact assessment module	-	-	-	-	-	-	YES	-	-	N/A	-	-
D11: Policy environment module	-	-	-	-	-	-	YES	-	-	-	N/A	-
D12: Agricore interface module	-	YES	-	-	-	YES	YES	-	-	-	-	N/A

Table 1 Module communication traceability matrix

Design and Implementation Constraints

The design and implementation processes used to implement and achieve the presented methodology and technologies rely on a set of different defined constraints:

- The AGRICORE tool is open source. This policy ensures that the technologies, tools and third-party platforms used for the design and implementation of software quality assurance measures must be public, available and accessible for all researches, institutions and developers who want to use and improve the tools implemented during the AGRICORE project.
- The continuous integration processes implemented has been integrated using GitLab in its Gold version. The use of this tool will allow to increase the integration of all the software parts that are involved in the AGRICORE tool.
- The GitlabCI platform in its Gold version provides 50,000 minutes per month to execute continuous integration tasks at this stage of the project. At the time when the minute rates decreases the continuous integration processes should also decrease[2].
- Performance tests have to be executed on a local machine due to the high demand of resources needed by the machine to launch them. These tests will be executed statically when AGRICORE tools have been developed.

2.3 Assumptions and dependencies

To comply with the software quality expected for the AGRICORE platform, the following sections define several assumptions and dependencies to frame the work coverage area, as well as the dependencies required for the development of the project.

2.3.1 Assumptions

- All the tests defined in AGRICORE must be written in English.
- The datasets requested by the use cases to generate the different models needed must be available during the life cycle of the project to be consulted and used for testing purposes.
- The datasets are independent, so there are no interdependencies among them, and joint operations are not going to be necessary during the ETL process.
- All datasets used along the project are anonymised due to the execution of the automatic tests in external platforms such as GitlabCI or metric platforms.

2.3.2 Dependencies

- The datasets requested by the use cases to generate the different models needed must be available during the life cycle of the project.
- The platforms used for the continuous monitoring of quality measures are web-based solutions. These platforms must be publicly available to provide the expected levels of quality in terms of metrics measurements and continuous integration processes integrated in the project workflow.

3 Development workflow

To ensure that all the developments tasks are executed using a single and well-defined methodology allowing to all the partners and developers interact in all the developments of the AGRICORE project, the development process has been defined using the most common git workflows defined and implemented in most of the software development projects. These workflows frame and minimize the most common errors that occur during software development, specifically in projects where several developers are working together. The following workflow is based on the best practices provided by the official book of Git[3].

3.1 Git

As it has been mentioned previously, this workflow is associated with the use of the version control tool called Git. Git is a tool that provides to manage and track the maintainability of the software versions. The version control system records all the changes associated with a file or set of files over time, so it can recall specific versions later. Software projects are allocated in *git repositories*, the places where all the files and data, including changes, are stored. Remote git repositories are published in a distributed server or a web-based service. In the AGRICORE project context, the remote repository is hosted by GitLab.

When developers need to download a repository to work with, they have to perform the operation *clone,* downloading a copy of the remote repository on their local machine. Then, all the changes done in a file or multiple files are packaged in a single incremental that is called *commit*. When developers are working in a new modification or a new feature, modifying, creating or removing files from a project, they could pack all the changes in one or multiple commits. These commits are stored locally (on their own machine) and it could be published by the developers using the operation called *push,* sending their commits to the remote git repository. Later, other co-workers would need to retrieve the last changes (commits) from the distributed git repository, they could perform the operation *pull* to retrieve all the last changes published in the remote git repository and update their local version of the project.

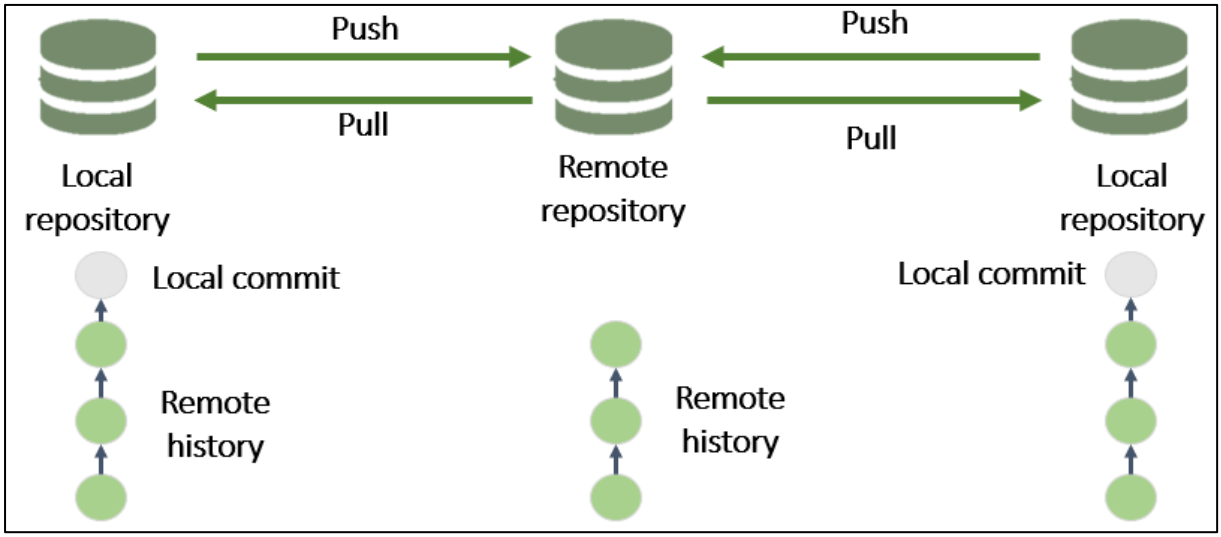

Figure 2 Git repository - Remote and local repository

Due to several developers could modify the same file at the same time, a conflict may appear when a user needs to integrate its changes. Conflicts must be resolved as soon as possible before executing any git operation.

Another main concept of git are the *branches. Branches* are used to separate the different purposes of a list of commits. Git repositories always have a default branch called *master* where the commits are stored as a graph. Also, we can perform many operations on branches, for example, we could create new branches from others, safeguarding the state of the latter, we could merge a branch into another, or we could delete branches too.

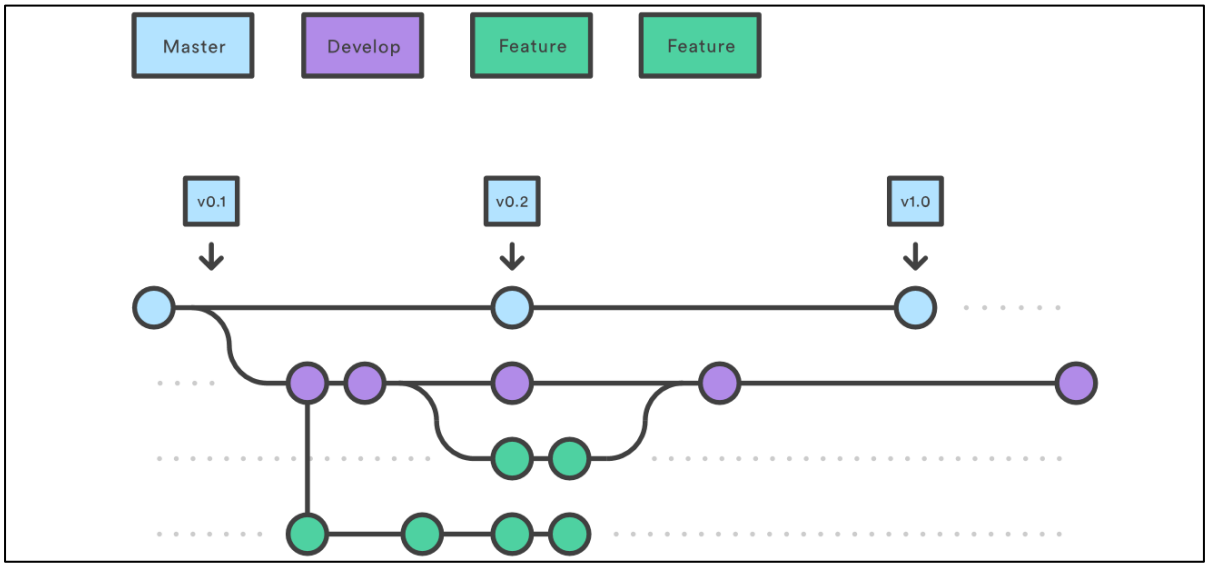

Figure 3 Example git workflow

Although git provides several operations, processes and mechanisms to manage and reach all the main objectives that a version control system must achieve, GitLab and other git hosting services offer another set of features constructed over git to complement its functionality with new capabilities. One of its main features that is very useful, and it will be used during the AGRICORE project is the *merge request*.

A merge request is a request to integrate one branch into another. This process is a petition used to visualize and collaborate on proposed changes to source code. It displays a great set of information about the changes proposed, as well as a description about the process, discussion threads and more information about external services that could be integrated into the git-flow process such as CI / CD pipelines[4]. The merge request adds an extra protection layer to trace and verify that a new incremental of the source code is going to be added in another branch. This brings the possibility to add reviewers in the project to check that the source code follows the guidelines defined in terms of quality, goals and code style among others. In the case of AGRICORE project, when a merge request is created, the CI processes will be executed to verify that all the tests have been done successfully, as well as to execute the code metrics processes.

Operating with multiple branches leads to a Git specific workflow, known as GitFlow. GitFlow defines a strict branching model developed by Vincent Driessen[5] that helps other developers to take more control and organisation in software development. GitFlow gives guidelines on how functions should be assigned to branches and how they should interact with each other. Despite this introduction, the purpose of this document is not to how git works in detail. For that, resources such as Pro Git book, written by Scott Chacon and Ben Straub can be consulted[6]. The following subsection will describe the specific GitFlow applied to AGRICORE project.

3.2 Workflow

The next subsection will cover how the git repository must be initialized and all the steps involved in the workflow.

3.2.1 Initialization

The git repository must be initialized with the next two branches:

- *master:* This branch contains the latest stable version of the AGRICORE module. This branch will be updated with new incremental on each milestone of the module implementation.
- *develop:* This branch birth from the master branch and allocates the develop version of the module. When a milestone is reached, the develop branch will be merged into master.

3.2.2 New feature

Each development should be associated to a technical task of a requirement. In GitLab, each requirement is registered as an epic, so new technical tasks should be associated to a specific requirement using the *issues.* An issue is a task that must be associated to a specific project inside GitLab. The issue has several fields to be filled such as description and the weight of the task. An issue has associated a unique identifier that can be referenced during the development of the task. It is a good practise to create small and independent merge requests and tasks to minimize and determine all the incremental developed during the AGRICORE project.

When a technical task is created, a new branch must be created from *develop* using the format described below:

- *feat-<issue_id>-<brief_description>*
    - E.g. feat-1-ldap-service-integration.
    - *issue_id:* Unique id of the technical task.
    - *brief_description:* Description about the purpose of the branch created.

In case that a technical task must be decomposed in different small developments, this process could be performed over a feature branch instead of *develop*. This procedure makes it possible to trace to which incremental the technical task belongs and minimize the time spent reviewing merge requests processes. Technical tasks creation and how to link a branch to a specific task is a tedious process, but it can improve the management and traceability of all the developments performed in a git repository with GitLab.

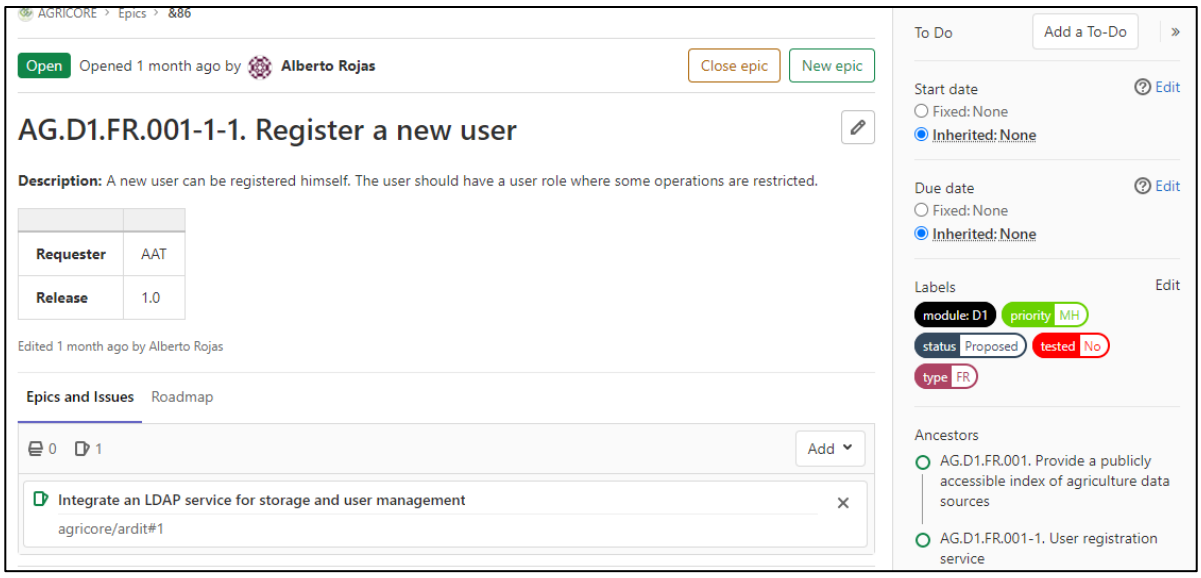

Figure 4 Requirement registered in GitLab

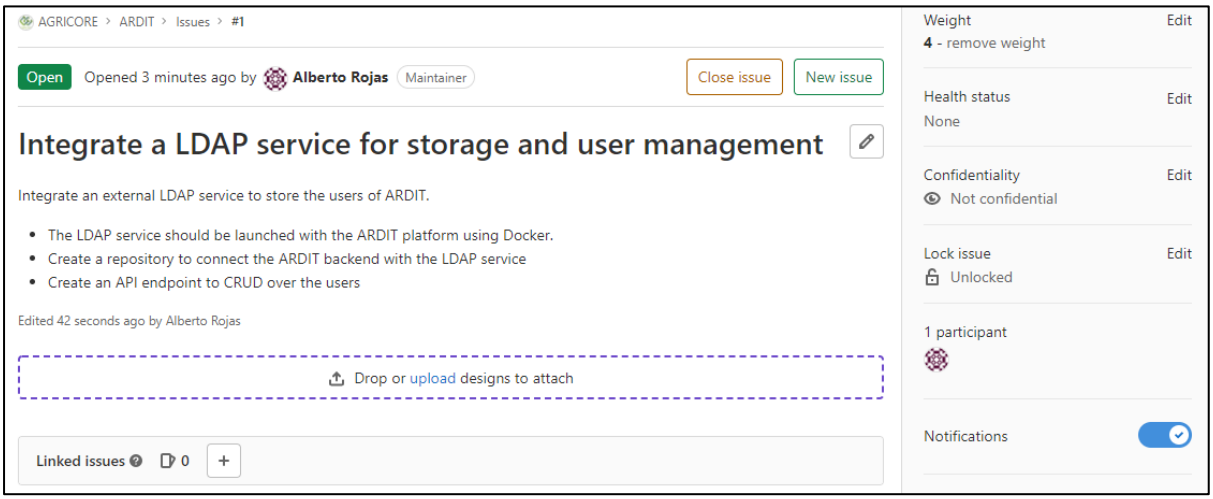

Figure 5 Technical task in GitLab

When a branch is created, developers can work on their own branches and make there all the incremental necessary to finalize the technical task. The guideline recommends creating commits of atomic incremental instead of a big one with several changes that may hinder the review process. The commit messages should have a descriptive title about what is the purpose of the incremental, as well as the reference of the technical task, but it is not mandatory, because the merge request will squash all the commits into a single one when it is accepted.

3.2.3 Finishing a feature

This process starts when a developer finalises the development of a technical task. When developers are ready to add all their changes to the parent branch (*develop or another feature branch)*, they must create a merge request in GitLab. A merge request must have a descriptive title, on which is recommended to add the reference to the technical tasks typing or using its ID after the prefix character *'#'*. The next figure shows an example of how a merge request should look with a reference to the issue with ID 10:

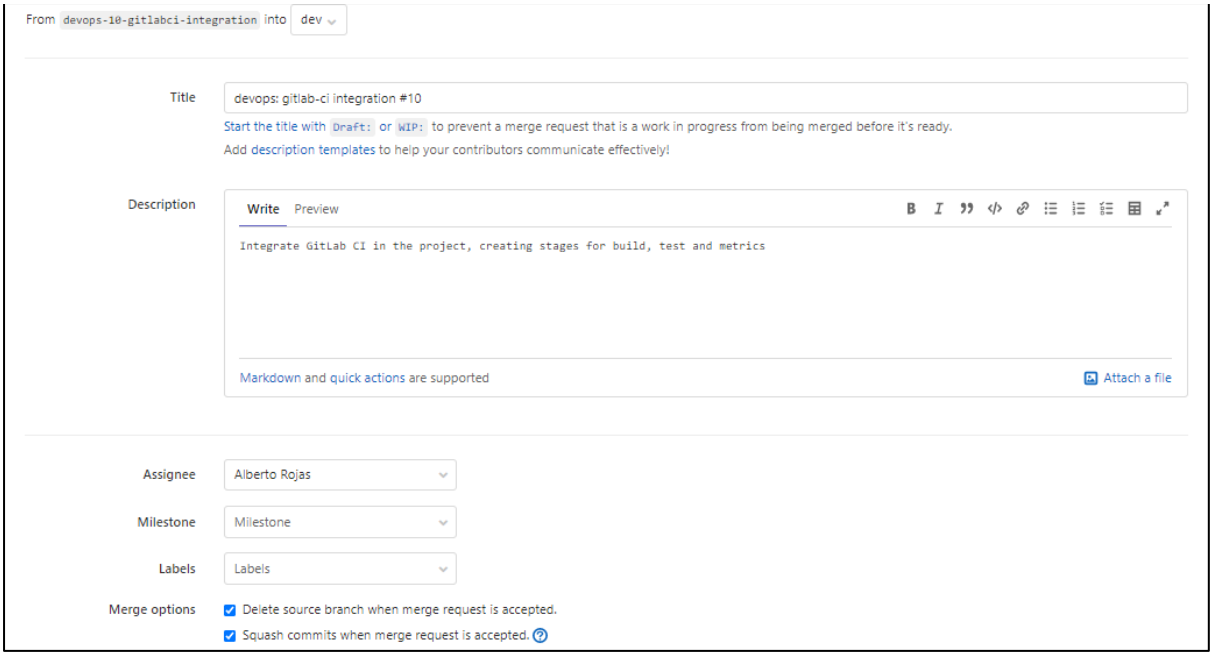

Figure 6 Merge request creation

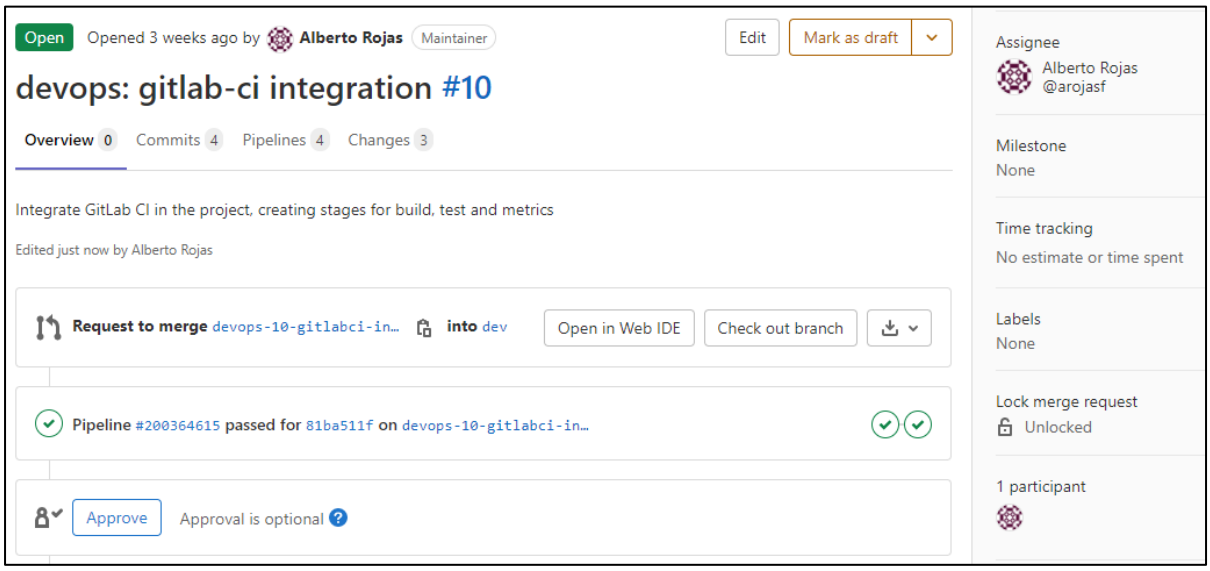

Figure 7 Merge request result

In this process, some developers will review the code and highlight errors or improvements on it. Then, the developers that made the request, should verify their work and send some feedback with the solutions implemented. This will trigger the CI processes where all the tests defined and implemented will be launched automatically to check that the solution does not have any error. When the merge request has been verified and all the conflicts had been resolved, developers that review the code can merge the development into the targeted branch.

During the merge request process, GitLab offers the possibility to delete the source branch once it is merged with the targeted branch. This is a very useful operation in terms of cleaning the GitFlow, deleting branches that were used to develop some new features or correct some bugs, but they aren't going to be useful again, maintaining only the main branches to keep working. Another option that GitLab offers during merge is *squashing*, this operation combines all the commits done on source branch into one to keep a clean history of commits on the targeted branch. For example, having a source branch with 8 commits and enabling squash, a simple

commit will be generated on the targeted branch and will represent the previous eight. That commit's message content would be set as a representation of the merge operation itself, for example, *Merge 'branch-1-name' into 'branch-2-name'*. In the end, squashing prevents the number of commits from growing exponentially with each merge request on targeted branches.

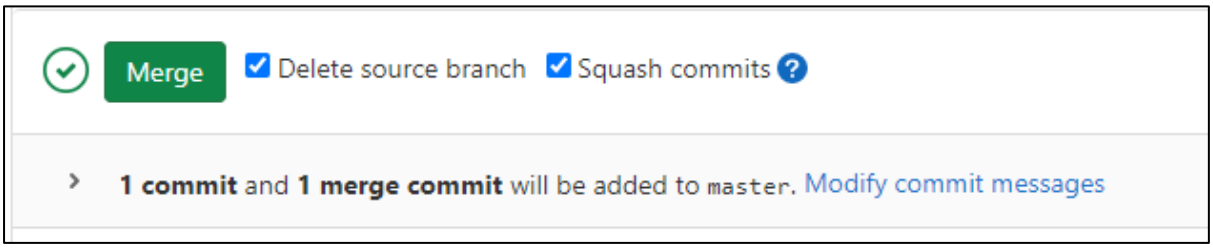

Figure 8 Merge request configuration in Gitlab

4 Quality Assurance measures

Software quality assurance is defined as "the degree to which a software product meets established requirements; however, quality depends upon the degree to which those established requirements, accurately represent stakeholder needs, wants, and expectations"[7]. The software quality assurance process monitors all the software development processes, tools and deliverables that are involved during the life cycle of the project.

This assurance could be applied including standards and procedures that several roles, such as developers, can use to review and audit the software deliverables and activities to verify that the expected output meets the quality measures defined. Because software project is composed of multiple processes, monitoring all of them can be a very costly process concerning resources to be used. A balance between quality assurance and agility during the project must be made, trying to maximize the effectiveness of the project and automate as many processes as possible.

4.1 Metrics

Software metrics are some kind of measures of a software product or project, which determine or enhance the quality of it. The measurement of systems and software product quality is defined by the ISO/IEC 25023:2016.

Software metrics can be quantitative or qualitative, depending on whether metrics can be expressed in values or applied in the software development to improve its characteristics. Also, metrics must be simple, consistent, understandable, reliable and should not depend on any programming language.

This section is a simple introduction to the use of metrics to measure the quality of software and to define the metrics used in AGRICORE project. The description of the steps to obtain them or the description of the tools used can be found in the Continuous Integration chapter of this document.

4.1.1 Code quality

Code quality is not a simple software metric, but a set of them that are sometimes related. It includes both quantitative metrics like number of lines per function or code complexity, and qualitative metrics like readability, code clarity or maintainability.

4.1.1.1 Complexity

Within the quantitative metrics, the main reference is code complexity which, in turn, is conformed by other sub metrics like **cognitive** and **cyclomatic complexity**. The first one defines how difficult the code is to read or understand based on a set of rules, for example, using shorthand and collapsing multiple statements into one is considered a good practice to reduce complexity, but long nested structures like conditionals or loops increase it. On the other side, cyclomatic complexity measures the number of executions paths through code, or in other words, the number of decisions that a block of code needs to make. Complexity is measured by a numerical value in a range. The greater the number, the greater complexity will be and vice versa.

4.1.1.2 Code clarity

Code clarity is an indicator of quality that measures whether a piece of code it is ambiguous or not. This, along with readability indicates again how easy is the code to be understood. Clarity and readability are related to complexity, if this one has a high value, the first ones will decrease and vice versa. Another point within code clarity are duplications, parts of the code that are duplicated.

4.1.1.3 Maintainability

Code maintainability is a qualitative measure that defines how easy it will be to make changes to the code in the future. Maintainability depends on other quality metrics, which includes duplication, similar blocks of code, complexity and structural issues like file or method lengths.

4.1.2 Coverage

Code coverage is a measure used in testing to determine which parts of the code have been covered by unit tests, and which parts have not. Coverage is usually measured through percentages that indicate the total number of lines covered, but quality assurance tools can also generate detailed reports that can even indicate, in the source code itself, which lines are covered or not by unit tests. Coverage is very important to ensure that most of the code is tested, the closer it is to 100% value the better, but it can also give a false sense of security and confidence because coverage alone does not guarantee that the software will work as expected. This is where other quality and code cleanliness metrics come into play.

4.1.3 Other metrics

There are other quality metrics such as performance, correctness or integrity, but these can be measured through tests in the code, as it will be explained in the following section.

4.2 Testing guidelines

The following section purpose a defined guideline of how the testing operations must be performed during the lifecycle of the AGRICORE project. Software testing is one of the software development core activities that can be executed during the development processes and/or at the end of the development cycle of a project. In AGRICORE project, one of the main requirements provided by the Grant Agreement is to avoid any last-minute integration problems that may affect the scheduling of the project. To achieve this goal, a well-defined guideline about how to proceed in the software testing process will decrease the risk enumerated previously.

As It was defined in the previous deliverable D4.1 "AGRICORE requirements and project management platform", Software testing is a process that has different targets, and at this stage of the project, is important to clearly define our focus on the main core activity of ensuring product quality levels by using different methodologies, frameworks and practices. The main goal of the software testing process during the AGRICORE project is to ensure that the AGRICORE tool offers an excellent level of robustness and that all the requirements defined by the Grant Agreement and the stakeholders are satisfied and traced by a set of tests.

These tests, excepting the unitary tests, must be defined previously in a list enumerated to get track about the current number of tests available for the AGRICORE tool. They will be added in the present document due to this process will be in process during the life cycle of the project, in parallel with the development process. Because the project is not at a high level of maturity in terms of development, this process has been defined to be performed manually but, when this level of maturity increases, the process could be designed to be executed automatically making an integration of all the modules involved in the AGRICORE tool in a single centralized repository, using the features provided by GitlabCI to perform this operation.

In the following list provides a template about how the tests must be defined:

Code	**Unique test ID. The code must follow the next convention: TEST.<main module>.<type>.<incremental number>-<child number>. <descriptive title>** • **E.g. TEST.D1.FT.001-1. Example test.** ○ **main module: Module associated to the tests. Although a test could be associated to two modules, the association to a module is mandatory.** ○ **type: Test type.** ▪ **IT: Integration test.** ▪ **FT: Functional test.** ▪ **PT: Performance test.** ○ **incremental number: Unique ID using 3 digits.** ○ **child number: Test annidation.** ○ **descriptive title: The titles of the texts must be descriptive to easily know what the test covers.**
Requirements	List of requirements IDs that are covered by the test.
Modules	List of modules IDs that are involved in the test. At least, two modules must be covered by the test.
Description	Detailed description of the test, its purpose and/or the definition of the format and/or communication protocol and expected outcome.
Result	Indicates whether the tests have been passed or not. The values could be *Passed* and *Not passed.* If the tests wasn´t carried out yet, the value *Not tested yet* will be shown.

As well as the requirements, the tests could be appended and grouped into another one. This convention has been defined to increase the evolution of the test suite. Due to the incremental number must be different for each test, it decreases the flexibility of the methodology. The child number of the tests provide a way to create different tests for a specific feature.

To illustrate this problem, an example is provided. If the feature 'user login' has to be tested, it could be necessary to provide different tests for this feature using only a single unique code:

- TEST.D1.FT.001. User login successfully in the platform
- TEST.D1.FT.002. Admin login in the platform
- TEST.D1.FT.003. Maintainer login in the platform
- TEST.D1.FT.004. <Test of another feature not related with the login use case>

Using a test annidation mechanism:

- TEST.D1.FT.001. User login
    - TEST.D1.FT.001-1. User login successfully in the platform
    - TEST.D1.FT.001-2. Admin login in the platform
    - TEST.D1.FT.001-3. Maintainer login in the platform
- TEST.D1.FR.002. <Test of another feature not related with the login use case>

If during the project life cycle the creation of new scenarios for the same feature is requested, the incremental ID will not be aligned with the previous scenarios defined due to other tests could had been registered in the project. To provide a solution to this problem, a mechanism of test annidation is provided that allows that the test suite could be modified and increased without losing readability and traceability.

4.2.1 Unit testing

Unit tests are used to analyse small portions of code and components of a software project to validate that each one performs as designed or expected. Unit testing is the first and most basic level of software testing, developers use it to increase confidence in maintaining code, running tests every time that any code line is changed to detect possible errors or bugs introduced due to changes. This decreases the cost of fixing an error as developers only need to scan small and well-defined portions of code.

Unit tests are composed of three phases:

1. **Initialization phase:** initialization of useful data to be used by the test.
2. **Supply phase:** tests supply data to the tested component, usually calling a method.
3. **Observation phase**: where results are analysed. If the resulting behaviour is the expected one, the unit test passes, otherwise, it fails.

The next figure shows an example of a unit test structure where *ExampleClass* class has a method, *multiply*, which receives two numbers and applies the multiplication of both. In the observation section, *multiply* method is called receiving the values with which it will carry out the operation. Then, *assertEquals* method will check if the returned value is equal to *zero*. In this case, the test will fail because the first two behaviours will run as expected, but the last one will not be fulfilled because 10 multiplied by 1 is not 0.

```
[Test Method]
public void testExample() {

    ExampleClass test = new ExampleClass(); // Initialization

    // Observation
    assertEquals(0, test.multiply(10, 0)); // Supply
    assertEquals(0, test.multiply(0, 10));
    assertEquals(0, test.multiply(10, 1));
}
```

Figure 9 Unit test example

Besides, AGRICORE's unit tests must comply with the following guidelines:

- Unit tests must be simple functions easy to maintain.
- Stateless and independent of each other.
- Must avoid conditionals statements on them.
- Each unit test must test one thing.
- Negative tests must be also tested to verify robustness on error handling.
- Tests must be named accordingly, using, for example, *test<purpose>* like *testUpdateExistingUser()* or *testCreateUser()*.
- Unit tests must keep the system in the same state it was before their execution, any modification or change made to the databases must be undone.

4.2.2 Integration tests

According to ISQTB (International Software Testing Qualifications Board), Integration test is "a testing performed to expose defects in the interfaces and the interactions between integrated components or systems"[8]. Because these tests are executed for a group, several levels of testing could be defined for them. Regarding its implementation in AGRICORE, the integration tests must be defined to check all the communication between modules, making use of the traceability matrix among modules presented in the section "AGRICORE Project summary", as well as satisfy the requirements of the AGRICORE tool.

About the traceability matrix, at least one integration test per module must be defined for each pair of modules that composed the communication. Of course, all the requirements must be satisfied for a set of tests, so any requirement that is not a use case, it has to be tested using an integration test (e.g. AG.D3.FR.004. Data output stored in DHW).

As it was defined previously, this process has been defined to be performed manually but, when this level of maturity increases, the process could be designed to be executed automatically making an integration of all the modules involved in the AGRICORE tool in a single centralized repository, using the features provided by GitlabCI to perform this operation.

In the following list, it has been provided an example about how the integration tests must be defined:

Code	**TEST.D1.IT.001. ARDIT send and launch an ETL to the DWH**
Requirements	• AG.D1.FR.007-2. ETL execution in the DWH • AG.D1.FR.007-2-4. Launch an ETL in the job queue launched
Modules	• D1: ARDIT • D2: DWH
Description	The module ARDIT must send an ETL script into the DWH and launch it into the DWH. The ETL is sent using a REST API with the following format: ...
Result	Not passed

4.2.3 Functional tests

The functional tests is a quality assurance process where a testing operation is performed to evaluate if a component or system satisfies the functional requirements[9]. It is based on the specification software component and there are many types of functional test depending on the level and the type of the test. In this case, the functional tests performed in the AGRICORE project to verify that the requirements have been satisfied are going to be the acceptance tests.

The purpose of the acceptance tests is to validate that a system achieves the expected performance and allows to the user that the system satisfies their needs. This process must be performed by the user and verify that the use cases aligned with the requirements has been covered. This acceptance tests could be executed automatically, or they could be checked manually but, as it was mention in the previous section, the integration of all the modules of AGRICORE and the automatic execution of the tests that are involved in several modules has not been designed and implemented yet.

Regarding the functional test guideline, all the functional tests must be defined using the Gherkin syntax. Gherkin is a domain-specific language which helps to describe business behaviour without the need to go into detail of implementation. This notation is compatible with several programming languages and they can be integrated with several testing tools. Gherkin uses a set of special keywords to give structure and meaning to the scenario. Information about the Gherkin syntax could be consulted in the official reference guide[10].

```
Example: Multiple Givens
  Given one thing
  And another thing
  And yet another thing
  When I open my eyes
  Then I should see something
  But I shouldn't see something else
```

Figure 10 Gherkin example

A Gherkin scenario is composed by two sections defined below:

- **Example/Scenario:** Concrete example that illustrates a business role. The keyword Example is a synonym of Scenario.
- **Steps:** Defines a step in the scenario in the sequence written.

Although Gherkin support several keywords and options, in the present guideline will use the following keywords:

- **Example/Scenario:** Definition of the scenario described.
- **Given:** Describe the initial context of the system. The keyword **And** can be used to concatenate more initial contexts.
- **When:** Describe an event or an action.
- **Then:** Describe an expected outcome or result. The keyword **And** can be used to concatenate more expected results.

An example of functional tests defined using the guideline defined is provided below:

Code	TEST.D1.FT.001. ARDIT user login
Requirements	• AG.D1.FR.001-2-1. Login
Modules	• D1: ARDIT

Description	• Scenario: Existing user login in ARDIT
	• Given a user visits the login form
	• When the user enters its username in the "username" field
	  ○ And the user enters its password in the "password" field
	  ○ And the user presses the "login" button
	• Then the user sees the home page

Result	Passed

4.2.4 Performance tests

According to ISQTB, performance testing is the process that determines the performance efficiency of a component or a system[11]. Typical parameters measured in this process include processing speed, data transfer rate, network bandwidth and throughput, workload efficiency and reliability[12]. In the context of AGRICORE, the performance tests executed will be related to the number of agents that can be launched during the simulation process, as well as to the time

that elapses from start to finish from when a user runs a simulation until the result is displayed to the user.

This process could be executed using specific tools for this purpose such as Apache JMeter[13] that can be used to simulate a heavy load on a server to test its strength or to analyze overall performance. In the current state of the project, no such performance tests have been designed as these developments have not yet been addressed, nor have performance tests been defined that would provide added value to AGRICORE.

5 Continuous Integration

Continuous integration is a software engineering mechanism used to integrate all the changes of a project automatically to prevent or detect errors as soon as possible. CI is extensively used in merge requests when developers share a remote repository, running automatic builds and tests every time a developer integrates new code in it. This allows to monitor project quality continuously, detecting possible errors earlier and validating all the code before merging.

Previously, developers did their part of the work separately and in the end, they joined all the developments which led to multiple errors that could not be detected earlier, making the integration process more difficult and increasing delivery times. With CI, developers frequently commit changes to the repository, running tests each time as a verification measure before integration. CI allows to improve developer productivity, generating more collaboration between co-workers in the development cycle, to find and fix bugs earlier, thanks to the tests carried out automatically, to reduce delivery times and, in conclusion, to improve the project quality.

GitLab continuous integration is activated by a configuration file called *gitlab-ci.yml* placed at the repository's root path. When a merge request is opened, GitLab CI file creates a ***pipeline***, a process that includes a set of ***jobs***, for example, a job for code compilation and another one for testing, and ***stages***, which define when to run the jobs. For example, the stage for code compilation and all its jobs must run before the stage for testing. In AGRICORE project, two stages have been defined currently in GitLab CI file. The first one, *build stage*, is used to build the application and compile it. The second, *test stage*, groups the jobs that run unit tests and measure software quality metrics.

More information about how to integrate CI in the workflow can be found in the official GitLab documentation[14].

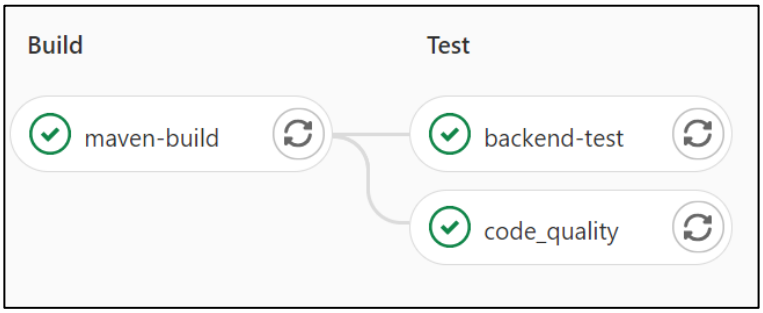

Figure 11 Agricore pipeline stages and jobs

5.1 Metrics

This section describes the process followed to measure software quality on GitLab, explaining the tools used and how they were configured.

5.1.1 Coverage

For code coverage, GitLab takes data from external tools and shows the results obtained by them. The tools used to get code coverage depend on the technologies and languages involved, for example, some tools

can only be used for a specific programming language and others that can analyse coverage in several languages. These tools usually generate coverage reports automatically, every time unit tests are launched. These reports are stored in HMTL, CSV and XML format files which can be processed by GitLab later.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
eu.agricore.indexer.controller.api		74 %		72 %	11	35	22	107	6	26	1	5
eu.agricore.indexer.model		44 %		0 %	12	21	22	38	5	14	0	2
eu.agricore.indexer.security		79 %		83 %	2	10	21	75	1	7	0	2
eu.agricore.indexer.ldap.util		10 %		n/a	1	2	8	9	1	2	0	1
eu.agricore.indexer.dto		0 %		n/a	5	5	10	10	5	5	1	1
eu.agricore.indexer.ldap.dto		81 %		n/a	5	19	8	39	5	19	1	4
eu.agricore.indexer.ldap.model		82 %		n/a	1	11	6	30	1	11	0	1
eu.agricore.indexer		86 %		n/a	2	8	2	12	2	8	0	1
eu.agricore.indexer.ldap.service		92 %		100 %	1	12	1	24	1	11	0	1
eu.agricore.indexer.ldap.repository		98 %		50 %	3	27	2	134	0	24	0	2
eu.agricore.indexer.controller		42 %		n/a	2	3	2	3	2	3	0	1
eu.agricore.indexer.util		99 %		81 %	3	18	0	48	0	10	0	2
eu.agricore.indexer.service		100 %		100 %	0	13	0	27	0	10	0	2
eu.agricore.indexer.ldap.config		100 %		n/a	0	3	0	8	0	3	0	1
eu.agricore.indexer.config		100 %		n/a	0	3	0	7	0	3	0	2
eu.agricore.indexer.ldap.exception		100 %		n/a	0	1	0	2	0	1	0	1
Total	367 of 2.243	83 %	26 of 68	61 %	48	191	104	573	29	157	3	29

Figure 12 Code coverage report example

The example below explains how to use the ***Jacoco Code Coverage Library***[15] to compute coverage for code written in Java. Information about code coverage on GitLab and the tools used in each programming language can be consulted in GitLab's official documentation[16][17].

Once a specific coverage tool has been added in the source code, the next task consists in showing coverage total value on GitLab's merge requests processes. To do this, the following steps are required:

1. Enable test coverage parsing on GitLab. On project CI settings, a regular expression must be defined to let GitLab to find the test coverage output. The example below uses Jacoco regular expression given by GitLab.
2. On *gitlab-ci.yml* file, the path to Jacoco HTML report file must be specified on *test stage*, to allow GitLab to locate the coverage report and apply the regular expression.

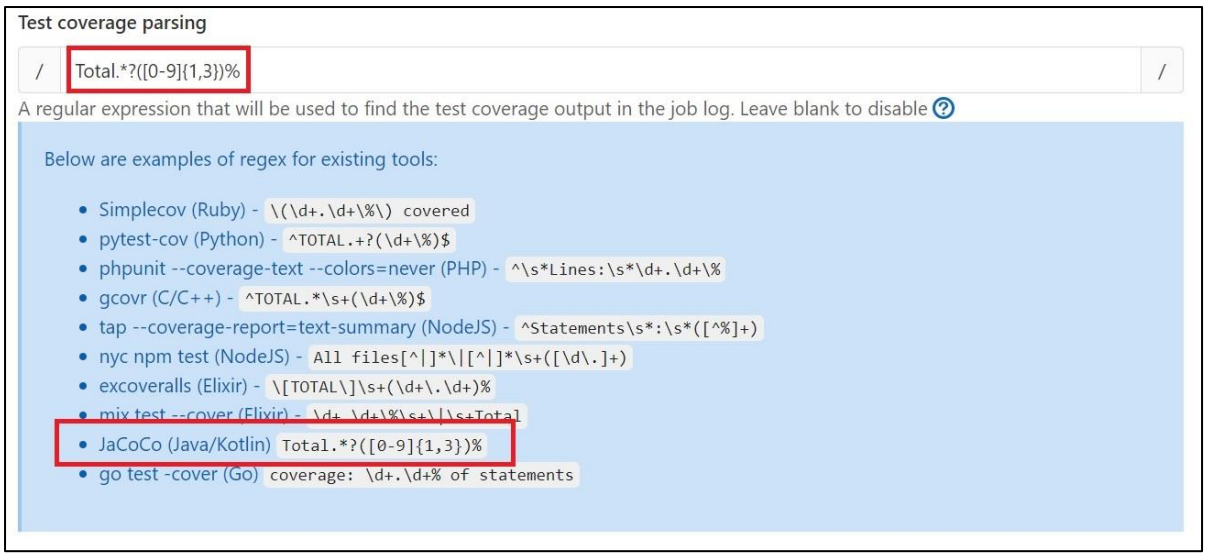

Figure 13 GitLab regular expression for coverage

Finally, when a merge request is opened, GitLab will run all unit tests created. This will generate a coverage report automatically and then, GitLab will take the overall result and show it in the details of the executed pipeline.

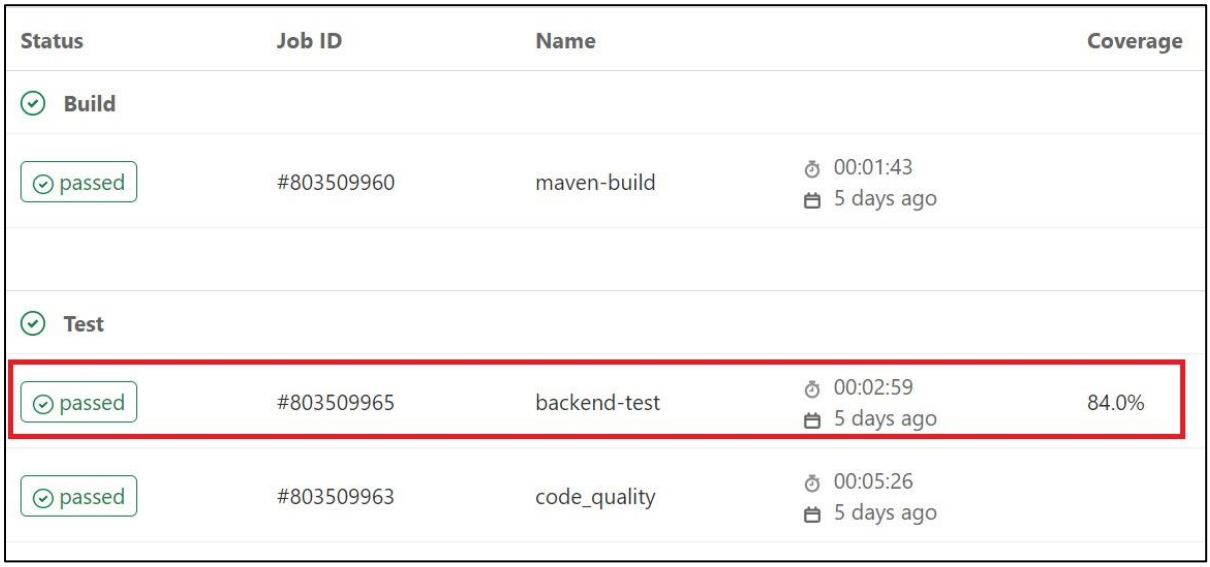

Figure 14 Unit testing coverage value on GitLab

5.1.2 Code quality

To ensure that the project code is kept simple, clean, readable and easy to understand, GitLab uses ***Code Climate Engines***[18][19] tool to analyse source code quality. Code Climate tool can be activated calling a template on *gitlab-ci.yml* file, enabling default configurations. A pipeline's stage must be also specified on GitLab CI file to determine when, the quality code process, will be executed.

Once the process has been executed, Code Climate tool generates a report as long as there is an existing report on the targeted branch, because Code Climate compares both reports to determine whether or not the code quality is being degraded. This allows developers to avoid merging the branches if quality is deteriorating. The report can be displayed on GitLab website or downloaded

as a JSON file. The following picture shows an example of a code quality report with some complexity and clarity issues found.

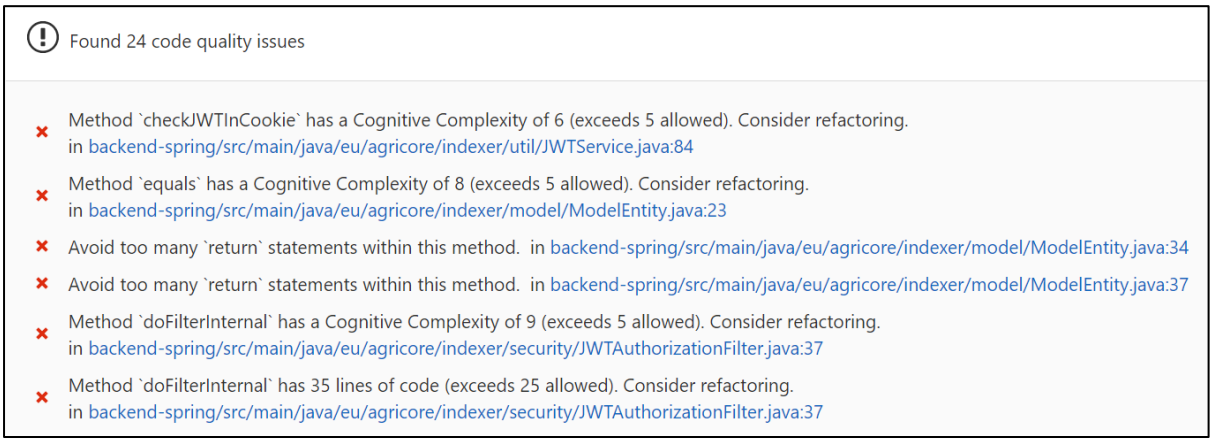

Figure 15 Code quality report on GitLab

5.2 Tests

This section describes the process followed to run tests automatically on GitLab when a merge request is opened, and how to display the details about that execution later.

5.2.1 Unit tests

To run all unit tests on merge requests, a new pipeline job must be defined. This job uses a specific command, which depends on the technology or programming language used, to launch each of the existing unit tests in the project. AGRICORE uses, as described in the introductory paragraphs of this chapter, *backend-test* job to launch unit tests associated with Java language. More information about running unit tests and displaying their results can be found in the official GitLab documentation[20].

A unit tests report can also be displayed on GitLab. When the tests are running, technologies usually store all details about tests on XML format files. To allow GitLab to collect unit tests reports, paths of the generated XML files must be specified on *gitlab-ci.yml* file. Details about tests passed or failed and the duration of each test will be displayed in the pipeline view on GitLab. The following figure shows an example of this:

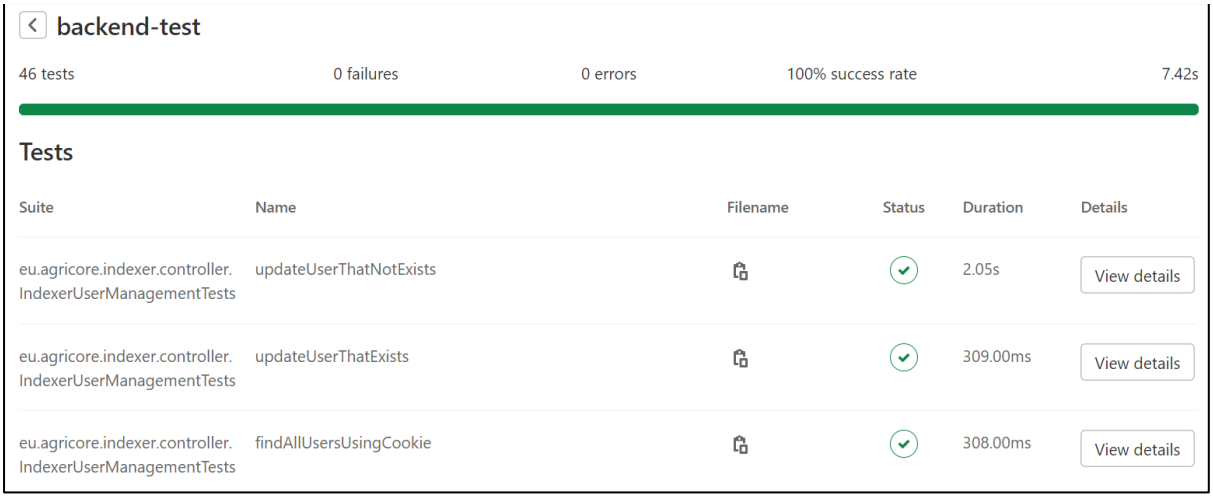

Figure 16 Unit tests report on GitLab

5.2.2 Integration and functional tests

As mentioned in the previous sections, at the current stage of the project, this process has not been implemented yet and included in the CI flow due to the level of maturity of the AGRICORE project. The previous process involved in the CI cycle has been designed and implemented to provide a proof that the CI processes add value to the software quality assurance process. However, it has been provided a hypothesis on how integration and functional tests could be executed and automatically integrated into the project development workflow.

All the modules that AGRICORE is composed by are stored in different, individuals and isolated repositories for each of them. In these repositories, all the metrics and unitary tests are calculated, executed, and measured using the guideline provided in the previous subsections of the present Continuous Integration section. Apart from this, there is a common repository called AGRICORE which is empty and does not house any modules. Using the potential of GitlabCI, the purpose of this repository could be defined as the integration of all the modules and the automatic execution of the integration and functional tests.

In order to define the expected result, the AGRICORE main repository could store each module in independent folders identified by its deliverable ID, and an extra folder could be created to store all the integration and functional tests definitions using automatic test execution technology simulating a user's interaction with such as Selenium tools[21]. As it was mention before, automatic instructions could be executed during the git workflow on each repository. By applying a branch management policy, each repository could integrate a pipeline in which when the source code is to be moved from development to master, the pipeline could push the source code to its respective folder within the common AGRICORE repository. With this solution, the AGRICORE repository could store each module in separate folders automatically.

The final step is to apply a new pipeline for the AGRICORE main repository, in which when a source code is pushed into the global repository, the execution of the integration and functional tests is applied. If GitlabCI provides all the mechanisms to execute these steps, all the metrics, unit tests, integration tests, functional tests and software measures could be executed in a single workflow.

During the development of the AGRICORE project, a more in-depth analysis and proof of concept will be carried out to verify that this mechanism can be implemented and integrated.

6 Test reports

This section will gather a list of the tests developed during the development of the AGRICORE project. At the current stage of the project, it is not possible to provide an exhaustive report of the results of the tests due to the still low maturity of the modules. So, in the next subsections, a list of analysed tests to be executed in the future is provided. The tests defined will be mainly focused on the AGRICORE ARDIT module, whose development is more advanced at the current stage of the project. As reference, the low-level requirements defined in GitLab for the ARDIT module will be used for the definition of tests. Tests management will be developed within the platform GitLab to have all requirements, test and development integrated in one single platform.

6.1 Functional tests

6.1.1 D1. ARDIT platform

6.1.1.1 ARDIT platform

Code	TEST.D1.FT.001. ARDIT public accessible platform
Requirements	• AG.D1.FR.001. Provide a publicly accessible index of agricultural data sources
Modules	• D1: ARDIT
Description	• Scenario: ARDIT published in the web • Given an anonymous user visits the ARDIT platform • When the anonymous user visits the platform • Then the user can search a dataset ○ And the user can see the details of a dataset
Result	Not tested yet

6.1.1.2 ARDIT user service

Code	TEST.D1.FT.002. User registration service
Requirements	• AG.D1.FR.001-1. User registration service
Modules	• D1: ARDIT
Description	• Scenario: ARDIT published • Given an anonymous user • When the anonymous user goes to the home page ○ And the user clicks in the "sign-in" section • Then a form is displayed to register the user on the platform
Result	Not tested yet

Code	TEST.D1.FT.002-1. Register new user
Requirements	• AG.D1.FR.001-1-1. Register a new user
Modules	• D1: ARDIT

Description	• Scenario: User not registered in ARDIT
	• Given an anonymous user visits the sign-in form
	• When the user enters its username
	  ○ And the user enters its password
	  ○ And the user enters its email address
	• Then the user sees a screen that indicates that an email has been sent to confirm the email address

Result	Not tested yet

Code	TEST.D1.FT.002-2. User verification
Requirements	• AG.D1.FR.001-1-2. User verification
Modules	• D1: ARDIT
Description	• Scenario: Anonymous user has been registered waiting to verify its email • Given the user receives an email sent by ARDIT to verify it • When the user clicks in the link • Then a message is displayed indicating that the user has been verified successfully.
Result	Not tested yet

6.1.1.3 ARDIT login service

Code	TEST.D1.FT.003. Login service
Requirements	• AG.D1.FR.001-2. Login service
Modules	• D1: ARDIT

Description	• Scenario: User registered in the system
	• Given a user visits the home page
	• When the user enters in the home page
	  ○ And the user sees a login button
	  ○ And the user clicks on it
	• Then a login form is displayed

Result	Not tested yet

Code	TEST.D1.FT.003-1. User login
Requirements	• AG.D1.FR.001-2-1. Login
Modules	• D1: ARDIT

Description	• Scenario: User registered in the system
	• Given a user visits the login form
	• When the user enters its username
	  ○ And the user enters its password
	• Then the user sees a message that indicates that the credentials are correct
	  ○ And the user sees the home page with the header changed with its username displayed

Result	Not tested yet

Code	TEST.D1.FT.003-2. User account recovering email
Requirements	• AG.D1.FR.001-2-2. Account recovering
Modules	• D1: ARDIT

Description	• Scenario: A user is registered in the system, but she cannot remember its password
	• Given a user visits the login form
	• When the user clicks in the recovering account message
	  ○ And a recovering form is displayed
	  ○ And the user enters its email in the platform
	• Then a message is displayed indicating that an email has been sent.

Result	Not tested yet

Code	TEST.D1.FT.003-2-1. User account recovering reset
Requirements	• AG.D1.FR.001-2-2. Account recovering
Modules	• D1: ARDIT

Description	• Scenario: A user is registered in the system, but she cannot remember its password
	• Given a user clicks in the recovery link provided by an email
	• When the user clicks in the email
	  ○ And a recovering form is displayed
	  ○ And the user enters its new password
	• Then a message is displayed indicating that the password has been changed
	  ○ And the user can log in with its credentials

Result	Not tested yet

6.1.1.4 ARDIT administration service

Code	TEST.D1.FT.004. Administration service
Requirements	• AG.D1.FR.001-3. Administration service
Modules	• D1: ARDIT
Description	• Scenario: An administrator is registered and logged in the system • Given an admin is in the home page • When the admin clicks in the admin section ○ And the admin clicks in the users' section • Then a page is displayed with the list of users
Result	Not tested yet

Code	TEST.D1.FT.004-1. Administrator creates a user
Requirements	• AG.D1.FR.001-3-1. Administrator creates a user
Modules	• D1: ARDIT

Description	• Scenario: An administrator is logged in the system
	• Given an admin is in the user administration section
	• When the admin clicks in the create a user section
	  ○ And enters a username
	  ○ And enters a password
	  ○ And enters an email
	• Then a message is displayed indicating that the user has been created
	  ○ And the user is displayed in the users list

Result	Not tested yet

Code	TEST.D1.FT.04-2. Administrator modifies a user
Requirements	• AG.D1.FR.001-3-2. Administrator modifies a user
Modules	• D1: ARDIT

Description	• Scenario: An administrator is logged, and a user is registered in the system
	• Given An administrator is logged in the user administration section
	• When the admin clicks in a user
	  ○ And the user details are displayed
	  ○ And the admin clicks in the edit user button
	  ○ And the admin modifies its user data
	• Then the user information is modified
	  ○ And the user information is displayed in the details view

Result	Not tested yet

Code	TEST.D1.FT.04-3. Administrator modifies basic user information
Requirements	• AG.D1.FR.001-3-2-1. Modify the basic user information
Modules	• D1: ARDIT

Description	• Scenario: An administrator is logged, and a user is registered in the system
	• Given An administrator is logged in the user administration section
	• When the admin clicks in a user
	  ○ And the user details are displayed
	  ○ And the admin clicks in the edit user button
	  ○ And the admin modifies its user data
	• Then the user information is modified
	  ○ And the user information is displayed in the details view

Result	Not tested yet

Code	TEST.D1.FT.04-4. Administrator assign a predefined role
Requirements	• AG.D1.FR.001-3-2-2. Assign a predefined role to a user
Modules	• D1: ARDIT

Description	• Scenario: An administrator is logged, and a user is registered in the system
	• Given An administrator is logged in the user administration section
	• When the admin clicks in a user
	  ○ And the user details are displayed
	  ○ And the admin clicks in the add role button
	  ○ And the admin selects the role 'maintainer'
	• Then the user role is modified
	  ○ And the new user role is displayed in the detailed view

Result	Not tested yet

Code	TEST.D1.FT.04-5. Administrator deletes a user
Requirements	• AG.D1.FR.001-3-3. Delete a user
Modules	• D1: ARDIT

Description	• Scenario: An administrator is logged, and a user is registered in the system
	• Given An administrator is logged in the user administration section
	• When the admin clicks in a user
	  ○ And the user details are displayed
	  ○ And the admin clicks in the delete user button
	  ○ And the admin confirms the operation
	• Then the user is deleted
	  ○ And the user is not displayed in the users list

	Not tested yet

6.1.1.5 ARDIT public features

Code	TEST.D1.FT.05. Available for all stakeholders
Requirements	• AG.D1.FR.002. Available for all stakeholders
Modules	• D1: ARDIT

Description	• Scenario: any given user accesses ARDIT webpage.
	• Given a registered or an anonymous user.
	• When the user accesses ARDIT website.
	  ○ And the user does not have an account
	  ○ Or is not logged in.
	  ○ Or is logged in.

	• Then the user can navigate and interact with the website and its features.
Result	Not tested yet

Code	TEST.D1.FT.05-1. Stakeholder can list datasets
Requirements	• AG.D1.FR.002-1. Anyone can list datasets.
Modules	• D1: ARDIT

Description	• Scenario: any given user wants to list datasets.
	• Given a registered or an anonymous user.
	• When the user accesses ARDIT website.
	  ○ And the user does not have an account.
	  ○ Or is not logged in.
	  ○ Or is logged in.
	• Then the user can search and list datasets without restrictions.

Result	Not tested yet

Code	TEST.D1.FT.05-2. Stakeholder can download an ETL
Requirements	• AG.D1.FR.002-2. Anyone can download ETL
Modules	• D1: ARDIT

Description	• Scenario: any given user wants to download an ETL.
	• Given a registered or an anonymous user.
	• When the user accesses to a dataset details page on ARDIT website.
	  ○ And the user does not have an account.
	  ○ Or is not logged in.
	  ○ Or is logged in.
	• Then the user can download ETL associated with a dataset without restrictions.

Result	Not tested yet

Code	TEST.D1.FT.05-3. Legal notice
Requirements	• AG.D1.FR.002-3. Anyone can access to global legal notice section
Modules	• D1: ARDIT

Description	• Scenario: any given user access to legal notice section.
	• Given a registered or an anonymous user.
	• When the user accesses to legal notice section on ARDIT website.
	  ○ And the user does not have account.
	  ○ Or is not logged in.
	  ○ Or is logged in.
	• Then the user can read the policy statement, the cookies policy or the copyright notice.

Result	Not tested yet

Code	TEST.D1.FT.05-4. Contact form
Requirements	• AG.D1.FR.002-4. Anyone can access to a contact form
Modules	• D1: ARDIT

Description	• Scenario: any given user access to the contact form section.
	• Given a registered or an anonymous user.
	• When the user accesses to the contact form section on ARDIT website.
	  ○ And the user does not have an account.
	  ○ Or is not logged in.

	○ Or is logged in. • Then the user can fill in the form and notify any problem or suggestion.
Result	Not tested yet

Code	TEST.D1.FT.05-5. Other websites section
Requirements	• AG.D1.FR.002-5. Anyone can access to other websites section
Modules	• D1: ARDIT
Description	• Scenario: any given user access to other websites section. • Given a registered or an anonymous user. • When the user accesses to other websites section on ARDIT website. ○ And the user does not have an account. ○ Or is not logged in. ○ Or is logged in. • Then the user gets links to other websites related to AGRICORE project.
Result	Not tested yet

Code	TEST.D1.FT.05-6. Help section
Requirements	• AG.D1.FR.002-6. Anyone can access to a help section
Modules	• D1: ARDIT
Description	• Scenario: any given user access to the help section. • Given a registered or an anonymous user. • When the user accesses to help section on ARDIT website. ○ And the user does not have an account. ○ Or is not logged in. ○ Or is logged in. • Then the user gets information about how to use the application.
Result	Not tested yet

6.1.1.6 Datasets services

Code	TEST.D1.FT.006. Store relevant information of the dataset
Requirements	• AG.D1.FR.003. Store relevant information of the dataset
Modules	• D1: ARDIT
Description	• Scenario: Anonymous user in the home view • Given An anonymous user in the home view • When she clicks in the search database button ○ And select a dataset from the list • Then a detailed view is displayed ○ And contains information about spatial scope ○ And contains information about the resolution ○ And contains information about the aggregation level ○ <WIP: Define all the relevant information of the datasets using DCAT-AP 2.0>
Result	Not tested yet

Code	TEST.D1.FT.006-1. Register a new dataset
Requirements	• AG.D1.FR.003-1. Register a new dataset
Modules	• D1: ARDIT
Description	WIP: Role/s to be defined • Scenario: User <Role> registered in the system • Given An <Role> user in the home view logged • When the <Role> user clicks in the Register dataset button • Then a form is displayed to characterise a new dataset
Result	Not tested yet

Code	TEST.D1.FT.006-1-1. Add information of a dataset
Requirements	• AG.D1.FR.003-1-1. Add information of a dataset
Modules	• D1: ARDIT
Description	WIP: Role/s to be defined • Scenario: User <Role> registered in the system • Given An <Role> user in the register a dataset view • When the <Role> fills all the information of the dataset ○ And the <Role> press the save button • Then a detailed view is displayed with the dataset registered
Result	Not tested yet

Code	TEST.D1.FT.006-1-2. Add and remove a <Role> of a dataset
Requirements	• AG.D1.FR.003-1-2. Add and remove a maintainer of a dataset
	• D1: ARDIT

Description	• WIP: Role/s to be defined
	  ○ Scenario: Two <Role> users registered in the system
	  ○ Given An <Role> user in the register a dataset view
	  ○ When the <Role> clicks in the button add <Role>
	  ▪ And selects another user
	  ▪ And press the save button
	  ○ Then a detailed view is displayed with the dataset registered
	  ▪ And the new <Role> user is displayed

Result	Not tested yet

Code	TEST.D1.FT.006-1-3. Add an ETL in a dataset
Requirements	• AG.D1.FR.003-1-3. Add an ETL in a dataset
	• D1: ARDIT

Description	WIP: Role/s to be defined
	• Scenario: User <Role> registered in the system
	• Given An <Role> user in the register a dataset view
	• When the <Role> fills all the information of the dataset
	  ○ And the <Role> press the button Add an ETL
	  ○ And the <Role> select an ETL from its computer
	  ○ And the <Role> press the save button
	• Then a detailed view is displayed with the dataset registered
	  ○ And the ETL file is displayed

Result	Not tested yet

Code	TEST.D1.FT.006-1-4. Add an ETL common errors comment
Requirements	• AG.D1.FR.003-1-5. Add an ETL common errors comment
	• D1: ARDIT

Description	WIP: Role/s to be defined
	• Scenario: User <Role> registered in the system
	• Given An <Role> user in the detailed view of a dataset that can edit
	• When the <Role> presses in the button Add ETL common errors comment
	  ○ And a form is displayed
	  ○ And the <Role> adds a comment
	  ○ And the <Role> presses the save button
	  ○ And the detailed view is displayed
	  ○ And the <Role> user press the ETL common errors button
	• Then a list of ETL common error comments is displayed
	  ○ And its comment is displayed in the list

Result	Not tested yet

Code	TEST.D1.FT.006-2. Edit a dataset
Requirements	• AG.D1.FR.003-2. Edit a dataset
	• D1: ARDIT

Description	WIP: Role/s to be defined
	• Scenario: User <Role> registered in the system
	• Given An <Role> user in the detailed view of a dataset that can edit
	• When the <Role> press the edit button
	  ○ And the <Role> modifies the name
	  ○ And the <Role> press the save button
	• Then the detailed view is displayed with the dataset modified

Result	Not tested yet

Code	TEST.D1.FT.006-3. Remove a dataset
Requirements	• AG.D1.FR.003-3. Remove a dataset
	• D1: ARDIT
Description	WIP: Role/s to be defined • Scenario: User <Role> registered in the system • Given An <Role> user in the detailed view of a dataset that can edit • When the <Role> press the delete button ○ And the <Role> confirms the operation • Then a message is displayed indicating that the dataset has been removed
Result	Not tested yet

Code	TEST.D1.FT.006-4. Set ETL as correct or incorrect
Requirements	• AG.D1.FR.003-4. Set ETL as correct or incorrect
	• D1: ARDIT
Description	WIP: Requirement to be defined
Result	Not tested yet

Code	TEST.D1.FT.006-5. Display a dataset
Requirements	• AG.D1.FR.003-7. Display a dataset
	• D1: ARDIT

Description	• Scenario: Anonymous user in the system
	• Given an anonymous user in the home page
	• When the user wants to search a dataset
	  ○ And the user presses the search button
	  ○ And the user selects a dataset
	• Then the detailed view of the dataset is displayed

Result	Not tested yet

Code	TEST.D1.FT.006-5-1. Navigate to the dataset link
Requirements	• AG.D1.FR.003-7-1. Navigate to the dataset link
	• D1: ARDIT
Description	• Scenario: Anonymous user in the system • Given an anonymous user detailed view of a dataset • When the user press in the link of the dataset • Then the user navigates to the web resource linked
Result	Not tested yet

Code	TEST.D1.FT.006-5-2. Display the number of views of a dataset
Requirements	• AG.D1.FR.003-7-2. Display the number of views of a dataset
	• D1: ARDIT
Description	• Scenario: Anonymous user in the system • Given an anonymous user in the home page • When the user wants to search a dataset ○ And the user presses the search button • Then the number of views of a dataset is displayed with other fields
Result	Not tested yet

Code	TEST.D1.FT.006-6. Update notifications
Requirements	• AG.D1.FR.003-8. Update notifications
	• D1: ARDIT
Description	• WIP: Requirement to be defined and priority WH
Result	Not tested yet

6.1.1.7 Scope extension

Code	TEST.D1.FT.007. Researchers will be able to extend its scope with additional datasets
Requirements	• AG.D1.FR.004. Researchers will be able to extend its scope with additional datasets
Modules	• D1: ARDIT

Description	• Scenario: any given user wants to suggest a new dataset.
	• Given a registered or an anonymous user.
	• When the user accesses to ARDIT website.
	  ○ And the user does not have account
	  ○ Or is not logged in.
	  ○ Or is logged in.

	• Then a link to a form page must be found to allow sending suggestions about new datasets.
Result	Not tested yet

Code	TEST.D1.FT.007-1. Anyone can suggest a new dataset
Requirements	• AG.D1.FR.004-1. Anyone can suggest a new dataset
Modules	• D1: ARDIT

Description	• Scenario: any given user suggests a new dataset.
	• Given a registered or an anonymous user.
	• When the user accesses to the dataset suggestion section on the webpage.
	  ○ And fills in the form shown (WIP).
	  ○ And clicks on the submit button (WIP).
	• Then the user can send a new suggestion to add a new dataset to the website.

Result	Not tested yet

6.1.1.8 Semantic search

Code	TEST.D1.FT.008. Semantic search will be allowed
Requirements	• AG.D1.FR.005. Semantic search will be allowed
Modules	• D1: ARDIT

Description	• Scenario: any given user wants to search for datasets.
	• Given a registered or an anonymous user.
	• When the user accesses to ARDIT website.
	  ○ And the user does not have an account.
	  ○ Or is not logged in.
	  ○ Or is logged in.
	• Then the user must be able to search for datasets using natural language.

Result	Not tested yet

Code	TEST.D1.FT.008-1. Search a dataset using natural language
Requirements	• AG.D1.FR.005-1. Search a dataset using natural language
Modules	• D1: ARDIT
Description	• Scenario: any given user searches for a dataset typing any text in an input. • Given a registered or an anonymous user. • When the user accesses to the home page on ARDIT website. ○ And types any text on a given search bar. • Then the user gets datasets whose names or attributes match the parameters searched.
Result	Not tested yet

6.1.1.9 Advanced search

Code	TEST.D1.FT.009. Advanced search will be allowed
Requirements	• AG.D1.FR.006. Advanced search will be allowed
Modules	• D1: ARDIT

Description	• Scenario: any given user wants to search for datasets using specific attributes or values.
	• Given a registered or an anonymous user.
	• When the user accesses to ARDIT website.
	  ○ And clicks on the advanced search section.
	  ○ And the user does not have an account.
	  ○ Or is not logged in.

	○ Or is logged in. • Then the user will be able to select specific attributes and values to filter the search.
Result	Not tested yet

Code	TEST.D1.FT.009-1. Search the results by its title and any of its properties
Requirements	• AG.D1.FR.006-1. Search the results by its title and any of its properties
Modules	• D1: ARDIT
Description	• Scenario: any given user wants to search for datasets by its title and any of its properties. • Given a registered or an anonymous user. • When the user accesses to the advanced search section on ARDIT website. ○ And the user types a dataset title or select some of its properties. • Then the user gets the resulting datasets based on selected search filters.
Result	Not tested yet

Code	TEST.D1.FT.009-2. Filter the results by date
Requirements	• AG.D1.FR.006-2. Filter the results by date
Modules	• D1: ARDIT
Description	• Scenario: any given user wants to search for datasets and sort them by date. • Given a registered or an anonymous user. • When the user accesses to the advanced search section on ARDIT website. ○ And the user selects to sort the datasets by date of inclusion in ascending order. • Then the user gets the resulting datasets sorted from oldest to newest.
Result	Not tested yet

Code	TEST.D1.FT.009-3. Filter the results by any property
Requirements	• AG.D1.FR.006-3. Filter the results by any property
Modules	• D1: ARDIT
Description	• Scenario: any given user wants to search for datasets by any property. • Given a registered or an anonymous user. • When the user accesses to the advanced search section on ARDIT website. ○ And selects "PH" as variable associated with the dataset. ○ And selects "Yearly" as periodicity of publications. • Then the users get datasets that include "PH" as variable and have an annual periodicity.
Result	Not tested yet

6.1.1.10 Local ARDIT and ETL execution

Code	TEST.D1.FT.010. Local ARDIT capability
Requirements	• AG.D1.FR.007. Local deployment capability
Modules	• D1: ARDIT
Description	• Scenario: ARDIT local • Given an anonymous user • When the user navigates to the ARDIT local URL • Then ARDIT platform is available in local
Result	Not tested yet
Code	**TEST.D1.FT.010-1. DWH connection**

Requirements	• AG.D1.FR.007-1. DWH connection
Modules	• D1: ARDIT • D2: DWH

Description	• Scenario: ARDIT local and an administrator registered
	• Given an administrator user logged in the system
	• When the admin goes to the settings section
	  ○ And the admin selects de DWH connection section
	  ○ And the admin modifies the configuration of the DWH connection
	  ○ And press the test connectivity button
	• Then a message is displayed that the connection has been established

Result	Not tested yet

Code	**TEST.D1.FT.010-2. ETL execution in the DWH**
Requirements	• AG.D1.FR.007-2. ETL execution in the DWH
Modules	• D1: ARDIT • D2: DWH
Description	• WIP: General tests for this requirement
Result	Not tested yet

Code	**TEST.D1.FT.010-2-1. Add ETL to the job queue**
Requirements	• AG.D1.FR.007-2-1. Add ETL to the job queue
Modules	• D1: ARDIT
Description	WIP: Role user to be defined • Scenario: User <Role> logged and a dataset with an ETL stored in the system. • Given A <Role> user logged in the details view of a dataset • When the user presses the add ETL to queue button • Then A message indicates that the ETL has been added to the queue ○ And the ETL has been added into the job queue list view
Result	Not tested yet

Code	**TEST.D1.FT.010-2-2. Remove ETL to the job queue**
Requirements	• AG.D1.FR.007-2-2. Remove ETL to the job queue
Modules	• D1: ARDIT
Description	WIP: Role user to be defined • Scenario: User <Role> logged and an ETL added in the job queue • Given a <Role> user in the ETL job queue view • When the user presses the remove button in the ETL row • Then the ETL is removed from the list
Result	Not tested yet

Code	**TEST.D1.FT.010-2-3. Display ETL job queue**
Requirements	• AG.D1.FR.007-2-3. Display ETL job queue
Modules	• D1: ARDIT
Description	WIP: Role user to be defined • Scenario: User <Role> registered • Given a <Role> user logged in the home view

	• When the user press in the ETL job queue section • Then the ETL job queue view is displayed
Result	Not tested yet

Code	TEST.D1.FT.010-2-4. Launch an ETL in the job queue launcher
Requirements	• AG.D1.FR.007-2-4. Launch an ETL in the job queue launcher
Modules	• D1: ARDIT • D2: DWH
Description	WIP: Role user to be defined • Scenario: User <Role> registered and a dataset with an ETL registered and added in the job queue • Given a <Role> user logged in the job queue view • When the user presses the launch job queue button • Then a message is displayed indicating that the launching is in process ○ And the ETL status has changed to launching
Result	Not tested yet

Code	TEST.D1.FT.010-2-5. ETL execution feedback
Requirements	• AG.D1.FR.007-2-5. ETL execution feedback
Modules	• D1: ARDIT • D2: DWH
Description	WIP: Role user to be defined • Scenario: User <Role> registered and a dataset with an ETL registered and added in the job queue • Given a <Role> user logged in the job queue view • When the user presses the launch job queue button • Then a message is displayed indication that the launching is in process ○ And the ETL status has changed to launching
Result	Not tested yet

Code	TEST.D1.FT.010-2-6. ETL launched feedback
Requirements	• AG.D1.FR.007-2-6. ETL launched feedback
Modules	• D1: ARDIT • D2: DWH
Description	WIP: Role user to be defined • Scenario: User <Role> registered and a dataset with an ETL registered and added in the job queue • Given a <Role> user logged in the job queue view • When ETL has been executed or stored in the DWH. • Then a message is displayed giving feedback to the user about the process.
Result	Not tested yet

Code	TEST.D1.FT.010-2-7. ETL execution only if it is correct
Requirements	• AG.D1.FR.007-2-7. ETL execution only if it is correct
Modules	• D1: ARDIT
Description	• WIP: Role user to be defined

	○ Scenario: User <Role> registered and a dataset with an ETL registered but tagged as invalid ○ Given a <Role> user logged in the detailed view of a dataset ○ When the user tries to press the add to job queue button ○ Then button is displayed as disabled ▪ And a message indicates that the ETL is not valid
Result	Not tested yet

Code	TEST.D1.FT.010-3. Local indexer database synchronization
Requirements	• AG.D1.FR.007-3. Local indexer database synchronization
Modules	• D1: ARDIT
Description	WIP: Role user and process definition
Result	Not tested yet

6.2 Integration tests

6.2.1 D1. ARDIT platform

Code	TEST.D1.IT.001. DWH connection
Requirements	• AG.D1.FR.007-1. DWH connection
Modules	• D1: ARDIT • D2: DWH
Description	The ARDIT local platform and the DWH must relate to the main goal of launch ETLs from ARDIT to DWH. The communication is bidirectional due to ARDIT must know the status of the ETL execution.
Result	Not tested yet

Code	TEST.D1.IT.002. ETL execution in the DWH
Requirements	• AG.D1.FR.007-2. ETL execution in the DWH
Modules	• D1: ARDIT • D2: DWH
Description	WIP: Communication protocol and message format ARDIT must send an operation to the DWH to execute an ETL.
Result	Not tested yet

Code	TEST.D1.IT.003. ETL execution feedback
Requirements	• AG.D1.FR.007-2-5. ETL execution feedback
Modules	• D1: ARDIT • D2: DWH
Description	WIP: Communication protocol and message's format ARDIT must receive information about the status of the ETL during its execution
Result	Not tested yet

Code	TEST.D1.IT.004. ETL launched feedback
Requirements	• AG.D1.FR.007-2-6. ETL launched feedback
Modules	• D1: ARDIT • D2: DWH
Description	WIP: Communication protocol and message's format ARDIT must receive information about the final result of the ETL execution in the DWH
Result	Not tested yet

Code	TEST.D1.IT.005. ETL isolated tracked environment
Requirements	• AG.D1.FR.007-2-8 ETL isolated tracked environment
Modules	• D1: ARDIT • D2: DWH
Description	WIP: Communication protocol, message's format, and definition about how to isolate the context of each ETL ARDIT have to send information about how the ETL must be isolated from the other executions.
Result	Not tested yet

Code	TEST.D1.IT.006. Local indexer database synchronization
Requirements	• AG.D1.FR.007-3. Local indexer database synchronization
Modules	• D1: ARDIT

Description	WIP: Message's format and definition of how the synchronization is going to be performed, as well as the information to synchronize.
	Local ARDIT must synchronize its database with the Global ARDIT published.
	The local ARDIT platform must retrieve the following information:
	• Datasets:
	  ○ New public datasets
	  ○ Datasets modified in global ARDIT that has not been modified or removed in the local platform
	• Vocabularies:
	  ○ New vocabularies
	  ○ Vocabularies modified
	  ○ Removed vocabularies

Result	Not tested yet

7 Conclusions

The deliverable 6.6 has provided a guideline of how the software quality assurance is going to be applied and monitored during the project life cycle, as well as guidelines and mechanism to decrease the risk of errors during the development and integration processes of the modules. Besides, this metric also increases the quality of the code in terms of robustness, readability, and best practices.

These guidelines achieve the goal of guide the development of all the individual modules, avoiding any last-minute integration problems using advanced mechanisms such as automatic software metrics measures, automatic test execution and a flexible workflow designed and established according to the project needs.

Due to this deliverable is a first version developed at M15, it will be updated with new tests and improvements of the guidelines offered to satisfy and increase the performance during the project development.

8 References

1. ^ IEEE, "IEEE Standard for Software Quality Assurance Processes," IEEE Std 730-2014 (Revision of IEEE Std 730-2002), pp. 1–138, 2014, DOI: 10.1109/IEEESTD.2014.6835311.
2. ^ GitLab, "Upcoming changes to CI/CD Minutes for free-tier users on GitLab.com." [Online]. Available: https://about.gitlab.com/releases/2020/09/01/ci-minutes-update-free-users/.
3. ^ S. Chacon and B. Straub, Pro Git, 2nd ed. Apress, 2020, pp. 64–106.
4. ^ GitLab, "Merge request process documentation on GitLab.com." [Online]. Available: https://docs.gitlab.com/ee/user/project/merge_requests/.
5. ^ V. Driessen, "GitFlow branching model." [Online]. Available: https://nvie.com/posts/a-successful-git-branching-model/.
6. ^ Git, "Pro Git book available online on Git website." [Online]. Available: https://git-scm.com/book/es/v2.
7. ^ P. Bourque and R. E. Fairley, SWEBOK Guide to the Software Engineering Body of Knowledge, 3rd ed. IEEE, 2014, p. 174.
8. ^ ISQTB, "Integration test definition on isqtb.org." [Online]. Available: https://glossary.istqb.org/en/term/integration-testing-2.
9. ^ ISQTB, "Functional test definition on isqtb.org." [Online]. Available: https://glossary.istqb.org/en/search/functional%20testing.
10. ^ Cucumber, "Gherkin syntax reference guide on cucumber.io." [Online]. Available: https://cucumber.io/docs/gherkin/reference/.
11. ^ ISQTB, "Performance test definition on isqtb.org." [Online]. Available: https://glossary.istqb.org/en/term/performance-testing-2.
12. ^ M. Rouse, "Performance testing metrics." [Online]. Available: https://searchsoftwarequality.techtarget.com/definition/performance-testing.
13. ^ Apache, "Apache JMeter tool for testing." [Online]. Available: https://jmeter.apache.org/.
14. ^ GitLab, "Continuous integration official guidelines on gitlab.com." [Online]. Available: https://docs.gitlab.com/ee/ci/.
15. ^ EclEmma, "Jacoco Coverage Library official website." [Online]. Available: https://www.eclemma.org/jacoco/.
16. ^ GitLab, "Test coverage parsing official documentation on gitlab.com." [Online]. Available: https://docs.gitlab.com/ee/ci/pipelines/settings.html#test-coverage-parsing.
17. ^ GitLab, "Some test coverage tools examples in the official documentation of GitLab." [Online]. Available: https://docs.gitlab.com/ee/user/project/merge_requests/test_coverage_visualization.html.
18. ^ C. Climate, "Quality analysis documentation on codeclimate.com." [Online]. Available: https://docs.codeclimate.com/docs.
19. ^ GitLab, "Code quality reports using Code Climate tool on gitlab.com." [Online]. Available: https://docs.gitlab.com/ee/user/project/merge_requests/code_quality.html.
20. ^ GitLab, "Unit testing reports on gitlab.com." [Online]. Available: https://docs.gitlab.com/ee/ci/unit_test_reports.html.
21. ^ Selenium, "Tools for automatic integration and functional test execution on selenium.dev." [Online]. Available: https://www.selenium.dev/.

Apart from these references, for preparing this report, the following documents have been taken into consideration:

- AGRICORE Proposal: project proposes a novel tool for improving the current capacity to model policies dealing with agriculture by taking advantage of the latest progresses in modelling approaches and ICT.
- AGRICORE Grant Agreement ANNEX 1 Part A and B, Research and Innovation action, Number-816078: Official Grant Agreement of the AGRICORE project, which defined the terms and conditions of the project, as well as the main requirements of the project.

Deliverable Number	Deliverable Title	Lead beneficiary	Type	Dissemination Level	Due date
D4.1	AGRICORE requirements and project management platform	AAT	Report	Public	M12 31 Aug 2020