D6.9 – Continuous Integration and Validation

WP6 – EVALUATE: Piloting and Demonstrating

i4Q has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958205.

Document Information

GRANT AGREEMENT NUMBER	958205	ACRONYM	i4Q
FULL TITLE	Industrial Data Services for Quality Control in Smart Manufacturing		
START DATE	01-01-2021	DURATION	36 months
PROJECT URL	https://www.i4q-project.eu/		
DELIVERABLE	D6.9 – Continuous Integration and Validation		
WORK PACKAGE	WP6 – EVALUATE: Piloting and Demonstrating		
DATE OF DELIVERY	CONTRACTUAL 31-Dec-2022	ACTUAL	30-Dec-2022
NATURE	Report	DISSEMINATION LEVEL	Public
LEAD BENEFICIARY	ITI		
RESPONSIBLE AUTHOR	ITI		
CONTRIBUTIONS FROM	1-CERTH, 2-ENG, 3-IBM, 4-ITI, 5-KBZ, 6-EXOS, 7-IKER, 8-BIBA, 9-UPV, 10-TUB, 11-UNI		
TARGET AUDIENCE	1) i4Q Project partners; 2) industrial community; 3) other H2020 funded projects; 4) scientific community		
DELIVERABLE CONTEXT/ DEPENDENCIES	This document has a second iteration in Jun 2023 (D6.17) and a third iteration in Dec 2023 (D6.18). Its relationship to other documents is as follows: - Deliverables from the Build Work Packages: WP3, WP4 and WP5. - Deliverables from the Evaluate Work Package: WP6. - D9.4 Ops Setup and Quality Control Report v1.		
EXTERNAL ANNEXES/ SUPPORTING DOCUMENTS	None		
READING NOTES	None		
ABSTRACT	This document is the first release, of a set of three, of the Continuous Integration and Validation report. This first release provides an overview of the methodology to be followed to develop the software, and comprehensive guidelines to solution providers of the technical requirements to integrate the different solutions and pilots. This task will finish with the third release of the document by the end of the project implementation and it will include a high-level view of the degree of completion of the integration and validation activities of		

	i4Q solutions and pilots including software security and quality management.

Document History

VERSION	ISSUE DATE	STAGE	DESCRIPTION	CONTRIBUTOR
0.1	29-Sep-2022	ToC	Table of content sent to task Vice-Leader ENG	ITI
0.2	07-Nov-2022	Draft	First version sent to revision	ITI
0.3	14-Nov-2022	Draft	Comments by Vice-Leader	ENG
0.4	15-Nov-2022	Draft	3.3 Message broker included	CERTH
0.5	25-Nov-2022	Draft	Version ready for internal review	ITI
0.6	07-Dec-2022	Internal Review	Comments from reviewers received	TIAG, ENG
0.7	14-Dec-2022	Draft	Draft version with modifications ready for final review	ITI
1.0	30-Dec-2022	Final Document	Final quality check and issue of final document	CERTH

Disclaimer

Copyright message

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS/ACRONYMS

API	Application Programming Interface
ACLs	Access Control Lists
CD	Continuous Delivery
CI	Continuous Integration
CI/CD	Continuous integration, delivery and deployment
CNC	Computer Numerical Control
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
REST	Representational State Transfer
RIDS	Reliable Industrial Data Services
SCM	Source Code Management
SWEBOK	Software Engineering Body of Knowledge
TLS	Transport Layer Security
TN	Trusted Networks
WP	Work Package

Executive summary

i4Q Project aims to provide a complete set of solutions consisting of IoT-based Reliable Industrial Data Services (RIDS), the so called 22 i4Q Solutions, able to manage the huge amount of industrial data coming from cheap cost-effective, smart, and small size interconnected factory devices for supporting manufacturing online monitoring and control.

Besides, different pipelines including some of the i4Q Solutions will be used in order to improve the current industrial processes of the following pilot scenarios:

- **Pilot 1**: Smart Quality in CNC Machining.
- **Pilot 2**: Diagnostics and IoT Services.
- **Pilot 3**: White Goods Product Quality.
- **Pilot 4**: Aeronautics and Aerospace Metal Parts Quality.
- **Pilot 5**: Advanced In-line Inspection for incoming Prime Matter Quality Control.
- **Pilot 6**: Automatic Advanced Inspection of Automotive Plastic Parts.

In this project, the development work is aimed at producing high-quality code and enhancing the integration of all the i4Q Solutions in the pilots' pipelines. The current deliverable provides a general overview of the methodologies and approaches followed in the project to achieve these goals while building the i4Q Solutions.

Furthermore, this document explains the aspects monitored during the implementation of the i4Q Solutions to keep track of the progress made so far at each moment. In this project the focus is, especially- on, the requirements coverage, security issues, quality management, integration and data and risks issues.

Finally, this deliverable gives a comprehensive high-level view on the work to be carried out for the integration and validation phase of the project. That is, the steps to be followed by the i4Q solution providers and the industrial partners, to implement the pipelines defined for each pilot within their industrial processes, and to validate the impact of the solutions in such processes. Note that the integration and validation of the i4Q Solutions has not started, since this work will start in M25. Thus, the results of implementing these tasks, as well as detailed information regarding the possible changes in the methodologies or procedures to be followed during that phase, and the results of the integration and validation work will be presented in the subsequent releases of this document, by M30 and M36.

Document structure

Section 1: Introduction. In this section, the purpose and the approach of the Integration and Validation procedures in i4Q Project is described.

Section 2: Software development. This section shows a general overview of i4Q software development, explaining the methodologies followed and the tools used for the most relevant aspects.

Section 3: Solutions Integration. This section presents the technical requirements which have to be met by any i4Q Solution in order to be integrated in any of the pilots, and the activities and procedures shared by the different development teams.

Section 4: Solutions Overview. This section displays an overview of the i4Q Solutions developed in terms of requirements coverage, security issues, quality management, integration and data and risks issues.

Section 5: Integration and Validation phase overview. This section provides an overview of the integration and validation of the i4Q Solutions in the context of the seven pilots defined in this project (the six industrial uses cases and the generic pilot defined in T6.7). That is, the steps that will be performed and a preliminary calendar of this phase.

Section 6: Conclusions. Summarises the main results of the deliverable.

Appendix I: Step-by-step guide for building and publishing software solutions as Docker images.

1. Introduction

The main objective of task T6.8 is to perform integration and functionality tests in order to solve possible integration problems between the solutions from the BUILD Work Packages:

- WP3 Manufacturing Data Quality.
- WP4 Manufacturing Data Analytics for Manufacturing Quality Assurance.
- and WP5 Rapid Manufacturing Line Qualification and Reconfiguration.

A total of 22 i4Q Solutions will be produced and tested using 6 industrial pilot cases corresponding to the partners of this project, plus the generic pilot defined in T6.7.

The implementation of this task will allow to detect functional and/or integration problems at an early stage. Since this feedback can be given to the development tasks, those problems can be addressed before the official release of i4Q Solutions. Consequently, this task reduces the risk of facing those problems after the project finishes which, at the end of the day, could be expensive to solve in the future. Therefore, this task contributes to improve the solution's quality and reliability and makes them more likely to be used in real industrial scenarios.

In addition, modifications to the functionality of the use cases can be proposed, as long as they result in an improvement of the solutions.

Considering the number of software solutions, and the heterogeneity of the development teams, the implementation of the i4Q Solutions is a challenging work. Therefore, before starting the validation and integration of the i4Q Solutions, it is necessary to define a common approach to the development of the different solutions that facilitates this task.

This deliverable includes a general overview of the methodologies applied in the project to achieve T6.8 objectives. That is, it explains in detail the procedures followed during the implementation of the i4Q Solutions up to M24, the main aspects of software development being monitored in this project, and the tools used in each case. If solutions need to be further modified during the integration and validation phase, development teams will proceed in the same way.

Finally, this document explains the steps to be followed by the i4Q solution providers and the industrial partners during the integration and validation phase, to implement the pipelines defined for each pilot within their industrial processes, and to validate the impact of the solutions in such processes.

Note that the implementation of the i4Q Solutions is being released at the end of M24. Thus, the integration and validation have not started yet. The results of implementing these tasks, as well as detailed information regarding possible changes in the methodologies to followed during that phase, will be presented in the subsequent releases of this document, by M30 and M36.

Finally, a handbook will be generated for interested parties with information concerning the solutions deployment.

2. Software development

i4Q is a large project involving many different organisations where software solutions play a crucial role, and the approach taken to manage all these elements is not trivial. Several challenges are tackled in i4Q in this respect:

- Manage, integrate, combine, and/or customize a large set of heterogeneous software-based services, tools, and applications.
- Teamwork with multinational, multilingual, and multicultural teams.
- Handle different levels of expertise and skills in the development teams.

In order to address all these challenges, a common software development approach has been defined and followed in i4Q. This approach is aimed at enhancing consistency among the different solutions to facilitate their integration in further phases of the project and their adoption in real use case scenarios once the project finishes.

The rest of this section explains in detail the software development approach followed in this project. Firstly, the different development activities performed are exposed in Section 2.1. Then, the software development approach is discussed in more depth in Section 2.2, explaining the most relevant issues tackled, the specific methodology followed, and the set of tools used in each case.

2.1 Development activities reporting

i4Q has a variety of interconnected software solutions. As a result, there are a large number of deliverables and documents to report the development activities. The majority of these deliverables are related to the development of the i4Q Solutions, but some others correspond to the pilot use cases (whose pipelines involve different solutions) and other technical tasks of the project, such as T6.8, T9.4, for example. In addition, most of these deliverables have 2 formal iterations. In some cases, these are in months 18 and 24 (M18 and M24), and in other these are in months 18 and 36 (M18 and M36).

For easier finding of the information, **Table 1** has been defined, which summarises the topics covered in each one of the i4Q deliverables. The information contained in this table has been extracted from the European Commission's Amendment No AMD-958205-16 [1], and is composed by the following columns:

- **WP.** Indicates the Work Package to which a given deliverable belongs.
- **Deliverable Title.** Name of the deliverable.
- **ID M18.** ID of the first version (v1) of the deliverable, which is due in month 18 (M18).
- **ID M24.** ID of the second version (v2) of the deliverable, which is due in month 24 (M24).
- **ID M36.** ID of the second version (v2) of the deliverable, which is due in month 36 (M36).
- **Deliverable Description.** Contains a brief description of the deliverable's content.
- **Type.** Indicates the document type of the deliverable.

WP	Deliverable Title	ID M18	ID M24	ID M36	Deliverable Description	Type
3	i4Q Data Quality Guidelines	D3.1	D3.9	-	i4Q Data Quality Guidelines, versions v1 and v2	Report
3	i4Q QualiExplore for Data Quality Factor Knowledge	D3.2	D3.10	-	i4Q QualiExplore for Data Quality Factor Knowledge, versions v1 and v2	Other
3	i4Q Blockchain Traceability of Data	D3.3	D3.11	-	i4Q Blockchain Traceability of Data, versions v1 and v2	Other
3	i4Q Trusted Networks with Wireless & Wired Industrial Interfaces	D3.4	D3.12	-	i4Q Trusted Networks with Wireless & Wired Industrial Interfaces, versions v1 and v2	Other
3	i4Q Cybersecurity Guidelines	D3.5	D3.13	-	i4Q Cybersecurity Guidelines, versions v1 and v2	Report
3	i4Q IIoT Security Handler	D3.6	D3.14	-	i4Q IIoT Security Handler, versions v1 and v2	Other
3	i4Q Guidelines for Building Data Repositories for Industry 4.0	D3.7	D3.15	-	i4Q Guidelines for Building Data Repositories for Industry 4.0, versions v1 and v2	Other
3	i4Q Data Repository	D3.8	D3.16	-	i4Q Data Repository, versions v1 and v2	Other
4	i4Q Data Integration and Transformation Services	D4.1	D4.9	-	i4Q Data Integration and Transformation Services, versions v1 and v2	Other
4	i4Q Services for Data Analytics	D4.2	D4.10	-	i4Q Services for Data Analytics, versions v1 and v2	Other
4	i4Q Big Data Analytics Suite	D4.3	D4.11	-	i4Q Big Data Analytics Suite, versions v1 and v2	Other
4	i4Q Analytics Dashboard	D4.4	D4.12	-	i4Q Analytics Dashboard, versions v1 and v2	Other

4	i4Q AI Models Distribution to the Edge	D4.5	D4.13	-	i4Q AI Models Distribution to the Edge, versions v1 and v2	Other
4	i4Q Edge Workloads Placement and Deployment	D4.6	D4.14	-	i4Q Edge Workloads Placement and Deployment, versions v1 and v2	Other
4	i4Q Infrastructure Monitoring	D4.7	D4.15	-	i4Q Infrastructure Monitoring, versions v1 and v2	Other
4	i4Q Digital Twin	D4.8	D4.16	-	i4Q Digital Twin, versions v1 and v2	Other
5	i4Q Data-Driven Continuous Process Qualification	D5.1	D5.7	-	i4Q Data-Driven Continuous Process Qualification, versions v1 and v2	Other
5	i4Q Rapid Quality Diagnosis	D5.2	D5.8	-	i4Q Rapid Quality Diagnosis, versions v1 and v2	Other
5	i4Q Prescriptive Analysis Tools	D5.3	D5.9	-	i4Q Prescriptive Analysis Tools, versions v1 and v2	Other
5	i4Q Manufacturing Line Reconfiguration Guidelines	D5.4	D5.10	-	i4Q Manufacturing Line Reconfiguration Guidelines, versions v1 and v2	Report
5	i4Q Manufacturing Line Reconfiguration Toolkit	D5.5	D5.11	-	i4Q Manufacturing Line Reconfiguration Toolkit, versions v1 and v2	Other
5	i4Q Manufacturing Line Data Certification Procedure	D5.6	D5.12	-	i4Q Manufacturing Line Data Certification Procedure, versions v1 and v2	Other
6	Pilot 1: Fidia – Smart Quality in CNC Machining	D6.1	-	D6.11	Pilot 1: Fidia – Smart Quality in CNC Machining, versions v1 and v2	Demonstrator
6	Pilot 2: Biesse – Diagnostics and IoT Services	D6.2	-	D6.12	Pilot 2: Biesse – Diagnostics and IoT Services, versions v1 and v2	Demonstrator

6	Pilot 3: Whirlpool – White Goods Product Quality	D6.3	-	D6.13	Pilot 3: Whirlpool – White Goods Product Quality, versions v1 and v2	Demonstrator
6	Pilot 4: Factor – Aeronautics and Aerospace Metal Parts Quality	D6.4	-	D6.14	Pilot 4: Factor – Aeronautics and Aerospace Metal Parts Quality, versions v1 and v2	Demonstrator
6	Pilot 5: RiaStone – Advanced In-line Inspection for incoming Prime Matter Quality Control	D6.5	-	D6.15	Pilot 5: RiaStone – Advanced In-line Inspection for incoming Prime Matter Quality Control, versions v1 and v2	Demonstrator
6	Pilot 6: Farplas – Automatic Advanced Inspection of Automotive Plastic Parts	D6.6	-	D6.16	Pilot 6: Farplas – Automatic Advanced Inspection of Automotive Plastic Parts, versions v1 and v2	Demonstrator
6	i4Q Solutions Demonstrator	-	D6.7	D6.10	i4Q Solutions Demonstrator, versions v1 and v2	Demonstrator
9	Technical WP Reports	D9.3	-	D9.6	Technical WP Reports, versions v1 and v2	Report
9	Ops Setup and Quality Control Report	D9.4	-	D9.7	Ops Setup and Quality Control Report, versions v1 and v2	Report

Table 1. i4Q deliverables summary table

2.2 Approach

The software development approach followed in this project is based on task T9.5, in which it has been agreed how the source code should be managed and the tools to be used. In the following subsections it is discussed how these agreements have been implemented.

2.2.1 Methodology

i4Q is a large project where there is a need to coordinate multiple, geographically dispersed, development teams from different companies and integrate a large number of software components. Waterfall model offers a very well-structured methodology; however, it may not fit completely to i4Q since:

- It is a long research project, three years, where needs coming from industry may require software changes at any time.

- It is needed to work closely with industry to not lose focus on development work.
- Early prototypes and several development iterations will be needed to get the expected outcomes.

At the same time, whilst there are clear advantages with an agile approach, it may not work well at the end since i4Q requires solid foundations at early stages (requirements and analysis). Using a pure agile approach may end up in highly focused solutions on i4Q industry users, and not generic solutions valid for any manufacturing industry wanting to achieve the zero-defect goal and the technical partners' long-term goals.

Thus, a hybrid methodology is proposed to be used in i4Q, a mixture of **Waterfall**, and more agile approaches such as **Prototyping and SCRUM**.

In fact, Waterfall methodology has been used during the first 24 months of the project, Firstly, for the analysis phase, where use cases were identified, and requirements collected. Secondly, for the i4Q Solutions design and implementation phase, whose main result is a working version of the different i4Q Solutions.

After M24, a different methodology will be used, as the work on the i4Q Solutions starts the continuous integration and validation phase. During this phase, it might be necessary to perform minor updates to the solutions implementations that would need to be delivered as soon as possible. Since Waterfall methodologies are not the most appropriate for this type of work, it has been decided to use agile methodologies based on prototyping and SCRUM.

SCRUM [2] is a management methodology that enables organisations to generate value through adoptable solutions to complex problems. Participants in these organisations must work in teams and in iterations, so that features can be processed in an agile and flexible way. To achieve this goal, the following roles must be assigned.

- **Product Owner**. This represents the customer and provides features in the form of User Stories, which are prioritized and added to a list called Product Backlog. From this Product Backlog, User Stories are used as tasks for the next implementation period, called Sprints.
- **SCRUM Master**. Is a person who helps the development team by fulfilling their tasks using SCRUM in the right way and tries to avoid any distraction that may influence the performance of the development team.
- **Development Team**. Implements the specified Pilots from the User Scenarios. These pilots are split into smaller tasks that can be finished in a specified interval – normally in one day. The User Stories should be finished within the Sprint interval. Each of the domain pilots have a vice-leader, who will play a lead integration role cutting across all domains.

2.2.2 Source Code Management

Source Code Management (SCM) is the way to store, share, and work with the source code generated in the implementation phase of the project. Storing the source code in a secure way is important as it implies how the source code can be accessed and how changes are tracked.

In i4Q, GitLab [3] is used as an SCM tool, which is currently state-of-the-art in this area, proven, and widely adopted. It allows easy versioning and managing of files via branches and commands such as commit, push, and merge. There are other alternatives which were popular in the past

like Apache Subversion [4] or CVS [5]. Some of them can be used for free, like Mercurial [6]. However, the advantages of using GitLab are several: saves developer time, easy to undo changes, total control of commits, easy to review code, work offline, etc. [7], [8], [9].

GitLab helps to visualise the different projects/solutions. Furthermore, it supports the development process of the i4Q Solutions by enabling the simultaneous work of different members of the project and the management of the different versions. In addition, GitLab can easily be integrated into all Integrated Development Environments (IDEs) used by the partners or to be used in the command line by all partners not using IDEs, providing all the necessary features expected by the i4Q solution providers. The i4Q GitLab homepage is shown in **Figure 1** of the project is also displayed.

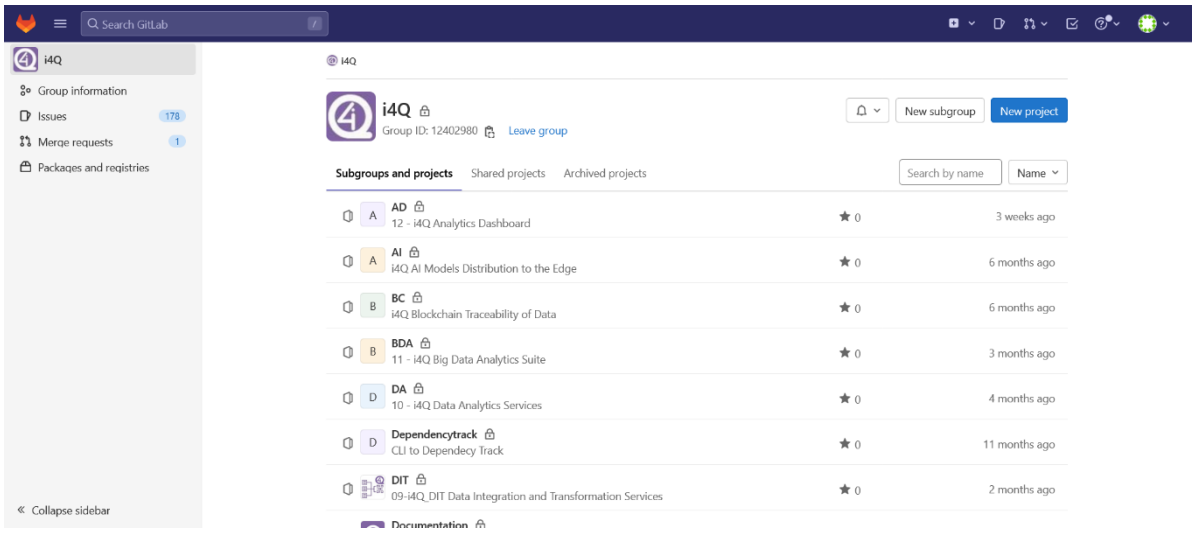

Figure 1. i4Q GitLab homepage

In i4Q, a **GitLab project** has been created for each one of the solutions identified in the proposal. The solution provider is responsible to keep their GitLab project up to date with all the information (repository files, issues, etc.), and following the guidelines established in this document.

Each project has a set of features available, such as its own shared repository, issues, etc. Files can also be directly edited simplifying small changes.

2.2.3 Continuous Integration

Continuous Integration (CI) is the process of systematically compiling and deploying software components after changes have been made to the source code. The goal of this process is enhancing the software quality by running unit tests for each build and therefore providing feedback quickly to the developer if compilation errors or functional errors arise. Changes in the Source Code Management can trigger builds of the corresponding component and by using integrated software tests, functional errors at build time can be revealed.

There are several options for CI. First, builds can be manually triggered, which is obviously not an optimal solution since developers do not take advantage of any kind of automation. The second one is to use Jenkins [11], which is a well-known tool in the area and extremely popular. However,

this approach requires using a plugin to work seamlessly with GitLab. The third and last one is to use GitLab CI/CD [12], the native application of GitLab for Continuous Integration.

GitLab CI/CD provides a central overview of all components and makes their status viewable by all project partners. It creates deployable artefacts (e.g., web applications, services, etc.) using the repositories provided by the Source Code Management System Git. Each entity represents an i4Q solution providing the required functionalities. These entities are deployed in an automatic way to provide the functionality of i4Q.

In i4Q, GitLab CI/CD is selected, since it allows zero effort integration with GitLab and does not depend on any plugin (which may need update efforts to avoid security breaches). Each solution provider will need to configure the right pipelines in GitLab according to their needs.

Continuous integration is just one of the DevOps practices, which seeks continuous improvement and process automation. Other well-known practices are continuous delivery and continuous deployment, which are defined below.

Continuous delivery (CD) [13] is the automatic deployment of changes made in a production or test environment after the new code has been compiled. In this way, an application can be deployed to production at any time at the push of a button, releasing small parts that are easy to fix if a problem is detected.

Continuous deployment [14] is a software release strategy that allows every change that passes the stages of the production pipeline to be sent to customers in an automated way. On the other hand, if the modifications introduced do not pass all the tests, they will not be deployed in the production environment. In this way, faster customer feedback is achieved, and the team can focus on software development, reducing the pressure to reach a release date.

2.2.4 Issue tracking

Issue tracking is the process of recording, managing and finally solving issues concerning an entity. Issues can also represent features, goals, discussions, and enhancements. The criticality, due dates, assigned programmers for an issue, etc. are managed.

In i4Q, GitLab Issues [15] is the selected tool for issue tracking purposes. The fact that it is already part of GitLab, and it is completely integrated with the rest of the tools proposed, makes it the most appropriate choice.

In i4Q, a board of issues with a set of labels has been created for each one of the GitLab projects. With these issues, a solution roadmap has been defined, so that each issue represents a micro task carried out in the project.

The i4Q solution issue detail must contain a list of the requirements that are directly tackled. If new requirements emerge during the course of the project, new issues must be opened to cover them.

Solution leaders are responsible to keep the issues list up to date and assign responsibilities to the corresponding participants.

In **Table 2** a set of i4Q labels is proposed. The solution leaders may customise these labels (modify, remove, or add) according to their particular needs.

Type	Label	Description
Status	To Do	The issue is part of the pending tasks, not started yet.
	Doing	The issue is part of an ongoing task.
	Reviewing	The issue is in revision or in testing or integration phase.
Priority	High	The issue requires to be tacked as soon as possible, as it has great impact on the expected outcome.
	Normal	The issue has a default priority.
	Low	Low priority issue, not critical. All the high and normal issues must be implemented before considering this one.
Category	Bug	Issue related to bugs (error, flaw, fault, or failure found in software).
	New feature	Issue related to a new feature (enhancement of the component).

Table 2. Set of i4Q labels proposed

2.2.5 System deployment

i4Q solutions run on different hardware specifications and therefore require different setups to work. Most i4Q Solutions are expected to be available as Docker images so they provide their own environment and tools and only a running Docker environment is required.

Docker [16] is an open-source software platform that allows the user to create, deploy and manage virtualised application containers on a common operating system and using any software component. These containers wrap a complete filesystem, which can be customised by the user like any other system. These containers are not bound to hardware, which makes relocation, testing, and later scaling of software components much easier.

Docker has many advantages to be used as containerisation technology:

- It enables to build a container image and use the same image across every step of the deployment process.
- It reduces the deployment time to seconds.
- Containers are configured to maintain all configuration and dependencies internally.
- Can be used in any operating system and in a multi-cloud platform.
- Makes sure each container has its own resources that are isolated from other containers.
- From a security point of view, Docker ensures that applications that are running on containers are completely segregated and isolated from each other, granting complete control over traffic flow and management.

Figure 2 shows the commands required to setup Docker Engine on a Linux-based system (Ubuntu 18.04 and higher versions) [17]. However, there are also Windows and Mac-based Docker environments downloadable as setups.

Install Docker Engine on Ubuntu

```
# Update apt package index:
$ sudo apt-get update

# Install packages to allow apt to use a repo over HTTPS:
$ sudo apt-get install ca-certificates curl gnupg lsb-release

# Add Docker's official GPG key:
$ sudo mkdir -p /etc/apt/keyrings
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
  sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

# Set up the repository:
$ echo "deb [arch=$(dpkg --print-architecture) \
  signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" |
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Install Docker Engine:
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli \
  containerd.io docker-compose-plugin

# Verify Docker Engine is installed correctly:
$ sudo service docker start
$ sudo docker run hello-world
```

Figure 2. Commands for installing Docker Engine on Ubuntu

In order to achieve the integration between the different solutions in the pilots, the proposed approach is to build and publish the solutions in the form of a Docker image[1]. The step-by-step process of creating this and other required artefacts can be found in **Appendix I**.

2.2.6 Software Design Quality Analysis and Evaluation

This section describes the quality analysis and evaluation topics specifically designed to evaluate the maintainability, portability, testability, usability, correctness, and robustness of the design of i4Q. According to the SWEBOK Guide [18], Software Quality Analysis and Evaluation Techniques can be classified into:

- **Software Design Review.** Informal reviews to determine the quality of the artefacts.
- **Static Analysis.** Semiformal static (non-executable) analysis of artefacts design.
- **Simulation and Prototyping.** These are dynamic techniques to evaluate a design. Some specific components provide simulation features to allow other developers to test interactions in design time to ensure that the design is correct in terms of interoperability. For instance, Data Acquisition should allow the simulation of a connection to physical devices.

[1] Exceptions may be allowed if this is not possible for some solution due to its nature (e.g. it implies hardware components), architecture, dependencies, or other reason.

In i4Q, the design of the different components is reviewed by the architecture leader every 6 months to guarantee that the definitions are consistent with the architecture design, the functional specification, and the technical specification, and putting an emphasis on its completeness, consistency, and correctness.

In addition, solution leaders have agreed on the static analysis to be carried out in case of need, as well as the simulation and prototyping activities to evaluate the design, as these will strongly depend on the component, prior to the start of any development or integration.

2.2.7 Software Construction Quality

Software Construction Quality consists of the evaluation of the quality in the software construction phase where software code is actively generated. Quality problems introduced in the construction phase could have a significant impact, such as in the case of security vulnerabilities.

Some of the techniques to evaluate the quality of code are:

- **Static Analysis Techniques.** Through software documentation and code analysis. From the range of techniques available, automatic code inspection is one of the most relevant.
- **Debugging.** Code debugging integrated into IDEs is used by developers.
- **Software Testing.** For example, unit testing and integration testing.
- **Technical Reviews.** Evaluation of the software product by a team of qualified personnel to determine its suitability.

2.2.8 Documentation

Further information about the different i4Q Solutions can be found at https://i4q.upv.es/. Details about each solution are specified on this website, such as:

- General description and solution features.
- Commercial information: authors, license and pricing.
- API specification, installation guidelines, deployment instructions, and user manual.

Moreover, several deliverables have been produced in the context of the tasks where the solutions have been implemented (see Section 2.1). These deliverables provide related information and may include an explanation of some decisions taken during the design and implementation of the solutions.

3. Solutions Integration

i4Q RIDS is a highly complex platform, which requires the integration of the different i4Q Solutions. Achieving the integration of these in the early stages of development is of crucial importance. In order to facilitate this integration, each of these solutions must meet a set of requirements to adapt to the pilot cases that make up the project.

Software development involves a multitude of different actions: choosing a programming language, validation, documentation, maintenance, updating and enhancement, among many others. This document is not intended to make each development team follow a list of steps faithfully, since each of these teams comes from a different context and follows its own steps. If the procedure described here is complicated, confusing or hinders the user experience, they will not adopt it.

Under this premise, i4Q leaves the door open to work in a variety of ways and allows development teams to choose the way that is most appropriate for their task. However, all teams are encouraged to follow some common procedures aimed at keeping a minimum degree of consistency among the solutions and facilitating the integration among them. More specifically, they all share a set of activities, interfaces and a way of producing software, all of which are described in the following subsections.

3.1 Common Structure

Every solution repository should follow as far as possible the structure described in **Figure 3.** Moreover, they should always take into account the particularities of their own development.

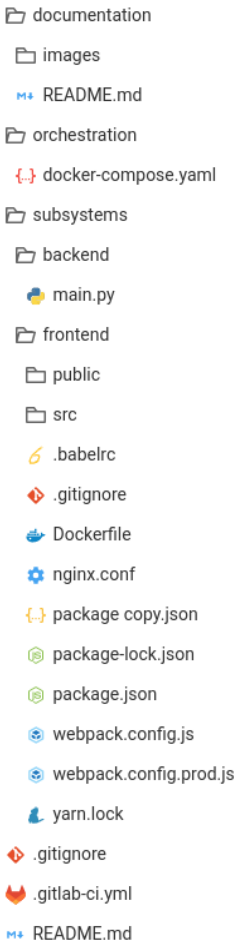

Figure 3. Common solution repository structure

From the structure shown in **Figure 3** above, the elements to be highlighted are:

- **documentation/README.md:** Contains the public documentation.
- **documentation/images:** Contains the images used in README.md.
- **orchestration/docker-compose.yml:** Contains the file that docker-compose uses to start the component in a completely unattended manner.
- **subsystems:** Contains the set of elements that make up the solution. **Figure 3** shows, as an example, two directories: *frontend* and *backend*, typical of web application development.

3.2 Solution orchestration

Each solution is composed of a set of subsystems. Each of these subsystems can be an API, a database, a graphical user interface, etc. From an integration point of view, which technology has been used in the development of these elements is not important. Therefore, the i4Q solution providers are free to choose both the paradigm and the technology that are more appropriate to solve their problem. However, whichever option is chosen, it was agreed that the result must be able to be executed in a Docker container.

The concept of orchestrating refers to the task of connecting, communicating and executing various elements, that is, the different subsystems of a solution. In this project, the goal is to make

this task as simple and automatable as possible, and above all, without requiring specific knowledge of the underlying technologies and implementations.

The following subsections show the different approaches to this orchestration, namely *monorepository*, *multirepository*, *external repository* or *mixed*, and the approach contemplated in i4Q.

3.2.1 Monorepository approach

In this approach, all source code is stored in the same project, *subsystems* directory, under element-specific directories.

This option has several benefits:

- Easy to set up a CI/CD pipeline. It is only needed to configure it in one place.
- The need to configure other tools such as package managers, image registries, external servers, etc. is avoided. The *docker-compose.yml* orchestration file has direct access to the source code of all subsystems.
- Everyone (with access to the repository) is able to review the code and understand how each part works.
- Unified management: issues, milestones, and code are placed in the same repository.

3.2.2 Multirepository approach

A different approach is to use a set of repositories. In this case and starting from the subsystems shown in **Figure 3**, we have two separate repositories, one for frontend and one for backend. These repositories, in turn, must have as output a Docker image that can be used from other repositories. Consequently, these images must be stored in a Docker registry (either GitLab's own or a different one).

Regarding the general repository containing the solution, it must explicitly use in the *docker-compose.yml* file contained in the *orchestration* directory the images stored by the other repositories.

In summary, this approach should:

- Allow the creation of as many repositories as the developer team needs, without restriction.
- Build docker images for every subsystem and store them in a Docker Registry (developer team are free to use the container registry they prefer).
- Maintain the *docker-compose.yml* in the orchestration folder, using the common structure and use explicitly the images that are saved in the Docker Registry.

3.2.3 External repository approach

In scenarios where the solution consists of a set of components and some of these contain proprietary software, it may be incompatible to store the source code in the i4Q repository. For these situations, the external repository approach is the most appropriate.

In this case, the source code is stored outside the i4Q repositories, however, the *docker-compose.yml* file, existing in the i4Q repository, must contain the necessary information to be able to run the solution (and all its components) correctly.

3.2.4 Mixed approach

As always, not everything is black and white, there are many shades in between. For those cases where the above options do not fit perfectly, this mixed approach is proposed.

In this mixed environment, we can find any combination of the aforementioned approaches, which are capable of supporting the needs of the solution. Even so, it is mandatory that a correctly configured *docker-compose.yml* file is included in order to be able to run the solution, regardless of where the source code and the Docker images are hosted.

3.2.5 i4Q approach

After analysing the different approaches and the complexity of the solutions, it has been concluded that, due to the many advantages of their use (see Section **3.2.1**), *monorepository* is the recommended approach for the i4Q project.

3.3 Solution communication

The i4Q project provides a complete suite of solutions with the option to deploy every solution either as a standalone tool or as a part of a package containing multiple solutions that synergize with each other to offer a specific set of functionalities. To achieve solution interoperability, a communication channel must be established to accommodate the exchange of information and data between them. The i4Q Message Broker is an additional i4Q component responsible to provide a fast and secure way of inter-solution communication through data streaming.

The i4Q Message Broker is based on Apache Kafka as distributed by the Confluent platform. Kafka is an open-source platform for event streaming capable of handling huge amounts of messages in real-time. It is used in all sorts of real-time data streaming applications as it is highly scalable while providing high data throughput with low latency.

The communication in Kafka is conducted through the use of Kafka topics. A topic is a storage unit that acts as an intermediary between the communication of the solutions. One or multiple solutions can produce messages in a certain topic and other solutions can later on consume these messages via subscribing to that specific topic. The data format of the messages can be either JSON or Avro which is a more efficient and compact way of exchanging messages, with a data model similar to JSON.

To ensure a secure communication between the different i4Q Solutions and the Message Broker, a TLS user authentication and data encryption based on OpenSSL is involved. These necessary digital certificates are being generated and provided to the Message Broker as well as the rest of the solutions by the i4Q Security Handler. In addition to certificates, authorization control via Access Control Lists (ACLs) has been established to restrict or allow user access to particular Message Broker functionalities, including topics.

During the present development phase of the i4Q project, the Message Broker is configured and operational at CERTH's premises. However, it is possible to deploy the Message Broker in any establishment as it is provided via Docker container in the i4Q GitLab repository.

There are two options in which each solution can interact with the Message Broker, namely:

- A **Kafka Python Client** for python native solutions.
- A **Flask based REST API** accessible from any solution.

A detailed user manual for the integration of the Message Broker in any solution can be found on the i4Q GitLab repository [20].

4. Solutions Overview

Although different development teams are responsible for the i4Q Solutions, the progress of the implementation work is measured in the same way in all cases. This section explains in more detail what aspects of the solutions development are being monitored, how this is done, and which tools are being used. More specifically, Section 4.1 focuses on tracking the completion of functional requirements. Then, Section 4.2 explains in detail the metrics considered to monitor the updates in the generated source code. Section 4.3 is devoted to the monitoring of quality of the software produced, whereas Section 4.4 is dedicated to security aspects. Finally, Section 4.5 gathers some risks that might arise during the implementation of the solutions as well as some other relevant issues.

4.1 Functional requirements status

In this section the current status of the functional requirements specified in deliverable D1.4 is analysed [21]. For this purpose, a set of tables have been defined (one for each solution) with the following characteristics:

- **ID**. Unique requirement identifier.
- **Type**. Requirement type (based on ISO/IEC/IEEE 29148).
- Title.
- **Definition**. Brief requirement overview.
- **From solution**. Indicates which i4Q Solutions this requirement is connected to.
- **Progress**. Percentage of the requirement currently completed.

Table 3 is a snippet from the internal document *i4Q_Solutions_Req_Follow-Up* [22]. More information can be found at this resource.

ID	Type	Title	Definition	From solution	Progress
BIBA1r1	Usability and Quality req	Easy to operationalize	The guideline should be easy to operationalize.	1-i4Q_DQG	0%
BIBA1r2	Guidelines req	Cover long-term and short-term measures	The guideline should cover long-term and short-term measures to improve data quality.	1-i4Q_DQG	100%
BIBA1r3	Guidelines req	Focus on information	The guideline should focus on information, not on database quality improvements.	1-i4Q_DQG	100%
BIBA1r4	Guidelines req	Use data life cycle model	The guideline should use a data life cycle model to define its scope.	1-i4Q-DQG	100%

Table 3. Snippet from i4Q_Solution_Req_Follow-Up

4.2 GitLab statistics

This section analyses some activity statistics extracted from the i4Q GitLab repository. In particular, the metrics that will be studied in the following subsections are commits, issues, jobs, and pipelines.

4.2.1 Commits

Most solutions use commits to fix or add functionality to the source code. A commit implies the addition, removal or modification of related files. For this reason, this metric can be a relevant indicator to get an overview of the activity occurring in each solution.

Since there are many solutions in this project, it is a bit complicated to create a chart that is able to graphically show the metrics of each solution. Therefore, they have been grouped by pilots. **Figure 4** shows the activity (number of commits) performed in each of the pilots.

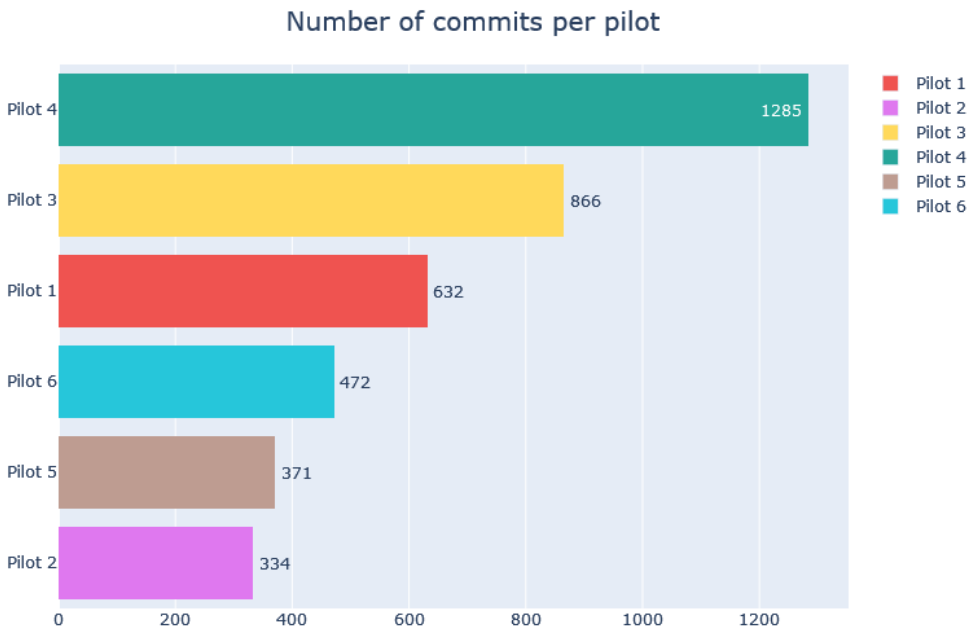

Figure 4. Number of commits per pilot

4.2.2 Issues

GitLab issues are used to report that an incident has been detected in the source code and enable tracking. These incidents can be created to:

- Fix bugs detected in the source code.
- Propose the implementation of a new feature.
- Discuss improvements to already implemented functionalities.
- Define future objectives.

In addition to the type of incident, each one may have an associated level of severity, an expiry date, people in charge, etc.

Each project can create and manage its own issues and the number of incidents depends on the level of description detail. In **Figure 5** a plot is shown with the activity of each pilot according to the number of issues.

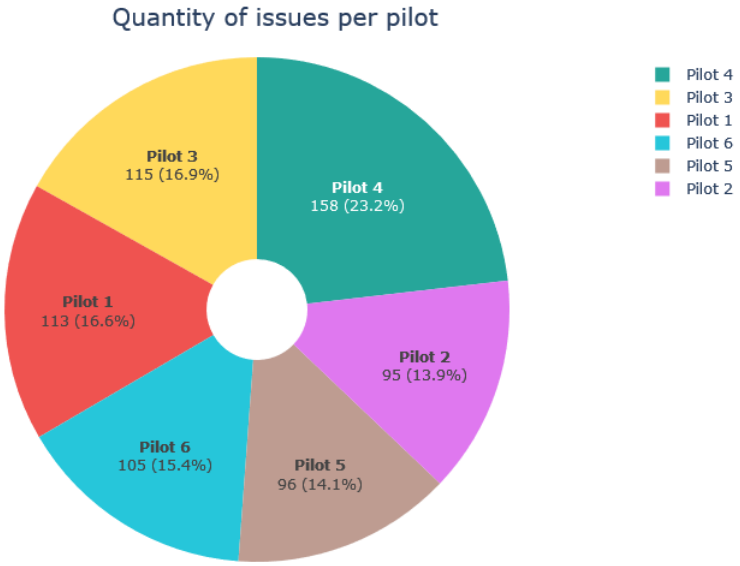

Figure 5. Quantity of issues per pilot

4.2.3 Jobs

A job is the most basic configuration component that can be run in GitLab. Known as “build step”, each one can contain multiple scripts and perform actions such as:

- Build or compile.
- Run unit tests.
- Check code quality.
- Deploy code to different environments.

Figure 6 shows the number of jobs that have been defined for each pilot. This number may vary depending on the quantity of commits performed and the number of jobs defined in each commit.

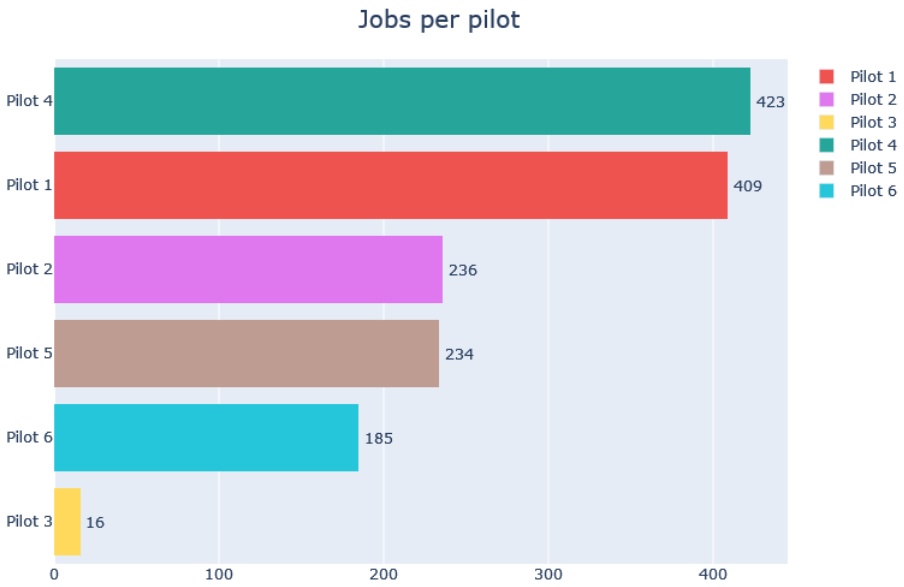

Figure 6. Amount of Jobs per pilot

4.2.4 Pipelines

According to GitLab Docs [23], pipelines are the top-level components of continuous integration, delivery and deployment (CI/CD) and are a composite of:

- **Jobs**, described in Section 4.2.3
- **Stages**. Define when to run the jobs and can include some jobs to execute or none.

If all the jobs in a stage are executed successfully, the pipeline goes to the next stage. Otherwise, if any job of a stage fails, the next stage is not run, and the pipeline execution ends with an error. This execution is usually done automatically once the pipeline is created. However, sometimes the development team may intervene manually.

Figure 7 illustrates the pipeline distribution per pilot. As with jobs, this quantity can differ with the number of commits performed and the number of pipelines defined in each one.

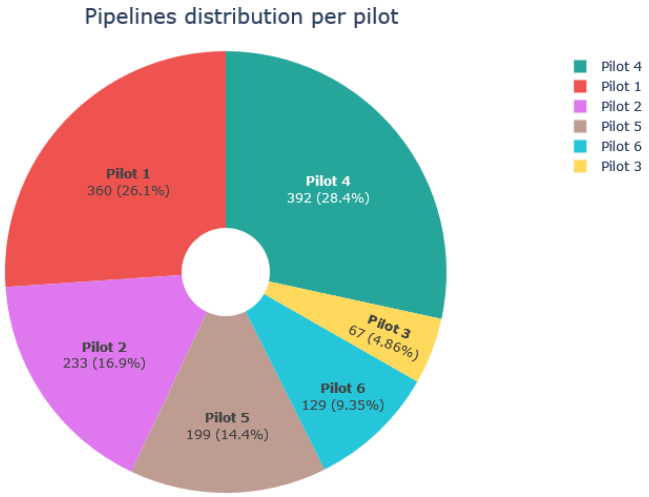

Figure 7. Pipelines distribution per pilot

4.3 Software quality

There are different methods for measuring the quality of the developed software and, depending on the method selected, different tools can be used. In this project it has been agreed to use SonarQube [24], which is an open-source tool that allows to review code automatically and detect bugs, vulnerabilities and code smells. In **Figure 8** an image of the main project panel is shown.

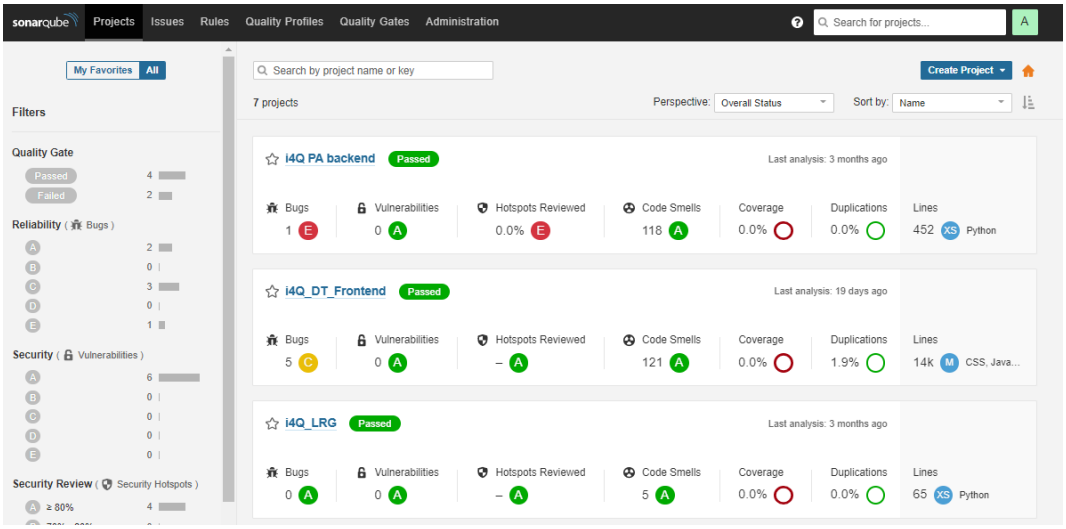

Figure 8. i4Q project SonarQube main panel

In order to achieve a good integration by phases with SonarQube, a CI/CD pipeline has been defined in the GitLab project repository.

The integration of SonarQube in each of the solutions requires that each partner defines a new stage in the *.gitlab-ci.yml* file. Here the configuration that allows to analyse the code is stored and sends the results to a central instance of SonarQube, which is deployed in a UPV server and is accessible to all the solution providers.

Task T9.4 will be responsible for analysing the code of the solutions periodically and report the results to the corresponding solution providers to introduce the appropriate modifications.

4.4 Security

As with quality, depending on the method selected to measure the security of the developed software, one tool or another may be employed. For this project it has been agreed to use Dependency Track [25], because it is an open-source platform that allows easy integration with the GitLab CI/CD engine.

This tool is capable of performing a detailed analysis of the different software components and detecting vulnerabilities. Once the analysis phase is completed, it generates a report with some graphs indicating the risk level of the vulnerabilities, a description of the problems, and proposes some corrective measures to solve these problems. **Figure 9** shows the charts generated by Dependency Track after analysing different components of a software solution.

Figure 9. Dependency Track interface example

As with SonarQube, a CI/CD pipeline has been defined in the project repository to enable the integration with Dependency Track. This integration requires that each partner defines a new stage in the *.gitlab-ci.yml* file. This stage contains the configuration to analyse the code and send the results to a central instance of Dependency Track, which is deployed on a UPV server and is accessible to all the solution providers.

The code of the solutions will be periodically analysed by task T9.4, which will also be responsible for reporting the results to the respective solution providers in order to introduce the required changes.

4.5 Risks or Issues

According to deliverable D9.10 [26], the main risks and issues in WP6 are those indicated in the following table.

Id	Description	Impact	Probability	Mitigation measures
R6-1	Communication problems between pilot factories due to the protocols used. This may result in not being able to easily obtain data or perform actions on the different devices.	Medium	Medium	Inform in advance about the communication protocols that may be used by pilots in their facilities. Regarding the ingestion process, it is proposed to solve this lack of data with other sensors that support the same protocols as the other solutions.
R6-2	If the number of incidents is not very large, or failures do not occur for a long time, it will be difficult to train a model that can make decisions based on this data.	Medium	Medium	Use a dataset to train the model in which a higher number of incidents are recorded or with a shorter period of time between failures.
R6-3	An incorrect estimation of the effort required or the level of complexity involved in integrating the different solutions may lead to delays in the process.	Medium	Medium	Put more effort into evaluating the complexity of each task, as well as planning and designing the integration tasks and the responsibilities of each member involved in the work package.
R6-4	Noise in the training data set labelling for the injection process.	Medium	Medium	Implementation of a laser barcode system to minimise the risk.

Table 4. Risks and issues in WP6

5. Integration and validation phase overview

After M24, once the i4Q Solutions are already implemented, the project will start the integration and validation phase. That is, the pipelines defined for each pilot use case (see deliverables from D6.1 to D6.7) will be implemented in the real scenario of the corresponding industrial partners business processes. This means that i4Q Solutions will be integrated among themselves, and with other elements of the industrial partner factory.

The implementation of the above-mentioned pipelines will contribute to validate the i4Q Solutions since it will allow solution providers to check whether their solutions completely fulfill the requirements of each pilot. Consequently, it may be necessary to modify the implementation of some i4Q Solutions. For instance, to provide some uncovered requirement, to adapt the solution to a pilot's specific characteristic, or bug.

The integration and validation of i4Q Solutions will be split into two iterations: the first one from M25 to M30, and the second one from M31 to M36. Results of each iteration will be presented in the next versions of this deliverable, that is D6.17 and D6.18, due at the end of M30 and M36, respectively.

In the following we describe the steps that will be followed in each iteration:

1. Industrial partners prepare the technological infrastructure where the i4Q Solutions must be deployed. There are several options for this step. First, they can deploy all the solutions on-premise, that is, in their own technological infrastructure. Another option is deploying the solutions on an external cloud platform. As explained above, most of the solutions have been implemented to be deployed as Docker containers and, thus, are compatible with almost any cloud service provider. The third option is to follow a hybrid approach, so that certain solutions are deployed on-premise, and others on an external cloud.

   However, this step may be different for some specific solutions. For instance, the i4Q^{TN} (involved in the generic pilot's pipeline) contains parts of hardware that need to be physically installed.

2. Solution providers together with the industrial partners deploy the latest version of the solutions in the infrastructure prepared in step 1.
3. Solution providers focus on the integration of the i4Q Solutions following the pipelines defined for each pilot and making use of the Message Broker. This step might require additional integrations with other components of the industrial partners.
4. Then, the focus will be the testing and validation of the involved i4Q Solutions in the context of the pilot's business process. During this phase industrial partners and solution providers have to monitor especially the aspects involved in the KPIs defined for each pilot. The results of this work must be appropriately gathered to be used as input later on.
5. The execution of step 4 may imply the identification of bugs, uncovered requirements, or minor changes that might be considered in the solutions. If this is the case, solution providers will have to gather these updates and address them to produce a new version of their solutions. The fixing of bugs can be performed in parallel to step 4.
6. Finally, solution providers and industrial partners will analyze the results obtained in step 4, possibly in parallel to step 5, and compare them with the KPIs defined for each pilot.

The outcome of this analysis, among other possibilities, can be used as input to step 5 to improve the solutions, or the industrial partners, to improve their business process.

The initial plan is to dedicate three months to perform steps 1-5, and three months to step 6. However, the efforts and time dedicated to performing each step may vary from one iteration to another. This is the case, for instance, of step 1. In the first iteration industrial partners may start from scratch to prepare the infrastructure. However, in the second iteration it may consist only of minor actions to update the infrastructure prepared in the first iteration, or even no related work at all. Similarly, steps 2 and 3 in the second iteration may just require updating the solution's version but no other significant changes.

6. Conclusions

The procedures to manage the work of the different project partners, and the roles and responsibilities of each partner have been properly defined since the beginning of the project. However, i4Q is a project with some particularities which make putting all this in practice quite challenging, especially:

- The fact that it involves large groups of solution providers and use cases (Pilots) coming from very different institutions, and geographically spread over Europe.
- It requires developing and integrating a considerable number of existing software pieces, quite heterogeneous in terms of functionalities.

This document gathers all the actions applied so far in the project to address and overcome these challenges. For instance, the procedures followed to manage and coordinate the work of the project partners, the methodologies applied for different aspects of the developments, and the tools used in each case.

As stated in this deliverable, i4Q is making use of well-known state-of-the-art tools (such as GitLab and SonarQube) which will enforce the procedures established and, of course, the use of these tools will help solution providers during the Integration and validation phase.

Of course, all the procedures described in this deliverable are subject to improvement. So far, i4Q has focussed on a first version of the Solutions and a Demo of the Pilots and the final version of all i4Q Solutions is being released at the same time as the submission of this document (M24). Since the integration and validation of the i4Q Solutions in the context of the seven pilot use cases defined in this project will start in M25 (following the procedure detailed in Section 5, the procedures described here might be updated during 2023. Furthermore, new versions of the i4Q Solutions might be released in the context of T6.8 to include necessary functionalities not covered before or to fix problems identified during the integration and validation phase.

All these updates will be gathered and explained in the next versions of this deliverable, which will be submitted in M30 and M36, respectively.

References

[1] "Grant Agreement number: 958205 – Industrial Data Services for Quality Control in Smart Manufacturing (i4Q)," European Health and Digital Executive Agency (HADEA), Jul. 2022. Accessed: Oct. 3, 2022. [Online]. Available: https://bit.ly/3MJvXcX

[2] K. Schwaber and J. Sutherland, "The 2020 Scrum Guide," *Scrumguides.org.* https://scrumguides.org/scrum-guide.html (accessed Aug. 10, 2022).

[3] "The One DevOps Platform," *GitLab.com*. https://about.gitlab.com/ (accessed Aug. 10, 2022).

[4] "Apache Subversion," *Apache.org.* https://subversion.apache.org/ (accessed Aug. 10, 2022).

[5] "CVS — Concurrent Versions System v1.11.23," *GNU.org.* https://www.gnu.org/software/trans-coord/manual/cvs/cvs.html (accessed Aug. 10, 2022).

[6] "Mercurial SCM," *mercurial-scm.org.* https://www.mercurial-scm.org/ (accessed Aug. 10, 2022).

[7] T. Günther, "8 Reasons for Switching to Git," *git-tower.com*, Jul. 2020. https://www.git-tower.com/blog/posts/8-reasons-for-switching-to-git (accessed Aug. 10, 2022).

[8] B. Morelli, "New Developer? You should've learned Git yesterday.," *codeburst.io*, Jul. 5, 2017. https://bit.ly/3EVrxgP (accessed Aug. 10, 2022).

[9] H. Bouasavanh, "4 Reasons Why Beginning Programmers Should Use 'Git'," *Medium*, Jan. 24, 2018. https://bit.ly/3XvYhVe (accessed Aug. 10, 2022).

[10] "i4Q project GitLab repository homepage," *GitLab.com.* https://gitlab.com/i4q

[11] "Jenkins Handbook," *Jenkins.io.* https://www.jenkins.io/doc/book/ (accessed Sep. 19, 2022).

[12] "GitLab CI/CD," *docs.gitlab.com.* https://docs.gitlab.com/ee/ci/ (accessed Sep. 19, 2022).

[13] S. Pittet, "Continuous integration vs. delivery vs. deployment," *Atlassian.com.* https://bit.ly/2tkhwnD (accessed Dec. 14, 2022).

[14] M. Courtemanche, "What is continuous deployment?," *TechTarget.com.* https://www.techtarget.com/searchitoperations/definition/continuous-deployment (accessed Dec. 14, 2022).

[15] "GitLab Issues," *docs.gitlab.com.* https://docs.gitlab.com/ee/user/project/issues/ (accessed Oct. 10, 2022).

[16] "What is Docker and How Does It Work?," *TechTarget.com.* https://www.techtarget.com/searchitoperations/definition/Docker (accessed Dec. 05, 2022).

[17] "Install Docker Engine on Ubuntu," *Docker Documentation.* https://docs.docker.com/engine/install/ubuntu/ (accessed Oct. 19, 2022).

[18] "ISO/IEC TR 19759:2005 – Guide to the Software Engineering Body of Knowledge (SWEBOK)," *ISO.org,* Sep. 2005. https://www.iso.org/standard/33897.html (accessed Sep. 26, 2022).

[19] "i4Q Reliable Industrial Data Services (i4Q RIDS)," *i4Q Solutions.* https://i4q.upv.es/ (accessed Nov. 3, 2022).

[20] "i4Q Message Broker GitLab Repository," *GitLab.com.* https://gitlab.com/i4q/message-broker (accessed Nov. 16, 2022).

[21] "D1.4 – Requirements Analysis and Functional Specification," Technische Universität Berlin (TUB), Apr. 2021. Accessed: Oct. 24, 2022.

[22] "i4Q_Solutions_Req_Follow-Up," Nov. 2022.

[23] "GitLab CI/CD pipelines," *docs.gitlab.com.* https://docs.gitlab.com/ee/ci/pipelines/ (accessed Oct. 28, 2022).

[24] "Code Quality and Code Security", *Sonarqube.org.* https://www.sonarqube.org/ (accessed Nov. 17, 2022).

[25] "Dependency-Track | Software Bill of Materials (SBOM) Analysis," *dependencytrack.org.* https://dependencytrack.org/ (accessed Nov. 18, 2022).

[26] I. Gialampoukidis and S. Vrochidis, "D9.10 – Short Interim Management Report v3", Centre for Research & Technology Hellas (CERTH), Nov. 2022.

Appendix I

This appendix is intended to address the following questions:

- How to enable the CI/CD option in the repository configuration.
- How partners can build Docker images of their solutions in an automated way using the .gitlab-ci.yml configuration file.

Enable CI/CD option in the GitLab repository settings

First of all, the repository needs to be configured to enable the CI/CD section. To do this, activate the CI/CD checkbox found in Settings/General/Visibility, project features, permissions. After this, a new CI/CD section will appear in Settings. Here, you can set the path to the .gitlab-ci.yml file in case is not in the default path: <root directory>/.gitlab-ci.yml.

Configuration of .gitlab-ci.yml file to generate Docker images automatically

The next step is to edit the .gitlab-ci.yml file to include the Docker image creation and storage tasks. To do this, the lines of code shown in **Figure 10** must be added.

```
# This is a GitLab CI configuration to build the project as a Docker image.
# Do not use "latest" here, if you want this to work in the future.
image: docker:20

# Use this if yout GitLab runner does not use socket binding
services:
  - docker:20-dind
variables:
    DOCKER_TLS_CERTDIR: ""

stages:
  - build docker image
  - push docker image
```

Figure 10. First lines of the .gitlab-ci.yml file

The lines of code shown above allow to use GitLab's Docker service to generate images. In addition, it defines two scenarios: build docker image and push docker image.

For each of these scenarios a template is provided which will need to be adapted to the characteristics of each partner's software solution.

Build Docker Image

In **Figure 11** a sample configuration of how to build a Docker image is depicted.

```
build develop:
  stage: build docker image
  only:
    - develop
  script:
    # docker login asks for the password to be passed through stdin for security.
    # We use $CI_JOB_TOKEN here which is a special token provided by GitLab.
    - echo -n $CI_JOB_TOKEN | docker login -u gitlab-ci-token --password-stdin $CI_REGISTRY
    # fetches the latest image (not failing if image is not found)
    - docker pull $CI_REGISTRY_IMAGE:develop || true
    # builds the project, passing vcs vars for LABEL notice the cache-from,
    # which is going to use the image we just pulled locally
    # the built image is tagged locally with the commit SHA, and then pushed to
    # the GitLab registry
    - >
      docker build
      --pull
      --build-arg VCS_REF=$CI_COMMIT_SHA
      --build-arg VCS_URL=$CI_PROJECT_URL
      --cache-from $CI_REGISTRY_IMAGE:develop
      --tag $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
      --target development
      .
    - docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
```

Figure 11. Sample code for building a Docker image

Every time a push is done to the develop branch, this configuration will be executed. First, the configuration will build a Docker image with the tag corresponding to the commit hash code and then will publish it to the image registry.

Push Docker Image

The second scenario is the publication of the image in the registry with a more distinctive label. A sample configuration can be seen in **Figure 12**.

```
push develop:
  stage: push docker image
  only:
    # Only "develop" should be tagged "develop"
    - develop
  variables:
    # We are just playing with Docker here.
    # We do not need GitLab to clone the source code.
    GIT_STRATEGY: none
  script:
    # docker login asks for the password to be passed through stdin for security.
    # We use $CI_JOB_TOKEN here which is a special token provided by GitLab.
    - echo -n $CI_JOB_TOKEN | docker login -u gitlab-ci-token --password-stdin $CI_REGISTRY
    # Because we have no guarantee that this job will be picked up by the same runner
    # that built the image in the previous step, we pull it again locally.
    - docker pull $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
    # Then we tag it "latest"
    - docker tag $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA $CI_REGISTRY_IMAGE:develop
    # And we push it.
    - docker push $CI_REGISTRY_IMAGE:develop
```

Figure 12. Sample code for image publication in the repository registry

This job runs after the Docker image build and retrieves the image tagged in the previous step and adds a new tag to it (in this case develop). After this, the new image is added to the registry.

Once the explanation in parts has been finished, the complete code of the .gitlab-ci.yml file can be found in **Figure 13**.

```
# This is a GitLab CI configuration to build the project as a Docker image.
# Do not use "latest" here, if you want this to work in the future.
image: docker:20

# Use this if your GitLab runner does not use socket binding
services:
  - docker:20-dind
variables:
  DOCKER_TLS_CERTDIR: ""

stages:
  - build docker image
  - push docker image

build develop:
  stage: build docker image
  only:
    - develop
  script:
    # docker login asks for the password to be passed through stdin for security.
    # We use $CI_JOB_TOKEN here which is a special token provided by GitLab.
    - echo -n $CI_JOB_TOKEN | docker login -u gitlab-ci-token --password-stdin $CI_REGISTRY
    # fetches the latest image (not failing if image is not found)
    - docker pull $CI_REGISTRY_IMAGE:develop || true
    # builds the project, passing vcs vars for LABEL notice the cache-from,
    # which is going to use the image we just pulled locally
    # the built image is tagged locally with the commit SHA, and then pushed to
    # the GitLab registry
    - >
      docker build
      --pull
      --build-arg VCS_REF=$CI_COMMIT_SHA
      --build-arg VCS_URL=$CI_PROJECT_URL
      --cache-from $CI_REGISTRY_IMAGE:develop
      --tag $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
      --target development
      .
    - docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA

push develop:
  stage: push docker image
  only:
    # Only "develop" should be tagged "develop"
    - develop
```

```
variables:
  # We are just playing with Docker here.
  # We do not need GitLab to clone the source code.
  GIT_STRATEGY: none
script:
  # docker login asks for the password to be passed through stdin for security.
  # We use $CI_JOB_TOKEN here which is a special token provided by GitLab.
  - echo -n $CI_JOB_TOKEN | docker login -u gitlab-ci-token --password-stdin $CI_REGISTRY
  # Because we have no guarantee that this job will be picked up by the same runner
  # that built the image in the previous step, we pull it again locally.
  - docker pull $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
  # Then we tag it "latest"
  - docker tag $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA $CI_REGISTRY_IMAGE:develop
# And we push it.
- docker push $CI_REGISTRY_IMAGE:develop
```

Figure 13. Full code from .gitlab-ci.yml file[i]

[i]