Title:	Document Version:
D4.1 WATERMED Benchmarking and Testing Environment	5.0

Project Number:	Project Acronym:	Project Title:
1821	WATERMED 4.0	Efficient use and management of conventional and non-conventional water resources through smart technologies applied to improve the quality and safety of Mediterranean agriculture in semi-arid areas

Contractual Delivery Date:	Actual Delivery Date:	Deliverable Type* - Security**:
31/05/2020	31/05/2020	R – PU

* Type: P – Prototype, R – Report, D – Demonstrator, O – Other
** Security Class: PU- Public, PP – Restricted to other programme participants (including the Commission), RE – Restricted to a group defined by the consortium (including the Commission), CO – Confidential, only for members of the consortium (including the Commission)

Responsible and Editor/Author:	Organization:	Contributing WP:
Antonio Skarmeta	Universidad de Murcia	WP4

Authors (organizations):

Antonio Skarmeta (UMU)

Abstract:

This document describes the work carried out so far in the scope of Task 4.1 Watermed Benchmarking and Testing Environment. Being more specifically, this deliverable provides a first version of Benchmarking and Testing Environment for Watermed 4.0. For this purpose, we have covered different phases, ranging from the methodology for testing functional and non-functional aspects, the System Development Life Cycle, which will be managed follow an agile methodology such as SCRUM, as well as the definition of an evaluation framework based on ISO/IEC 25010. Additionally, we describe the deployment process of the Watermed components using the GitLab framework and the provided pipelines for CI/CD for testing, integrating and deploying these components. Finally, we have covered the evaluation process and the test methodology too.

Keywords:

Benchmarking, Testing Environment, KPIs, GitLab, CI/CD

Revision History

The following table describes the main changes done in the document since created.

Revision	Date	Description	Author (Organization)
v1.0	21/04/2020	First version of the document	Antonio Skarmeta (UMU)
v2.0	04/05/2020	Contribution to Sections 5 and 6	Abou El Hassen Benyamina (LAPECI)
v3.0	06/05/2020	Edition of the document. Abstract, executive summary	Antonio Skarmeta (UMU)
v4.0	15/05/2020	Quality control revision	Abou El Hassen Benyamina (LAPECI)
v5.0	27/05/2020	Final document edition	Antonio Skarmeta (UMU)

Executive Summary

This document describes a first version of Benchmarking and Testing Environment for Watermed 4.0 as part of the work carried out in the scope of Task 4.1 Watermed Benchmarking and Testing Environment.

Although this task continues almost up the end of this project, we have progressed by defining a methodology for testing functional and non-functional aspects selecting a pragmatic perspective and selecting the most appropriate testing activities. Additionally, we also comment in this document the System Development Life Cycle, which will be managed follow an agile methodology such as SCRUM. We have specified an evaluation framework which is based on ISO/IEC 25010 which describes both sets of internal and external measures.

On the other hand, we describe the deployment process of the Watermed components using the GitLab framework which allow us to define different activities for building each of them. Likewise, the benchmarking and testing environment makes use of the different pipelines provided by Gitlab for CI/CD for testing, integrating and deploying these components.

Finally, we also address in this document the evaluation process and the test methodology that will be also performed over the Watermed architecture components.

Disclaimer

This project has received funding from the Partnership for Research and Innovation in the Mediterranean Area Programme (PRIMA) as part of the European Union's Horizon 2020 research and innovation programme under grant agreement No 1821, but this document only reflects the consortium's view. PRIMA or the European Commission is not responsible for any use that may be made of the information it contains.

Table of Contents

TABLE OF FIGURES

1. METHODOLOGY

There exist different approaches for software testing that cover both functional and non-functional aspects. Actually, Figure 1, presents the most relevant approaches for both categories [1] [2]. Although the adoption of all the approaches would be a desirable goal to have a complete set of tests to be carried out in our framework, we have adopted a more pragmatic approach by selecting a subset of them for the use case test scope. They are marked in red in the figure.

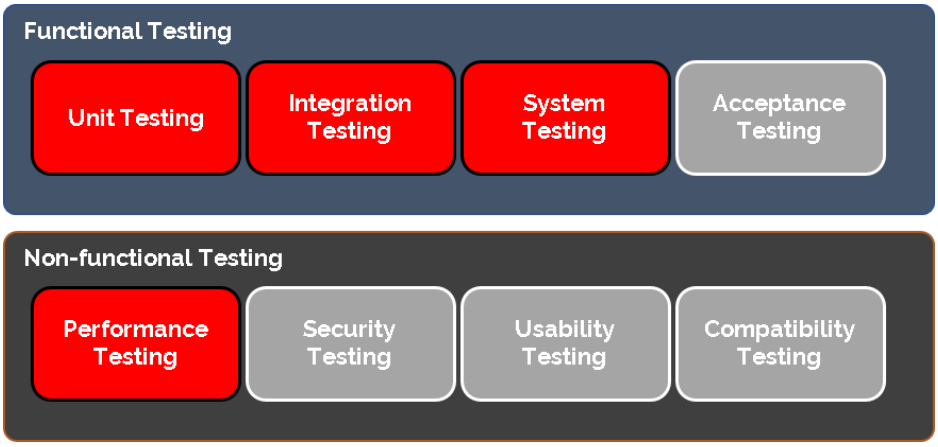

Figure 1: Testing methodologies selected for the Watermed scope

From functional testing methodologies perspective, we skipped acceptance testing as Watermed is a low Technology Readiness Level (TRL) system and does not have business stakeholders that could help to provide full business-oriented insight.

As for non-functional testing, Watermed will not perform security testing of their components as they are on different stages of development and some project partners might not have the resources to conduct full penetration testing of their modules. Usability testing will look at each component from the user perspective; however, in this scenario, most of the Watermed components are running in the background, and only the User Interface (UI) part of Watermed can be evaluated. We believe that more testing in this space can be completed at the end of the project when all functional, integration and system testing will be already completed. From compatibility viewpoint, at this stage, Watermed encourages usage of Dockerised components, and Docker [3] container technology will provide maximum coverage in this space.

Let us present the selected methodologies for the Watermed use case test scope.

- Unit Testing - will be used to validate functionality at the individual level. Each project partner will be required to provide as much coverage for required functionalities in their components at their level. This methodology will utilise best practices used in every modern programming language. Section 4 defines each core component where these unit tests must be performed.

- Integration Testing - the main scope of this document will be around defining test cases for integration between components and to support the integration via APIs, interfaces and common means of communication.

- System Testing - at this level, the Watermed architecture or parts of it will be tested and evaluated over specific test scenarios that will be defined under Task 6.3.

- Performance Testing - across all test cases a selection of KPIs will be measured to check against test cases definition of the acceptable level of KPIs established for each instance separately.

The Watermed architecture will consist of existing, adapted and new software components and libraries. Multiple stakeholders are contributing their work, which will be rolled out on different use-case and pilots. This will easily lead into complex conditions and requirements to gain deployable and reusable results. Furthermore, the final build of the framework needs to be maintainable to provide a sustainable utilization. Such software development, operation and maintenance challenges can be represented in the System Development Life Cycle (SDLC) methodology1. For the Watermed architecture the SDLC methodology phase itself is organized in SCRUM methodology, which is extending standard SDLC models towards an agile SDLC model.

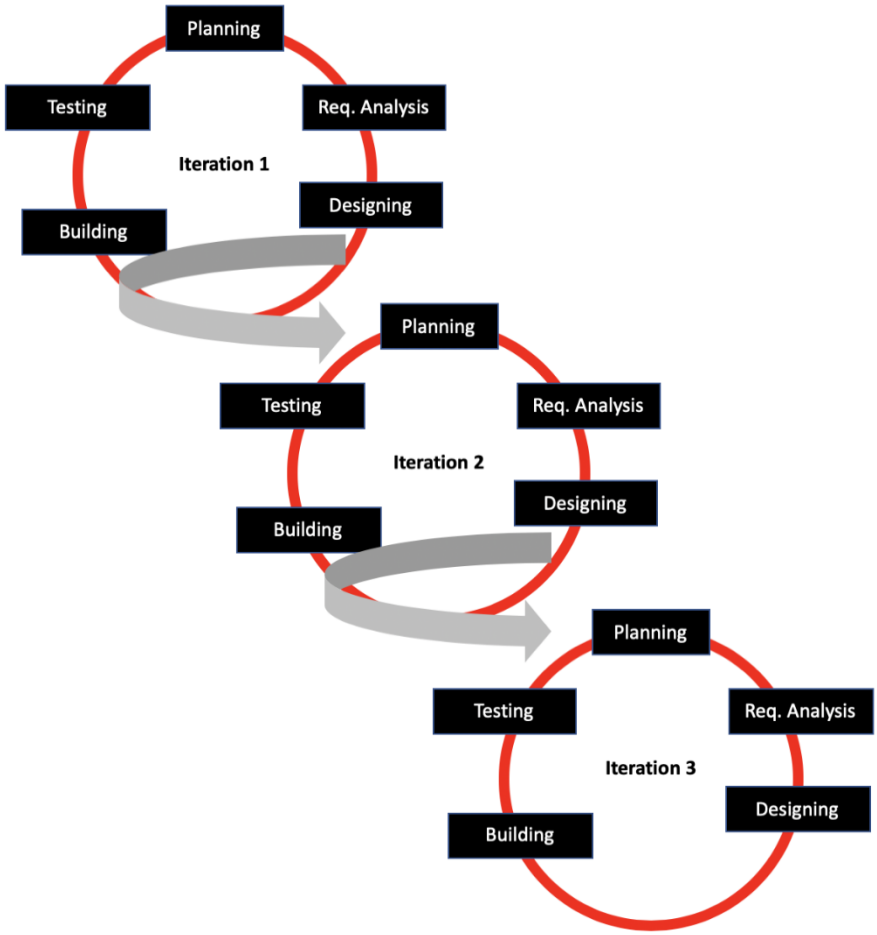

Figure 2: Agile SDLC methodologies selected for the Watermed scope

[1] *S.Barjtya, A.Sharma, U.Rani, 2017, A detailed study of Software Development Life Cycle (SDLC) Models, International Journal Of Engineering And Computer Science ISSN:2319-7242*

This agile SDLC methodology is a combination of iterative and incremental process models. It reflects the needs of adaptability for Watermed platform components. Besides, it allows to break the complex framework into incremental builds. In Watermed the development is a co-working process between international distributed teams. Building the framework in agile SDLC iterations, supports such teams to work simultaneous on their specific tasks and review the results with all project stakeholders at the end of an iteration. The framework functionality (and complexity) will increase step by step on incremental iterations towards the final build, which will include all features required by the use case pilots.

For Watermed, the phases defined in the SDLC can be linked to certain work packages and stakeholders. Before each iteration in the agile SDLC model the tasks on single stage are validated with the conditions of previous work (documented in deliverables), new conditions will be defined (results from new deliverables) or conditions will be changed (as a result from development and testing). The software contributions will be continuously documented in GitLab2 as collaborative code version management system.

Overview on agile SDLC cycle mapped to the Watermed development:

- Stage 1: Planning and Requirement Analysis

  Tasks are defining the Framework baseline requirements. This information is used to describe Watermed basic approach and conditions to fulfil the technical, legal and economic needs.

- Stage 2: Defining Requirements

  Tasks are defined by the results of Stage 1. In this stage the overall requirement analysis is used to build the specific product requirements for the Watermed architecture. Results of Stage 2 are building the Software Requirement Specification (SRS).

- Stage 3: Designing the Product Architecture

  Based on the SRS of Stage 2 the product architecture will be defined and documented in specific Design Document Specifications (DDS). This DDS will be reviewed during iteration cycles by all product stakeholders (developers and pilots) towards risks, robustness, modularity and the project timeline.

- Stage 4: Developing the Product

  In this stage of SDLC the Watermed architecture is built by the project developers. Programming code is adopted, created and documented by GitLab. As the Watermed architecture consist of code contributions from multiple partners, the project did not limit contributions on a specific coding language. To avoid friction losses the partners, agree on common methods for interoperability on modules (API's).

[2] *https://about.GitLab.com/*

- Stage 5: Testing

  Testing is a subset of all stages in modern SDLC models and therefore for Watermed the testing is mostly involved on stages 3 (Design evaluation), stage 4 (quality management in development e.g. by unit tests) and stage 6 (feedback from pilots).

- Stage 6: Deployment and Maintenance

  Once Watermed modules and components are tested, they will be deployed for usage in the framework and for specific pilots. This will be done via GitLab in defined iterations. The framework baseline components will be released finally on public Gitlab, when they have reached a sufficient maturity based on the DDS and testing results.

  In addition to these stages, the Watermed modules and components integrated into GitLab must be maintained during the operation of the platform in order to continuously ensure the functional needs of the different specifications defined previously and also to take care of the perfections required by the end users during Watermed's operations.

2. EVALUATION FRAMEWORK

For the evaluation of the quality of the developed platform, we chose a well-known standard as a basis, namely the ISO/IEC 25010:2011 "Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models" [4]. In more detail, the ISO/IEC 25010:2011 defines as stated on its official website:

- A quality in use model (in our case Actual Usage Evaluation) composed of five characteristics (some of which are further subdivided into sub-characteristics) that relate to the outcome of interaction when a product is used in a particular context. This system model is applicable to the complete human-computer system, including both computer systems in use and software products in use.
- A product quality model (in our case Technical Evaluation) composed of eight characteristics (which are further subdivided into sub-characteristics) that relate to static properties of software and dynamic properties of the computer system. The model is applicable to both computer systems and software products.

The technical assessment of the Watermed architecture will consider not only the software elements but also the perceived usefulness and appropriateness. Additionally, the technical validation on Watermed can be conducted using a subset of the product quality model of the ISO 25010 and based on the defined KPIs.

2.1 The Product Quality Model

The product quality model describes the internal and external measures of software quality. Internal measures describe a set of static internal attributes that can be measured. The external measures focus more on software as a black box and describe external attributes that can be measured.

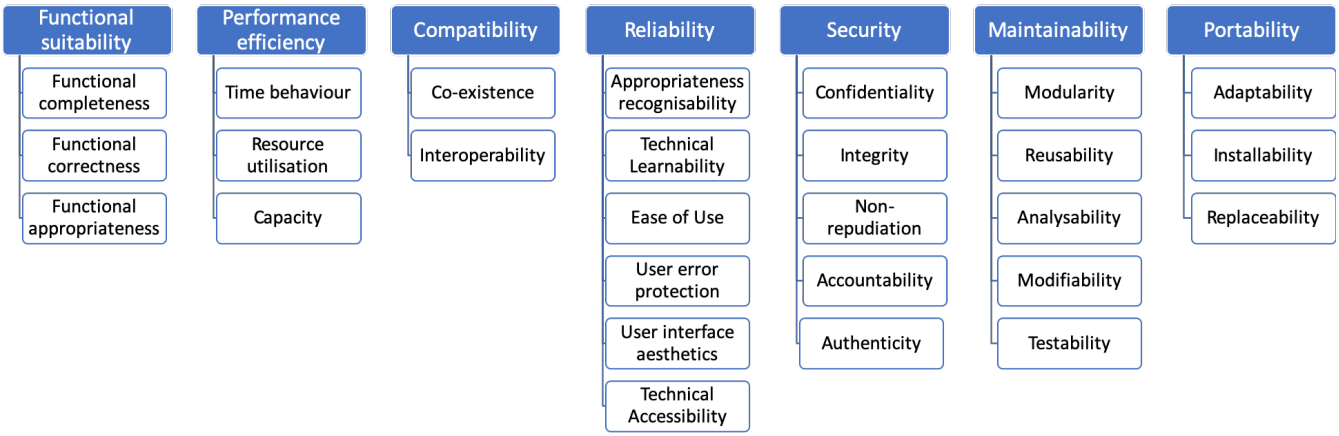

Figure 3: A product quality model view based on the ISO/IEC 25010:2011 standard

Generally speaking, this model evaluates software quality using a structured set of characteristics (each of them includes other sub-characteristics), which are the following:

1. Functional suitability - The degree to which the product provides functions that meet stated and implied needs when the product is used under specified conditions.

2. Performance efficiency - The performance relative to the number of resources used under stated conditions.

3. Compatibility - The degree to which two or more systems or components can exchange information and perform their required functions while sharing the same hardware or software environment.

4. Operability - The degree to which the product has attributes that enable it to be understood, learned, used and attractive to the user, when used under specified conditions.

5. Reliability - The rate to which a system or component performs a series of functions under parametrized conditions for a specified period.

6. Security - The degree of protection of information so that only legitimate persons or systems can have access to the data, denying it to non-authorised ones.

7. Maintainability - The degree of effectiveness and efficiency with which the product can be modified.

8. Portability - The degree to which a system or component can be effectively and efficiently transferred from one hardware, software or other operational or usage environment to another.

Table 1 showcases the sub-characteristics of each category and indicates their relativity to the Watermed architecture.

Table 1: Technical Characteristics and Sub-characteristics relevant to Watermed technical validation

Sub-characteristics	Definition
Functional suitability	
Functional completeness	Degree to which the set of functions covers all the specified tasks and user objectives.
Functional correctness	System provides the correct results with the needed degree of precision.
Performance efficiency	
Time behaviour	Response, processing times and throughput rates of a system, when performing its functions, meet requirements.
Resource utilisation	The amounts and types of resources used by a system, when performing its functions, meet requirements.
Reliability	
Maturity	Under normal operation system meets requirements for reliability.
Availability	System is operational and accessible when required for use.
Fault tolerance	System operates as intended despite the presence of hardware or software faults.
Recoverability	System can recover data affected and re-establish the desired state of the system is case of an interruption or a failure.

2.2 Watermed architecture: Product Quality Evaluation Framework

Table 2 presents which criteria could be measured during the Watermed architecture operation. Since the ISO/IEC 25010:2011 standard does not define specific attributes (measuring wise) for each one of the sub-characteristics, we have devised the following list for allowing the technical assessment of the Watermed architecture. It is worth highlighting that due to the nature of the project and based on the operation conditions of the pilots, we have considered some of the below-mentioned as optional, as their measurement might not be possible or do not produce meaningful results.

In addition, other measures or metrics can be selected or defined, in particular those related to architectural properties like coupling and cohesion. As example of metrics, we can cite: Coupling Between Object (CBO), Message Passing Coupling (MPC), Response For a Class (RFC), and Lack of Cohesion in Methods (LCOM). The purpose is to study and assess the quality of the code of the various components (programs) developed within the Watermed platform. For this, we can use existing tools or develop new tools dedicated to extracting and calculating these metrics.

Table 2: Quantitative Evaluation Metrics selected for the Watermed architecture

Sub-characteristics	KPIs	Calculation Type
Functional completeness	Portion of covered functional requirements	(Completed high priority functional requirements / Total Number of high priority requirements) * 100 %
Functional correctness	Portion of functional requirements covered without reported bugs, after tests	(Completed functional requirements of high priority without bugs / Total Number of high priority requirements) * 100 %
Time behaviour	Average Latency	(Total Response Time) / (No. of Requests)
	Throughput	(Total No. of Kilobytes) / (Total Time of Operation)
Resource utilisation	Mean % CPU Utilisation	(Σ (% CPU utilisation probes)) / (No. of probes)
	Max. Memory Used	No. of max Megabytes of RAM Memory recorded
	Max. Processing Power Used	Max % CPU utilisation recorded
Maturity	Max. Concurrent Users Supported	No. of Max. Concurrent Users Recorded
	Simultaneous Requests	No. of Simultaneous Requests
Availability	% Monthly Availability	1- ((Downtown Time Minutes) / (Month Days*24*60))
	Error Rate	(No. of Problematic Requests) / (Total Number of Requests)
Fault tolerance	Number of Software problems identified without affecting the platform	No. of Non-Critical Software Errors
Recoverability	Mean time to recover from software problems	(Total Recovering Time due to Software Issues) / (Total Software Issues resulting to recovery)

3. DEPLOYMENT PROCESS

3.1 Watermed Core Components Deployment

In general, each component has an individual deployment process, which is depending on the specific development history and status. Components shall be deployed in virtual containers (e.g. Docker), while existing or adopted containers will be deployed on instances (e.g. virtual machines or virtual containers). Each core component is developed or adopted, tested and published on own quality management cycles.

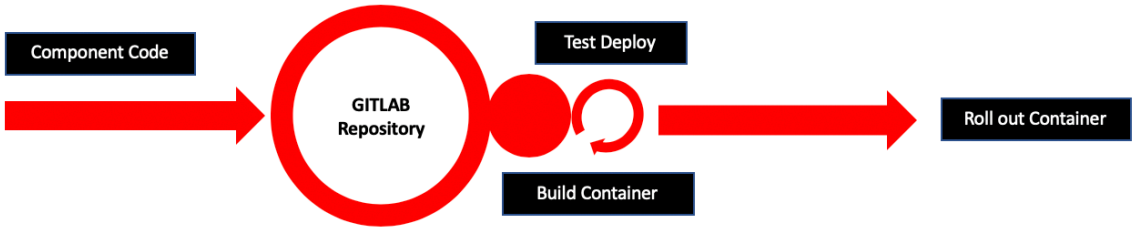

Figure 4: Principle of deploy management of a single component / enabler

GitLab is the central repository to keep the sources and developer information on core components (enablers) The component's product owner will decide, if a specific development version is ready to go for a release in a container version. The testing procedures and quality management will be done after the build of a release.

3.2 Watermed Core Framework Deployment Process

The core architecture will be deployed by combining the core components needed for the specific use-case and deploying the specific configuration and settings for the functional interaction of core components and use-case specific components. By that, specific use-cases can use a different set of core components in the pilots.

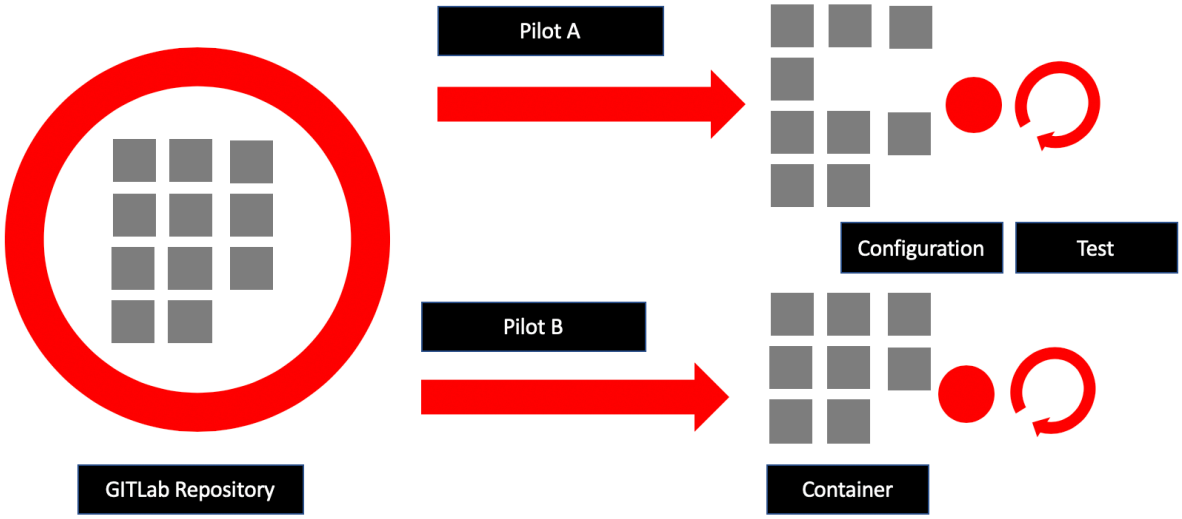

Figure 5: Deployment of use-case / pilot specific core components

4. BENCHMARKING AND TESTING ENVIRONMENT

This task aims to deploy software components in the Watermed project and ensure their quality as outlined in the KPIs defined above. In Section 2, we defined a framework to develop, build and test the different components of the architecture. For that, CI/CD (standing for Continuous Integration / Continuous Deployment) pipelines [5] on GitLab have been specified and setup in the Watermed GitLab platform.

4.1 GitLab Pipelines for Watermed Benchmarking and Testing Environment

GitLab Pipelines are the top-level component of continuous integration, delivery, and deployment. This GitLab Pipelines for Watermed comprise:

- Jobs that define what to run. For example, code compilation or test runs.
- Stages that define when and how to run. For example, that tests run only after code compilation.
- Anticipate undesirable situations or blockages in operation.
- Locate these malfunctions at code, function or process level.
- Establish the remedies and measures to be taken on the technical level (actions to be taken, even triggered automatically) in order to avoid any blocking scenario.
- Find in the code (programs) the factors responsible for these malfunctions and correct them in new versions of the code.

4.1.1 GitLab AutoDevOps for Watermed Testing, integrating and deployment of components

Testing, integrating and deployment of components in a production grade environment will be established by setting up and executing GitLab AutoDevOps [6]. Auto DevOps provides pre-defined CI/CD configuration which allows you to automatically detect, build, test, deploy, and monitor applications for Watermed. Auto DevOps is also associated with testing, integrating and deployment of components the CI/CD pipeline in GitLab. The advantage is, a developer gets a preconfigured CI/CD pipeline to automatically detect, build, test, deploy and monitor the applications. It simplifies the CI/CD setup.

The following picture shows an example for an Auto DevOps pipeline:

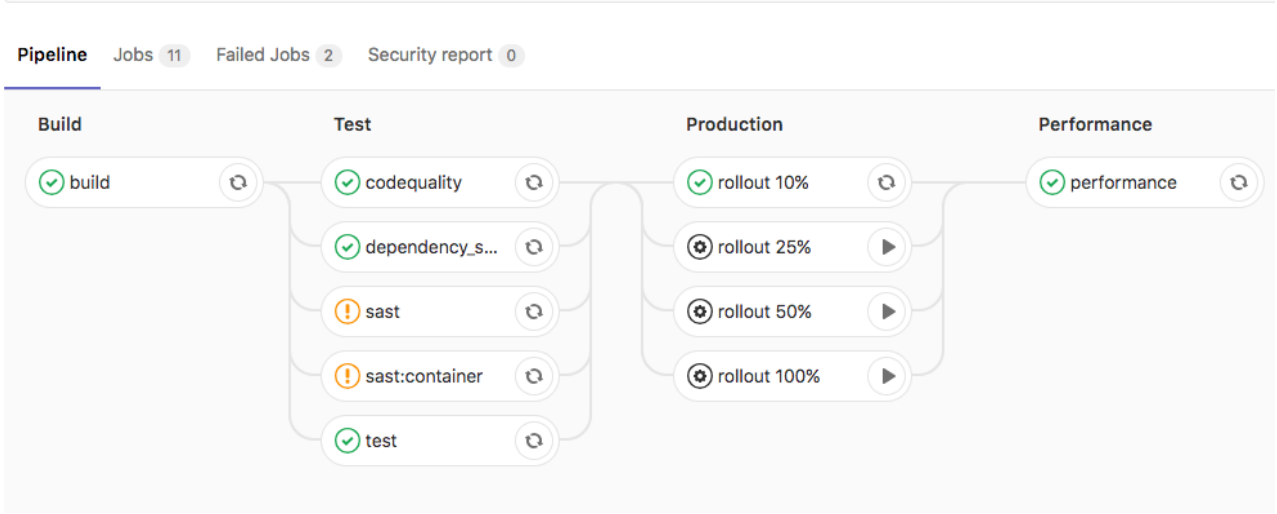

Figure 6: GitLab Auto DevOps Pipeline example

4.1.2 Auto DevOps Features on Watermed Benchmarking and Testing Environment

- **Auto Build**: Auto Build automatically detects and builds an application for Watermed. Without Auto DevOps, it is required to configure multiple stages in.gitlab-ci.yml file. The Auto DevOps takes care of it internally. It builds an application with existing Dockerfile [3] or Heroku [7] buildpacks.
- **Auto Test**: Auto Test can use Herokuish [8] and Heroku buildpacks to run the tests for an application after the build is successful. It supports test cases written in multiple languages and frameworks. The Auto Test helps developer at Watermed to identify the errors at an early stage and fix them.
- **Auto Code Quality**: uses the Code Quality image to run static analysis and other code checks on the current code. After creating the report, it's uploaded as an artifact which you can later download and check out. The merge request widget also displays any differences between the source and target branches.
- **Auto SAST**: Static Application Security Testing (SAST) uses the SAST Docker image to run static analysis on the current code, and checks for potential security issues. After creating the report, it's uploaded as an artifact which you can later download and check out. The merge request widget also displays any security warnings.
- **Auto Container Scanning**: Vulnerability Static Analysis for containers uses Clair (open source tool for Vulnerability Static Analysis for containers) to check for potential security issues on Docker images.
- **Auto Review Application**: Review Apps are temporary application environments based on the branch's code so developers, designers, QA, product managers, and other reviewers can actually see and interact with code changes as part of the review process. Auto Review Apps create a Review App for each branch. Auto Review Apps deploy your application to your Kubernetes cluster only. If no cluster is available, no deployment occurs.
- **Auto DAST:** Dynamic Application Security Testing (DAST) uses the popular open source tool OWASP ZAProxy to analyze the current code and check for potential security issues. After the DAST scan completes, any security warnings are displayed on the Security Dashboard and the merge request widget.
- **Auto Monitoring**: It is important to monitor and assess the Watermed application running on its Kubernetes cluster after deployment. The Auto Monitoring feature makes it feasible in a very easy and seamless way. It utilises Prometheus [9] open-source monitoring system to get system metrics such as CPU, or memory utilisation uses from the Kubernetes cluster.

5. EVALUATION PROCESS

For the Evaluation Process we have chosen a well-known standard as a basis, namely the ISO/IEC 17040 (2005) Conformity assessment - General requirements for peer assessment of conformity assessment and accreditation bodies [15]. As stated on its official website:

- This International Standard specifies the general requirements for the peer assessment process to be carried out by agreement groups of accreditation bodies or conformity assessment bodies. It addresses the structure and operation of the agreement group only insofar as they relate to the peer assessment process.
- This International Standard is not concerned with the wider issues of the arrangements for the formation, organization and management of the agreement group, and does not cover how the group will use peer assessment in deciding membership of the group. Such matters, which could for example include a procedure for applicants to appeal against decisions of the agreement group, are outside the scope of this International Standard.
- This International Standard is applicable to peer assessment of conformity assessment bodies performing activities such as:

a) testing,

b) product certification,

c) inspection,

d) management system certification (sometimes also called registration), and

e) personnel certification.

More than one type of activity can be included in a peer assessment process. This can be considered particularly appropriate when the body under assessment conducts combined assessments of multiple conformity assessment activities.
This International Standard is also applicable to peer assessment amongst accreditation bodies, which is also known as peer evaluation.

This International Standard focuses on the steps to be followed in the peer review process and specifies requirements other than those relating to the process only when absolutely necessary. It can be used in conjunction with ISO / IEC Guide 68 and when the peer review process is required in both regulatory and voluntary areas of conformity assessment.

The purpose of the agreement group and the use made of the results of the peer review process determine its nature. The purpose of the agreement group may include one or more of the following:

a) compliance of organizations with the specified requirements;

b) equivalence of results between organizations;

c) acceptance of the results provided by the other bodies, within the framework of their conformity assessment activities.

6. TEST METHODOLOGY

The test methodology consists in coupling the GitLab management environment with other functional and temporal verification tools, notably formal ones. Translation support of a given code can be envisaged in order to make a fine processing of the modifications brought by the developers by connecting design and realization (programming). Integrate the finer elements involved in embedded solutions, particularly in terms of parallelism (multi-core) and memory consumption. Online testing is preferred because of the performance offered by GitLab. Take care of the functional properties specifically Unit, integration and system testing, as mentioned in the beginning of this document, and understand the necessary properties such as acceptability and non-functional properties such as performance. Security, usability and compatibility will be taken into account gradually.

The architecture implemented will be tested with a range of deployment configurations involving the use of smart algorithms and analytics in the cloud, fog-based smart decisions. The kind of application implemented is essentially embedded.

The layered architecture GitLab we can consider several types of services to ensure its replication and adaptability to different crops and locations.

The IoT services, virtual entities and storage services are practically replicated, complementing with data analysis and machine learning.

6.1 Formal methods as tool for verification and validation.

Modelling and specifying a GitLab is not easy task to do. Many tools for Modelling and verifying this kind of applications; we suggest integrating formal methods. These latter are based on mathematic notions which make it sure and proved specification. Event-B is a formal method for system-level modelling and analysis. Key features of Event-B are the use of set Theory as a modelling notation, the use of refinement to represent systems at different abstraction levels (a successive refinements as Abstract Machines: AM) and the use of mathematical proof to verify consistency between refinement levels. AM describes the labelled transactions of the system [10]. A formal method is a mathematical notation and a Theory behind this notation. A formal specification and analysis allow:

- Validation: Are authors creating the right product?
- Verification: Are they creating it right?
- Unambiguous, realistic, verifiable and evolvable code.

An AM is composed of a static part, which contains the states, its invariants and its properties and a dynamic part containing transitions (events). An AM is completed by a formalism called the context. It plays an important role in model parameterization and instantiation. The Event-B is developed in the RODIN platform: Rigorous Open Development Environment for Complex Systems This platform is a tool to develop and to prove Event-B specification under Eclipse environment [11].

7. CONCLUSIONS

Defining a benchmarking and testing environment is a cornerstone milestone to be defined so that the source code of the Watermed platform components can be monitored for assuring the quality of each of them, as well as to provide an environment where different tests could be achieved. In this deliverable we have presented the approach agreed for the Watermed 4.0 project.

Actually, this deliverable has addressed these aspects covering the evaluation framework, the deployment process using the GitLab platform. We have also considered the use of CI/CD pipelines for automatizing the integration and deployment of the Watermed components provided by GitLab platform too. Moreover, we have presented a table where a set of KPIs have been identified for performance purposes. Finally, this deliverable contained two different sections for specifying the Evaluation process and the Test methodology to be achieved over the Watermed components.

8. REFERENCES

[1] Functional Testing levels: https://www.test-institute.org/Software_Testing_Levels.php

[2] Non-functional tests: https://www.testing-whiz.com/blog/types-of-non-functional-software-tests

[3] Docker Inc., 2020: https://www.docker.com

[4] ISO/IEC 25010:2011. "Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models" https://www.iso.org/standard/35733.html

[5] GitLab Docs: CI/CD Pipelines on Gitlab, 2020, GitLab, https://docs.gitlab.com/ee/ci/pipelines.html

[6] GitLab Docs: AutoDev Ops on Gitlab, 2020, GitLab, https://docs.gitlab.com/ee/topics/autodevops/#auto-devops

[7] Heroku Inc., 2020, https://www.heroku.com/

[8] GitHub, 2020, https://github.com/gliderlabs/herokuish

[9] Prometheus Open Source Monitoring System, 2020, https://prometheus.io/

[10] Abrial, J. R (2005). The B-book: Assigning programs to meanings.

[11] Maamria, I. & Fathabadi, A. (2014). Theory Plug-in User Manual, Technical report. University of Southampton.

[12] Benchmarking Peer-to-Peer Systems Understanding Quality of Service in Large-Scale Distributed Systems Wolfgang Effelsberg and Others, LNCS 7847, Thorsten Strufe (Eds.), Springer.

[13] Adam Trendowicz, Software Cost Estimation, Benchmarking, and Risk Assessment, Spinger, (English Edition), 2013.

[14] Bert Scalzo Database Benchmarking and Stress Testing: An Evidence-Based Approach to Decisions on Architecture and Technology (English Edition), Apress (Eds.), 2018

[15] ISO/IEC 17040 (2005): https://www.iso.org/standard/31815.html