Building Trust in Ecosystems and Ecosystem Components

Deliverable 8.1

BIECO Verification and Testing Strategy

Technical References

Document version	:	1.0
Submission Date	:	31/08/2021
Dissemination Level	:	Public
Contribution to	:	WP8 - Integration, Pilots, and Validation
Document Owner	:	HOLISUN
File Name	:	BIECO_D8.1_30.08.2021_V1.0
Revision	:	3.0

Project Acronym	:	BIECO
Project Title	:	Building Trust in Ecosystem and Ecosystem Components
Grant Agreement n.	:	952702
Call	:	H2020-SU-ICT-2018-2020
Project Duration	:	36 months, from 01/09/2020 to 31/08/2023
Website	:	https://www.bieco.org

This project has received funding from the European Union´s Horizon 2020 Research and Innovation Programme under Grand agreement No. 952702.

Revision History

REVISION	DATE	INVOLVED PARTNERS	DESCRIPTION
0.0	17.04.2021	HS	Skeleton and TOC Creation
0.1	24.05.2021	HS	Content added
0.2	08.06.2021	HS	Content added to section 1 and 2
0.3	15.06.2021	HS	Content added to section 3, 4 and 5
0.4	06.07.2021	HS	Content added to section 6, 7, 8, 10
0.5	15.07.2021	HS	Content improvement
0.6	27.07.2021	HS	Content added to section 9
0.7	02.08.2021	HS	Content improvement
0.8	05.08.2021	HS	Content added to section 9.1
0.9	12.08.21	IESE	Content added to section 9.12 and 9.15
1.0	17.08.2021	UNI	Review
1.1	19.08.2021	UTC	Content added to Verification Strategy Section
1.2	23.08.2021	CNR	Testing of Runtime phase added
1.3	25.08.2021	7B, IFEVS	Content added to section 9.3, 9.4. and 9.16
2.0	27.08.2021	HS	Finalizing deliverable and delivering to coordinator
2.1	28.08.2021	UNI	Revision by Coordinator
3.0	30.08.2021	UNI	Finalizing deliverable and submission

List of Contributors

Contributor(s): Oliviu Matei (HOLISUN), Daniela Delinschi (HOLISUN), Rudolf Erdei (HOLISUN), Antonello Calabrò (CNR), Said Daoudagh (CNR), Eda Marchetti (CNR), Emilia Cioroaica (IESE), Enrico Schiavone (RES), Paweł Skrzypek (7B), Radosław Piliszek (7B), Eva Sotos (GRAD), Riccardo Introzzi (IFEVS), Francisco Marques (UNI), Ovidiu Cosma (UTC), Cosmin Sabo (UTC)

Reviewer(s): Pietro Perlo (IFEVS), Sara Matheu (UMU), Sanaz Nikghadam-Hojjati (UNI), Jose Barata (UNI)

BIECO project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No **952702**.

Acronyms

Acronym	Term
AIT	AI Investments use case application
CI	Continuous Integration
DB	Database
GW	Gateway
GUI	Graphical User Interface
HW	Hardware
ICT	Information and Communication Technology
ICT GW	ICT Gateway
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
SW	Software
TC	Test Case
TS	Test Scenario
TQI	TIOBE Quality Indicator

Glossary

Term	Definition
Actor	An Actor represents a non-cyber-physical party of the ecosystem, such as a specific person, company, or some other legal entity that interacts with systems and digital assets, such as software components.
Framework	Composition of tools that interact_over well specified interfaces. It enables implementation of methods.
ICT	Information and Communication Technology - it indicates the domain of telematics, computer science, multimedia and internet.
Software Smart Agent	An intelligent software component involved in the automation of processes within a system, system component or ecosystem.
Stub	A piece of code simulating a method/object interaction and response
User	An actor or an ecosystem, or system or a system component that interacts with the ecosystems.
Validation	A set of activities intended to ensure that a system or system component meets the operational needs of the user. The user in this sense can be an actor within the ecosystem, or another system or system components that receives its services.
Verification	A set of activities that checks whether a system or a system component meets its specifications.
Vulnerability	A weakness an adversary could take advantage of to compromise the confidentiality, availability, or integrity of a resource.
Security Testing	The validation or verification process to be executed to determine that the system or component under test can: protect its data and resources; and /or maintains its properties and functionalities and/or is free from specific weaknesses.
Unit Testing	Unit tests are typically automated tests written and run by software developers to ensure that a section of an application (known as the "unit") meets its design and behaves as intended.
Integration Testing	Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group. Integration testing is conducted to evaluate the compliance of a system or component with specified functional requirements.
System Testing	System Testing is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications.
Non-Functional Testing	Non-functional testing is the testing of a software application or system for its non-functional requirements: the way a system operates, rather than specific behaviours of that system.
Continuous Integration	Continuous integration (CI) is the practice of automating the integration of code changes from multiple contributors into a single software project. It's a primary DevOps best practice, allowing developers to frequently merge code changes into a central repository where builds and tests then run
Continuous Deployment	Continuous Deployment (CD) is a software release process that uses automated testing to validate if changes to a codebase are correct and stable for immediate autonomous deployment to a production environment

Continuous Delivery	Continuous delivery (CD) is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time and, when releasing the software, without doing so manually.

Executive Summary

This deliverable presents the verification and testing strategy of BIECO platform.

The objective of software verification and testing is to give insight in quality and to minimize effort by detecting software errors in an early stage of a project life cycle.

This deliverable has been elaborated within task T8.1, which is responsible for defining the strategy for the verification and testing of the BIECO modules that will be defined in the scope of WP3, WP4, WP5, WP6 and WP7. The objective of this task is twofold: 1. to quantify and understand module performance in a meaningful way through test cases and 2. to define a sound methodology for development and testing.

The strategy described here applies to the BIECO Platform and all modules developed within the project, but it is also highly recommended for any module developed for the BIECO Platform.

Project Summary

Nowadays most of the ICT solutions developed by companies require the integration or collaboration with other ICT components, which are typically developed by third parties. Even though this kind of procedures are key in order to maintain productivity and competitiveness, the fragmentation of the supply chain can pose a high-risk regarding security, as in most of the cases there is no way to verify if these other solutions have vulnerabilities or if they have been built taking into account the best security practices.

In order to deal with these issues, it is important that companies make a change on their mindset, assuming an “untrusted by default” position. According to a recent study only 29% of IT business know that their ecosystem partners are compliant and resilient with regard to security. However, cybersecurity attacks have a high economic impact and it is not enough to rely only on trust. ICT components need to be able to provide verifiable guarantees regarding their security and privacy properties. It is also imperative to detect more accurately vulnerabilities from ICT components and understand how they can propagate over the supply chain and impact on ICT ecosystems. However, it is well known that most of the vulnerabilities can remain undetected for years, so it is necessary to provide advanced tools for guaranteeing resilience and also better mitigation strategies, as cybersecurity incidents will happen. Finally, it is necessary to expand the horizons of the current risk assessment and auditing processes, taking into account a much wider threat landscape. BIECO is a holistic framework that will provide these mechanisms in order to help companies to understand and manage the cybersecurity risks and threats they are subject to when they become part of the ICT supply chain. The framework, composed by a set of tools and methodologies, will address the challenges related to vulnerability management, resilience, and auditing of complex systems.

Partners

Disclaimer

Table of Contents

List of Figures

List of Tables

1. Introduction

BIECO is a holistic framework that will help companies to understand and manage the cybersecurity risks and threats they are subject to when they become part of the ICT supply chain. The framework, composed by a set of tools and methodologies, will address the challenges related to vulnerability management, resilience, auditing of complex systems, risk analysis, mitigation strategies and security certification harmonization. The validation of BIECO will be achieved through the application of the tools and methodologies to four use cases (energy, finance, industry, and navigation), which include also complex IoT ecosystems.

The platform is an online software portal and orchestrator that is integrating all the tools developed in BIECO project, making them easy to use and integrate in the company's workflow. The platform will be able to plug in all the use cases.

The platform developed will:

a) deploy the tools of BIECO's framework;
b) manage the datasets of the project (use cases datasets and public datasets);
c) deploy the applications of the pilots.

BIECO's building blocks will be deployed as containers within a cloud platform, which will increase the efficiency of the developments and the use of the resources. As the cybersecurity landscape evolves rapidly and new threats are emerging every day, the framework will be instantiated in an iterative manner, which will enable a continuous evaluation and improvement of the security of the supply chain.

The methodologies and tools provided by BIECO's framework will be evaluated in four use cases (Figure 1) from different sectors:

1. ICT gateway (smart grid/energy)
2. Investment portfolio optimization platform (financial)
3. Smart microfactory (industry)
4. Autonomous Navigation

Figure 1 Use cases addressed by the BIECO project

1. ICT gateway (smart grid/energy): a software system that acts as a mediator with data sources and actuators, and connects to the smart grid. Analysing the behaviour of the system from the security perspective and making it resilient against attacks and failures is essential, not only from the systems' point of view, but also to avoid the propagation of vulnerabilities to the smart grid.

2. Investment portfolio optimization platform (financial): consumers are consistently ranking trust as a more important factor in their decision of where to deposit or invest their money. Online investment platforms are complex ICT systems that manage sensitive data and need to be trustworthy, so it is necessary to secure and monitor them accordingly.

3. Smart microfactory (industry): microfactories are IoT based systems that need to be connected to the Internet and to communicate among each other. However, this offers opportunities to cybercriminals to exploit flaws and vulnerabilities, whether those flaws may be human, hardware, or software based. Systems can be challenged via non-invasive (stolen password, eavesdropping, or exploiting system bugs to gain access), semi-invasive (taking advantage of uncontrolled states or injecting faults into a system), invasive (embedding software or modifying internal signals) or physical attacks.

4. Autonomous Navigation: is meant to serve as a pre-demonstration environment for the internal workshop. It entails the assurance of trust and safety in the context of the addition/update of a new module within the navigation environment, more specifically the local planner.

2. Approach

This chapter describes the verification and validation approach within the BIECO Platform. For the testing of BIECO, the following general rules are applicable:

- Software testing is done according to the applicable guide described by sections 0 to 7;
- Reviews are done according to the applicable review process, namely peer review by the individual developing partners.

2.1. Mapping to the development process

The development and the testing activities should run in parallel. Test implementation activities should start as soon as development activities start. Clearly, the actual testing will start as soon as the target of evaluation (TOE) is available.

2.2. Testing strategy

The quality attributes and their relative importance were derived from (Gorton, 2011). The results are reported in the Table 1 below.

Table 1 Relative importance of Quality Attributes

Quality attribute	Description	Relative importance (%)
Maintainability	The degree of effectiveness and efficiency with which the product can be modified.	22% (12/53)
Performance, Scalability and Capacity	The performance relative to the number of resources used under stated conditions.	19% (10/53)
Reliability	The degree to which a system or component performs specified functions under specified conditions for a specified period of time. Includes also ‘Availability’.	28% (8+7/53)
Security	The degree of protection of information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them.	23% (12/53)
Usability	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. Includes also ‘Serviceability and Manageability’.	8% (4+0/53)

There are a series of points noteworthy to mention:

The testing requirements [1] under the “Availability” attribute were merged under ‘Reliability’ to match the quality attribute description mentioned by ISO-25010.[1]

[1] https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

The non-functional requirements under ‘Serviceability and Manageability’ were considered under ‘Usability’ since ISO-25010 does not mention this quality attribute. In [1], this attribute refers to the ease of use, installation, and management of the platform by the final user.

The relative importance in the Table 1 above gives an indication of how to divide the test effort over the quality attributes.

Relative importance has been determined based on the priority levels assigned to the non-functional requirements. The only levels considered are *Must* and *Should, since they* are stated to be the levels that designate the requirements that must be satisfied by the implemented platform. For every quality attribute, the importance was determined by summing the assigned weight (*Must*=2, *Should*=1, *others*=0) for the priority level assigned to a requirement pertaining to that quality. The total number of points across all categories was 53. This score is meant only as a purely-indicative value.

The quality attributes will be measured using TIOBE’s Quality Index (TQI) as described in section 7 Non-Functional Testing, which is also based on ISO-25010. This matching guarantees that all quality attributes mentioned above are managed using a single instrument. Serviceability and Manageability, which is not part of ISO-25010, will be evaluated using user feedback and usability testing.

Although Maintainability is not a quality that can be tested through the usual testing activities, it can be measured, monitored, and enforced statically. Hence, by testing Maintainability we refer to the compliancy of the metrics detected on the source code.

Besides the quality attributes mentioned in [1], the testing activities also include tests for *functional suitability,* namely fulfilling the Functional Requirements and Use Cases. Since implementation efforts will follow the priority assigned in such a document, testing will follow the same priority of requirement testing.

2.2.1. Quality characteristics per test level

Testing based on quality attributes needs not be done for all quality attributes on each test level. In this paragraph, the quality attributes are assigned to one or more test levels. There can be different quality attributes for different sub-systems or modules. The system’s software architecture is not yet defined at the moment of writing; thus, we refer to such components based on the use cases diagrams.

Unit testing

Security should be tested in this test level only in the modules which code handles user sensitive data.

Maintainability is checked at unit level by the developing partners that must comply, individually, to the related non-functional requirements.

Integration test

Performance should be tested in this level because there might be several interactions between different software components implemented by the same partner or by different partners. Such components will be identified in the software architecture, but according to the Use Cases definitions, the following subsystems emerge: extensive source code-based analyses, data manipulation and analysis, and forecasting are the main subject of testing activities.

Functional suitability needs be tested at integration level due to the presence of several architectural components that provide key functionality by interacting between each other. These components may be implemented either by the same partner or by different partners. In both cases, testing at this level is required.

System test

- *Security* should be covered in the system tests in order to avoid possible data leaks and unauthorized accesses caused by unchecked sensible data exchange between system components;
- *Usability* should be covered in the system tests because the user-experience depends on the whole system. All the user interfaces must be tested here;
- *Performance* should be tested in this level because the overall performance of the systems depends on multiple components working together;
- *Reliability* should be tested in this test level because malfunctioning in one component may interrupt the service offered by the whole system;
- *Functional suitability* is thoroughly tested at this level to guarantee that the key requirements (Must have) have been correctly implemented.

Acceptance test

- *Usability*, because the final user has to approve the ease-of-use of the final system;
- *Functional suitability* is finally also tested by the end user of the system, checking whether it fulfils her expectations.

The quality attributes are assigned to the test level(s) they best fit in as follows:

Table 2 Quality Attributes assigned to the Test Levels

Quality Attribute	*Unit test*	*Integration Test*	*System Test*	*Acceptance Test*
Maintainability	+			
Performance		+		
Reliability			++	
Security	++		+	
Usability			+	++
Functional suitability		+	++	+

(Empty) **The quality attribute is not an issue at this level;**
+ **This test level will cover this quality attribute;**
++ **The quality attribute will be covered thoroughly- it is a major goal at this test level.**

3. Test Implementation Principles

This section describes the basic test implementation principles that underlie the entire test approach for the BIECO project. This is further elaborated at the appropriate level in each test specification section.

3.1. Traceability

Traceability of the requirements tested at each test level is achieved as follows:

- Traceability is done by associating test-case-identifiers to requirement identifiers using dedicated traceability matrixes. The matrix has one column for the test id, one column for the functional or non-functional requirement id, and a description column dedicated for extra information;
- The matrices will be filled-in after the individual test design process and are part of this document.

Checking whether the requirements are actually tested by each test is part of the review process.

Each test case is traced to the corresponding software requirement(s), if applicable, and eventually to the corresponding use-case (using an extra column).

Test specification documentation shall indicate which software requirements are covered by each specified test.

3.2. Test Activities

The following activities have to be performed:

- **Planning and Control**

The main purpose of this activity is to provide guidance for execution and testing completion activities.

- **Execution**

This activity mainly consists of executing the specified testware using the implemented test infrastructure and generating a report on the results.

Prior the actual execution of the tests, the code will be automatically compiled and checked for software quality control. Compiler errors and warnings will also be checked, which will not allow L1 and L2 warnings/errors.

The execution of system testing and integration testing will be part of the continuous integration activity. Their execution is hence fully automated.

However, for unit testing activities, the individual partners are responsible for adopting their own strategy for testing as long as it complies with the test strategy described in the previous section.

Hence, individual partners have the ability to individually specify their own test infrastructure and test units based on the architecture of their own tool. However, for what concerns testing results reporting and logging they must comply with the guidelines mentioned below.

3.2.1. Completion

In this activity all testware, the test logs and test basis are archived and an evaluation report is generated. The following guidelines apply for all test level:

- All tests executions must be tracked;
- Test coverage (code and possibly path), when available, and test results are the main variables that need to be tracked;
- The execution logs must be in XML format, or easily convertible, in order to ease automatic report generation, and must include, besides the other information, their unique test id;
- Test execution logs must be uploaded on the designated platform;
- The tool that will be used for report generation has to support XML as output.

3.2.2. Entry criteria

Before testing can start the following entry general criteria have to be met:

- Test basis must be available as described in Table 3;
- The code must be buildable without any compiler errors and the complete environment to get from code to executable must be available;
- For static testing of documents (review) the test items must be under version control and in 'Internal proposal state';
- For static testing of code, the test items must be buildable without compiler errors.

Table 3 Test basis of BIECO

Document ID	Description	Available
D2.1	Project Requirements	M4
D2.2	Use Case Definition	M9
D2.4	Architecture Update (Final)	M12
D3.3	Report of the tools for vulnerability detection and forecasting	M18
D4.1	Report on Self-checking of vulnerabilities and failures	M30
D4.2	Report on methods and tools for the failure prediction	M24
D5.2	First version of the simulation environment and monitoring solutions	M24
D5.3	Final version of the simulation environment and monitoring tools	M30
D6.1	Blockly4SoS model and simulator	M10

Furthermore, for each test level, the following must be met as well.

Unit testing

- The technical deliverables from WP2 must be at least in an advanced draft state, because the development itself and subsequently unit testing rely entirely on the architecture described in D2.3.
- Testable codes and units are available.
- The test environment is ready.

Integration testing

- Unit testing has been successfully completed.
- Top-priority bugs found during unit testing must have been fixed and closed.
- Integration testing plan and test environment for integration testing are ready.
- The technical deliverables from WP2 must be in their final version state and tools from WP3, WP4, WP5, WP6 must be at least in an advanced draft state, so that they may be integrated in BIECO ecosystem.

System testing

- Integration testing has been successfully completed.
- Top-priority bugs found during integration testing must have been fixed and closed.
- The technical deliverables from WP2 to WP7 must be in their final version state.
- Detailed system testing plans (using WP2 and WP8 pilots as a basis) are defined and system testing environment is ready.
- Artefacts (i.e. source code) from test cases pilots defined by task T2.2 and T8.3 are available to be provided as input to the BIECO platform for system testing, because this is the stage when one can talk about BIECO system: a working platform and working tools (in a pretty advanced stage of development).

Acceptance testing

- System testing has been successfully completed and acceptance testing environment is ready to be deployed (test cases from UNI, 7B, RES, IFEVS).
- Top-priority bugs found during system testing must have been fixed and closed.
- Top-priority functional and non-functional requirements are met.
- A beta version of the system is available to be deployed to the use case partners providers (UNI, 7B, RES, IFEVS). The alpha version of the system is to be tested internally by the consortium partners. The beta version is already a public release (although with limited spread ability) meant to be tested and accepted by the End-User (that is why it is called *User Acceptance Testing*).

3.2.3. Acceptance criteria

This paragraph describes for the static and dynamic test the targets to decide whether a test has passed or failed.

Minimal test Coverage

Minimal test coverage will be measured with different tools depending on the development platform used by the developing partner. In general, for Java-based projects, JUnit should be used as a test platform at the unit level and TTCN3 language, which is a standardized testing language and TITAN tool, that executes TTCN3 tests. Table 4 shows the minimal coverage percentages per test level.

Table 4 Coverage targets for acceptance:

Test level	% Code Coverage	% Path coverage	% Requirements coverage	% Pilot test case coverage
Unit Test	50%	50%	-	-
Integration Test	-	-	20%	-
System Test	-	-	80%	33% (1/3)
Acceptance Test	-	-	-	100% (3/3)

Requirement coverage will be measured using test execution traces and logs.

Pass/Fail criteria

The Table 5 below shows the criteria whether a test pass.

Table 5 Pass/Fail criteria for test execution

Test level	Pass\fail Criterion
Unit Test	The part of the code tested complies with the expected behaviour implemented by the test.
Integration Test	The requirement is correctly implemented and fully provides the expected functionality within the constraint defined by the non-functional requirements.
System Test	The requirement is correctly implemented and fully provides the expected functionality within the constraint defined by the non-functional requirements.

3.2.4. Validation criteria

The validation criteria of delivered work products and the execution of acceptance test cases is not part of this document. Such activities are indeed part of task T8.4 and the related deliverable.

3.3. Regression Testing

Regression testing is the core activity that reduces the risk of introducing bugs in the existing source code by adding functionality, fixing other bugs, or revising existing features.

Regression testing is usually applied in the advanced stages of development when the system has already started assuming a shape and there are several functionalities already available to be used. The first regression tests should start in parallel with system testing activities.

The regression testing strategy adopted in this project will be a combination of manual and automatic regression testing. This choice allows detecting types of bugs that cannot be detected only by adopting a single strategy.

3.3.1. Manual regression

Manual regression strategy is fully delegated to development teams, which should manually check the correct execution of the changed functionalities and the adjacent areas. Since this is a very time-consuming activity, it is only advised to perform after important changes that might impact the core functionality of the system have been made.

Additionally, in order to check the code related to minor changes, it is good for development teams to prepare a checklist of minor functionalities that have to be checked and check them all together once.

3.3.2. Automated regression

This kind of testing consists in re-executing a selected set of unit and integration tests that have been found to identify multiple bugs in the past. There are different regression testing techniques depending the test coverage [2].

To select such a set, it is necessary to collect statistics of passed and failed tests during past testing activities. However, developers and testers can also suggest specific tests to be used based on their experience with the code and previous bugs. Automated regression testing is considered part of continuous integration, applicable on all test levels.

3.3.3. Recommended regression testing tools

For regression testing we recommend Jenkins which is a popular CI orchestration tool. It provides numerous plugins for integration with multiple test automation tools and frameworks into the test pipeline. When it comes to test automation, Jenkins provides plugins that help run test suites, gather and dashboard results, and provide details on failures.[2]

3.4. Issue Reporting

3.4.1. Guidelines

To maintain an effective bugfix workflow and make sure the open issues will be solved in a timely manner, the reporter will follow some simple guidelines.

Before creating an issue, please do the following:

[2] https://www.jenkins.io/doc/developer/testing/

- Check the Developer Documentation and User Guide to make sure the behaviour you are reporting is really a bug, not a feature;
- Check the existing issues to make sure you are not duplicating somebody's work;
- Make sure, that information you are about to report is a technical issue;
- If you are sure that the problem you are experiencing is caused by a bug, file a new issue.

3.4.2. Issue Template

Issue Reporting Template is a default placeholder for every new issue. Please note, that higher level of detail in the report increases chance that a developer will be able to reproduce the issue. It is hard to advice on any problems which cannot be replicated.

3.4.3. Recommended regression testing tools

For issue reporting we recommend JIRA which is a very popular project tracking software. This tool provides the full set of recording, reporting and workflow features, as well as code integration, planning and wiki. With its robust set of APIs, JIRA can be integrated with almost all tools your team uses.[3]

- **Issue Title**

Title is a vital part of bug report for developer and helps to quickly identify a unique issue. A well written title should contain a clear, brief explanation of the issue, making emphasis on the most important points.

- **Issue Description**

Preconditions

Describing preconditions is a great start, provide information on system configuration settings you have changed, detailed information on entities created (Products, Customers, etc.), Magento version. Basically, everything that would help developer set up the same environment as you have.

Steps to reproduce

This part of the bug report is the most important, as a developer will use this information to reproduce the issue. Problem is more likely to be fixed if it can be reproduced. One should precisely describe each step one has taken to reproduce the issue. Much information as possible should be included, sometimes even minor differences can be crucial.

Actual and Expected result

To make sure that everybody involved in the fix are on the same page, precisely describe the result you expected to get and the result you actually observed after performing the steps.

Additional information

[3] https://www.atlassian.com/software/jira/free

Additional information is often requested when the bug report is processed, one can save time by providing logs, screenshots, repository branch and revision that has been checked out to install BIECO or any other artifacts related to the issue at the tester's judgement.

4. Unit Testing

4.1. Purpose

Unit testing is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed.

4.2. Scope

Unit Testing is the first level of software testing and is performed prior to Integration Testing and focuses on the source code itself.

4.3. Approach

A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. In procedural programming, a unit may be an individual program, function, procedure, etc. In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/ child class. Unit testing frameworks, drivers, stubs, and mock/ fake objects are used to assist in unit testing. Unit test frameworks will be used for continuous integration purposes as well, to enable automated regression testing.

As unit testing is closely related to development, it will be adopted in the development process itself.

4.4. Recommended unit testing tool

The unit testing tools depend on the programming language. For Java, the most common tool is Junit, which is a Java unit testing framework that's one of the best test methods for regression testing. An open-source framework, it is used to write and run repeatable automated tests. As with anything else, the JUnit testing framework has evolved over time.[4]

Also, TTCN-3, which is supported by TITAN tool, is a strongly typed testing language used in conformance testing of communicating systems. TTCN-3 was developed by ETSI in the ES 201 873 series, and standardized by ITU-T in the Z.160 Series. TTCN-3 is a language for testing reactive systems, so the system accepts stimuli from the environment and issues response.

[4] https://junit.org/junit5/

5. Integration Testing

5.1. Purpose

This chapter aims at defining and describing the overall integration testing process that will be adopted within the scope of BIECO, based on the selection of an integration testing approach and use of collaborative environments and continuous integration tools to support it.

5.2. Scope

Integration Testing is the second level of testing performed after Unit Testing and before System Testing. During Integration Testing, all individual units are combined and tested as a group to expose faults in the interaction between integrated units.

5.3. Approach

There are several Integration Testing approaches and the most widely used are:

- **Big Bang**: is an approach to Integration Testing where all or most of the units are combined together and tested at one go. This approach is followed when the testing team receives the entire software in a bundle. Big Bang Integration Testing should not be confused with System Testing, as the former tests only the interactions between the units while the latter tests the entire system.
- **Top Down:** is an approach to Integration Testing where top-level units are tested first and lower-level units are tested gradually after that. This approach is followed when top-down development is performed. Usually, lower-level units are not available during the initial phases of the development, so Test Stubs are used to simulate them.
- **Bottom Up:** is an approach to Integration Testing where bottom level units are tested first and top-level units are tested gradually after that. This approach is followed when bottom-up development is performed. Usually, higher level units are not available during the initial phases of the development, so Test Drivers are used to simulate them.

Among the aforementioned Integration Testing approaches the Bottom-up approach is the most suitable for the case of the BIECO Platform. The overall platform consists of individual components (i.e., toolboxes), which are implemented independently by the relevant partners, and will be available as individual microservices, unified under the BIECO platform. Both the top-down and the bottom-up approaches fit well in the Continuous Integration testing strategy that will be adopted for the implementation of the final platform, since they allow the integration testing to begin in parallel to the actual development of the platform. They provide higher flexibility, since the individual components are integrated to the broader system as soon as they are available and functional, while the behavior of components that are not ready yet are simulated through Test Drivers and Stubs, leading in that way to a reduced time to market. Between the two hierarchical approaches, the bottom-up approach is more suitable, since the bottom-up development process will be adopted for the implementation of the platform. - Actually, the development will start from the low-level individual functionalities that the overall platform -should -provide, and will progress with gradual integration of these functionalities into broader components and modules.

Before starting Integration Testing, it is important to ensure that there is at least one proper -design document available, where interactions between each unit are clearly– specified. In D2.4 Architecture Update (Final) deliverable, main system components and interfaces between them are specified. In addition, it is important that each separate unit is Unit tested prior to Integration Testing and that all tests are properly automated to the greatest extend, since manual testing can be inefficient because developers have to retain many build artefacts and test them manually. This can be achieved by enabling Continuous Integration, i.e., the process of automating the build and testing of code every time a team member commits changes to a collaborative environment.

5.4. Continuous Integration

The overall integration approach of BIECO will be based on the use of a collaborative environment, continuous integration tools and a plan of releases. The above strategy will allow on the one hand all developers to progress with the development of their own module working in independent processes, also using their own testing tools, and on the other hand to integrate their modules with each other into major releases, adhering to the foreseen plan of releases. This will also result in detecting deficiencies early on in development, where issues are typically smaller and easier to resolve. In particular, BIECO will use Continuous Integration (Figure 2) in order to automate the execution of Unit and Integration Test scripts included as part of the main toolkit on which all main APIs of the modules are integrated (i.e. Manage TD, Manage Dependability, Manage Energy Consumption, Forecaster and Decision Support Module). These scripts initially perform a series of Unit Tests in order to assert the smooth operation of each module and ensure that the APIs are working as expected. While running, they invoke a number of testing components, each one isolated from the others, to ensure that every resource or item endpoint works exactly as specified and documented. As soon as Unit Tests are successful, Integration Test comes next. All individual units (i.e., modules) are combined according to the Integration approach described above, and tested as a group to expose faults in the interaction between integrated units while reducing the risk of new updates causing unexpected side effects.

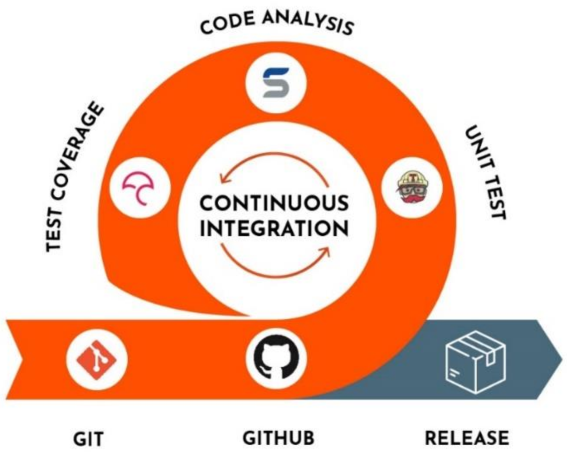


Figure 2 Continuous Integration diagram

One of the most widely used tools for Continuous Integration is Jenkins. Jenkins is used to build and test software projects continuously, allowing developers to integrate changes to their projects easily regardless of the platform they are working on. It can be integrated with a number of testing and development technologies and is entirely configurable via its friendly web GUI. Typical use-cases of Jenkins involve building an application from a version control system and running a series of automated tests. On the other hand, by using Jenkins immediate testing of the latest changes can be achieved and developers can get immediate feedback on the functionality of the written code. In case a bug emerges, the code can be reverted easily to a bug-free state without wasting too much time for debugging.

By enabling Jenkins in BIECO, the execution of tests will be triggered automatically every time a change in the code is pushed to the web-based Git-repository manager (e.g., GitHub, GitLab, etc.). As soon as the tests execution is completed, some useful pieces of information can be displayed such as the number of tests that were executed, how long did it take to execute and the details of a test failure. With Jenkins, automated testing the details of a particular failure can be accessed easily by just clicking on the corresponding link. Moreover, team members who will need to know when the tests have been completed along with the corresponding test results can be notified through Jenkins' support for email notifications.

Besides Jenkins there are other CI tools like:

- **Bitbucket Pipelines** which is a CI tool directly integrated into Bitbucket, a cloud version control system offered by Atlassian. Bitbucket Pipelines is an easy next step to enable CI if your project is already on Bitbucket. Bitbucket Pipelines are managed as code so you can easily commit pipeline definitions and kick off builds. Bitbucket Pipelines, additionally offers CD. This means projects built with Bitbucket Pipelines can be deployed to production infrastructure as well.
- **Amazon Web Services (AWS)** is one of the most dominant cloud infrastructure providers in the market. They offer tools and services for all manner of infrastructure and code development tasks. CodePipeline is their CI Tool offering. CodePipeline can directly interface with other existing AWS tools to provide a seamless AWS experience.
- **CircleCI** is CI Tool that gracefully pairs with Github, one of the most popular version control system cloud hosting tools. CircleCi is one of the most flexible CI Tools in that it supports a matrix of version control systems, container systems, and delivery mechanisms. CircleCi can be hosted on-premise or used through a cloud offering.
- **Travis CI** is a CI platform that automates the process of software testing and deployment of applications. It's built as a platform that integrates with GitHub projects so that developers can start testing their code on the fly. With customers like Facebook, Mozilla, Twitter, Heroku, and others, it's one of the leading continuous integration tools on the market.

For the implementation of the BIECO project we chose Jenkins because is the number one open-source for automating the project.

Advantages of using Jenkins are the followings:

- It is an open-source tool with great community support.
- It is easy to install.

- It has 1000+ plugins to ease your work. If a plugin does not exist, you can code it and share it with the community.
- It is free of cost.
- It is built with Java and hence, it is portable to all the major platforms.

6. System Testing

6.1. Purpose

This chapter aims at defining and describing the overall system testing process that will be adopted within the scope of BIECO, based on the selection of a system testing approach and partial use of collaborative environments and continuous integration tools to support it.

6.2. Scope

System Testing is the third level of testing performed after Integration Testing and before Acceptance Testing. During System Testing, the software complete and integrated software is tested to verify if the functional requirements are correctly implemented.

6.3. Approach

The most widely used approach for system testing is Black box testing also known as Behavioral Testing. Black box testing is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.

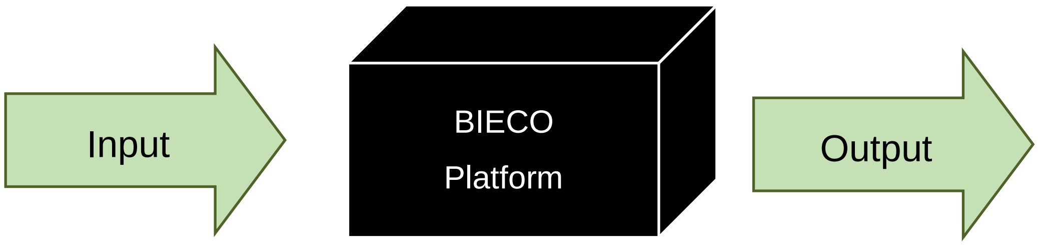


Figure 3 Black box testing

This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following categories:

- Incorrect or missing functions;
- Interface errors;
- Errors in data structures or external database access;
- Behavior or performance errors;
- Initialization and termination errors.

Following are some techniques that can be used for designing black box tests.

- **Equivalence Partitioning:** It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.
- **Boundary Value Analysis:** It is a software test design technique that involves the determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data.

- **Cause-Effect Graphing:** It is a software test design technique that involves identifying the cases (input conditions) and effects (output conditions), producing a Cause-Effect Graph, and generating test cases accordingly.

7. Non-Functional Testing

7.1. Purpose

The goal of non-functional testing is to retrieve metrics from the targeted source code that give a measure to indicate maintainability, reliability, compatibility, security and to some extend functional suitability and performance efficiency.

7.2. Scope

The focus is on code quality as opposed to e.g., quality of requirements or the architecture. Also, after describing the metrics, indicators for the effort to improve the software are given and the relations between metrics are described as well.

7.3. Reliability and Security

7.3.1. Code Coverage

In order to test the functionality of source code, it is important that developers write unit tests and make sure that these tests are applied via automated scripts to detect regressions as soon as possible. The maturity of unit tests can be measured with the aid of "statement coverage" and "branch coverage"

These metrics indicate the percentage of tested lines of code and the percentage of tested branches in the software, respectively. If the test coverage is low then either some parts of the code are not tested at all or some parts of the code are not reachable at all.

The TQI takes the average of the "statement coverage" and the "branch coverage".

7.3.2. Abstract Interpretation

Abstract interpretation, also known as "deep flow analysis", is a rather new technology that is capable of finding all kinds of fatal errors in software without actually running it. This is done by inspecting all possible execution paths through the code. In this way issues can be found such as "null pointer dereferences", "array out of bounds", "division by zero", "memory leaks" and "resource leaks" (for instance a database connection is opened but never closed for a certain execution path). Therefore, abstract interpretation will also cover some security related aspects.

The detected violations of this kind of fatal errors are weighted based on their importance and quantity. The eventual result is mapped on a scale between 0 and 100 according to a method as described in [3]. This is called the "compliance factor" or in short "compliance".

7.3.3. Compiler Warnings

Most software programs must be compiler before they can be executed. A compiler issues both compiler errors and compiler warnings during this process. If there are compiler errors in a program it can't be executed. On the other hand, compiler warnings are non-fatal but are an important indication whether there are still important issues in the software.

Compiler warnings are valued in the TQI in the same way as abstract interpretation. The number of occurrences is taken into account together with the importance of a compiler warning. This is the compiler warning compliance. The results are mapped on a scale between 0 and 100 according to a method as described in [3]. The compliance of compiler warnings is valued as defined in the table below.

7.4. Testability

7.4.1. Cyclomatic Complexity

The cyclomatic complexity of a function calculates the number of linear-independent execution paths of a function as defined by McCabe [4]. This metrics is used to measure the code complexity and testability of a software system. Usually, the average cyclomatic complexity of all functions is measured. An average cyclomatic complexity lower than 3 is generally considered as being very good.

7.4.2. Modularity

The modularity of a system at code level is measured by calculating the number of external dependencies per module, this is also called “fan out”. In case the average number of dependencies per module is high, it becomes hard to understand the software system and to test it in isolation. Moreover, the chances to reuse parts of the system is low in such a case.

7.5. Maintainability

7.5.1. Coding Standard

TIOBE is a high-tech company specialized in measuring and monitoring software code quality and they defines and maintains coding standards for various programming languages for their customers. These standards consist of generally accepted rules to which developers should adhere to in order to prevent errors and maintenance issues.

The coding standard TQI value is calculated in a similar way as is done for metrics “compiler warnings” and “abstract interpretation”. Besides the number of violations against the standard, also the severity of the violations and the size of the system is taken into account. The calculated results are mapped on a scale from 0 to 100 according to the method as described in [3]. The TQI value of this compliance factor for coding standards is as follows.

7.5.2. Duplicated Code

If a software system contains a lot of similar code at various locations, then this might influence the maintainability of the system. Suppose that a bug has been fixed in such a piece of code, then there is chance that the bug won't be fixed at one of the duplicated code locations.

Duplicate code has the following TQI score. Duplication measured by identifying 100 consecutive identical tokens without taking comments and layout into account.

7.5.3. Dead Code

Dead code in a software system is unnecessary waste. It costs maintenance effort. Despite the fact that this metric only counts for a very small part of the total code quality, it is a good indication of tidiness of the system.

7.6. Metric Relations

In order to improve on a specific metric, one has to put effort into the software engineering activities, aimed at this improvement. How much effort that will cost, can't exactly be described. What can be described though, is the comparison in effort to improve on a specific metrics. If we then consider the influence metrics have on each other, strategic planning can be applied.

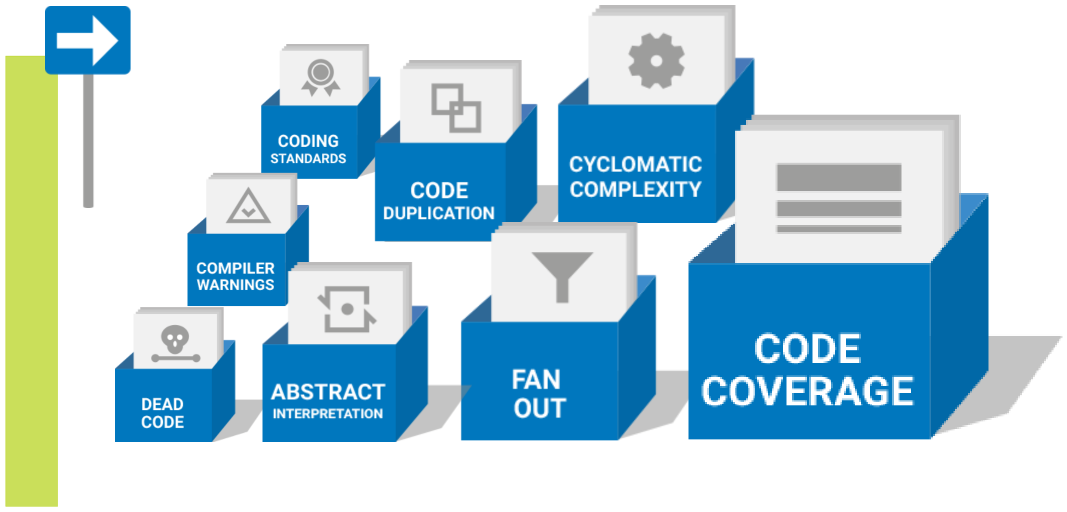


Figure 4 Metric effort comparison

The Figure 4 above indicates the effort to improve on a metric. The larger the box, the more effort it will cost to improve on that particular metric. From there the prioritization scheme can be retrieved.

7.6.1. Recommended non-functional testing tool

For Non-Functional testing we recommend Apache JMeter™ application which is open-source software, a 100% pure Java application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.[5]

[5] https://jmeter.apache.org/

Apache JMeter may be used to test performance both on static and dynamic resources, Web dynamic applications.

It can be used to simulate a heavy load on a server, group of servers, network or object to test its strength or to analyze overall performance under different load types.

Apache JMeter features include:

- Ability to load and performance test many different applications/server/protocol types:
    - Web - HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, …)
    - SOAP / REST Webservices
    - FTP
    - Database via JDBC
    - LDAP
    - Message-oriented middleware (MOM) via JMS
    - Mail - SMTP(S), POP3(S) and IMAP(S)
    - Native commands or shell scripts
    - TCP
    - Java Objects
- Full featured Test IDE that allows fast Test Plan recording (from Browsers or native applications), building and debugging.
- CLI mode (Command-line mode (previously called Non GUI) / headless mode) to load test from any Java compatible OS (Linux, Windows, Mac OSX, …)
- A complete and ready to present dynamic HTML report
- Easy correlation through ability to extract data from most popular response formats, HTML, JSON, XML or any textual format
- Complete portability and 100% Java purity.
- Full multi-threading framework allows concurrent sampling by many threads and simultaneous sampling of different functions by separate thread groups.
- Caching and offline analysis/replaying of test results.
- Highly Extensible core:
    - Pluggable Samplers allow unlimited testing capabilities.
    - Scriptable Samplers (JSR223-compatible languages like Groovy and BeanShell)
    - Several load statistics may be chosen with pluggable timers.
    - Data analysis and visualization plugins allow great extensibility as well as personalization.
    - Functions can be used to provide dynamic input to a test or provide data manipulation.
    - Easy Continuous Integration through 3rd party Open-Source libraries for Maven, Gradle and Jenkins.

8. Continuous Integration, Deployment and Delivery of BIECO

8.1. Continuous Integration

Continuous integration boils down to the practice where developers merge together their sources in a code repository [5]. A build system then builds the sources and test frameworks run their available tests. Doing these steps manually is laborious and cumbersome. However, by automating this process, it becomes very powerful as build and test results are quickly available and created consistently.

Continuous integration and Continuous Delivery are the processes in which the development team involves frequent code changes that are pushed in the main branch while ensuring that it does not impact any changes made by developers working in parallel (Figure 5). The aim of it is to reduce the chance of defects and conflicts during the integration of the complete project.

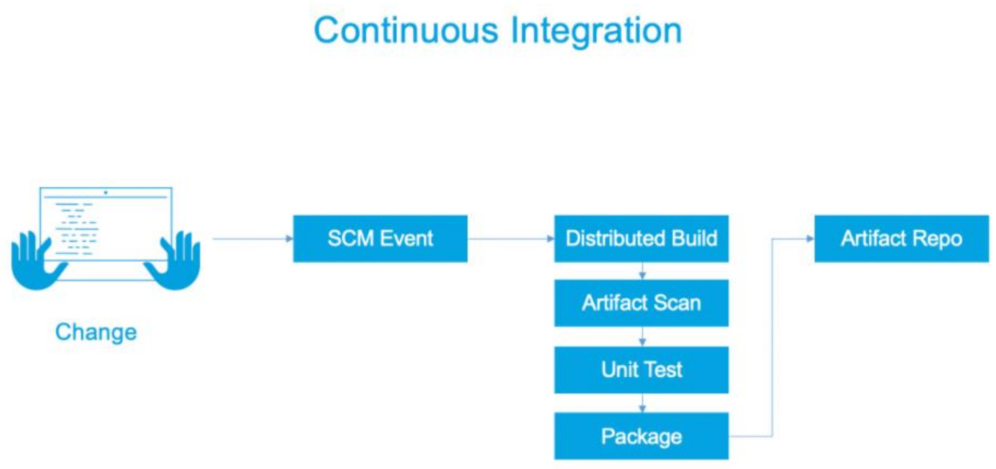


Figure 5 Flow diagram for Continuous Integration

Continuous Integration is a development methodology that involves frequent integration of code into a shared repository. The integration may occur several times a day, verified by automated test cases and a build sequence. It should be kept in mind that automated testing is not mandatory for CI. It is only practiced typically for ensuring a bug-free code.

The benefits of continuous integration for our application development lifecycle are listed below:

- **Early Bug Detection**: If there is an error in the local version of the code that has not been checked previously, a build failure occurs at an early stage. Before proceeding further, the developer will be required to fix the error. This also benefits the QA team since they will mostly work on builds that are stable and error-free.
- **Reduces Bug Count:** In any application development lifecycle, bugs are likely to occur. However, with Continuous Integration and Continuous Delivery being used, the number of bugs is reduced a lot. Although it depends on the effectiveness of the automated testing scripts. Overall, the risk is reduced a lot since bugs are now easier to detect and fix early.

- **Automating the Process:** The Manual effort is reduced a lot since CI automates build, sanity, and a few other tests. This makes sure that the path is clear for a successful continuous delivery process.
- **The Process Becomes Transparent:** A great level of transparency is brought in the overall quality analysis and development process. The team gets a clear idea when a test fails, what is causing the failure and whether there are any significant defects. This enables the team to make a real-time decision on where and how the efficiency can be improved.
- **Cost-Effective Process:** Since the bug count is low, manual testing time is greatly reduced and the clarity increases on the overall system, it optimizes the budget of the project.

8.2. Continuous Deployment

Continuous deployment is similar to continuous integration. It is the process where your application can be deployed at any time to production or test environment if the current version passes all the automated unit test cases [6].

Continuous Deployment focuses on the deployment; the actual installation and distribution of the bits. During a deployment, the application binary/packaging can transverse the topology on where the application or application infrastructure needs to serve traffic (Figure 6). In the traditional sense, Continuous Deployment focuses on the automation to deploy across environments or clusters. As you traverse environments from non-prod to the staging environment and eventually to production, the number of endpoints you deploy to increases. Continuous Deployment focuses on the path of least resistance to get the software into the needed environment(s).

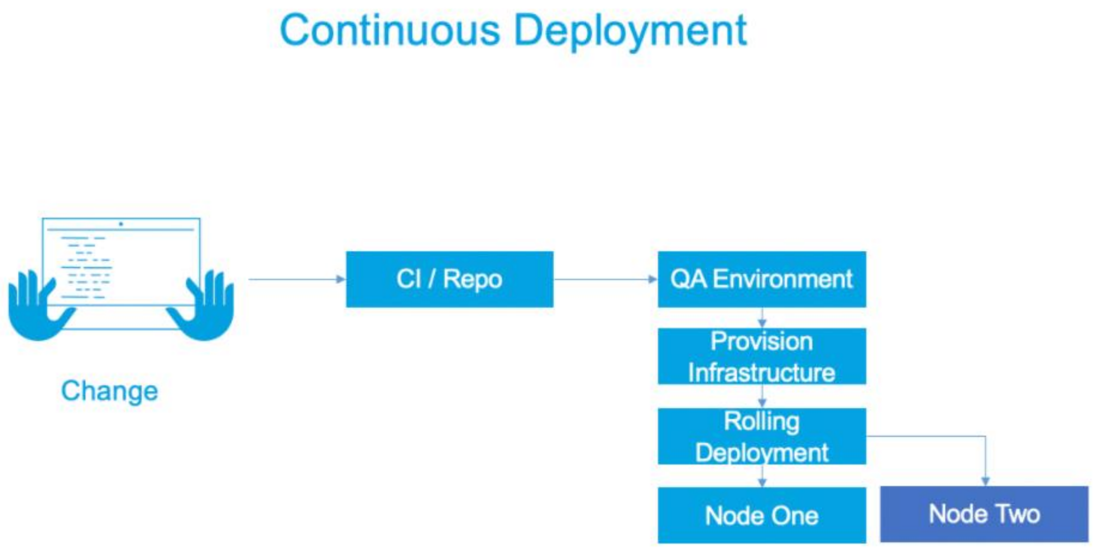


Figure 6 Flow diagram for Continuous Deployment

Deployments encompass two pairs: the installation/activation pair and the uninstallation/deactivation pair. From a pure deployment standpoint, leveraging a rolling deployment is the de facto standard. A rolling deployment allows for old application nodes to be replaced in an incremental interval, typically one by one, until all the nodes are the new version. The application instance/node being upgraded is taken out of the load balancer pool, then when the installation is complete, it is reconstituted back into the pool.

Having a clear map of the topology, especially if the infrastructure is elastic or on-demand, is key to understanding where your artifacts are going. Similar to the goals of Continuous Integration, keeping the deployment fast is a good goal to have. The appearance of speed can be there if certain tasks have to be run in parallel (i.e.: spinning up the infrastructure for artifacts to be deployed onto).

8.3. Continuous Delivery

Continuous delivery is the process of getting all kinds of changes to production. Changes may include configuration changes, new features, error fixes etc. They are delivered to the user in a safe, quick and sustainable manner [7] .

The goal of Continuous Delivery is to make deployment predictable and scheduled in a routine manger. It is achieved by ensuring that the code always remains in a state where it can be deployed whenever demanded, even when an entire team of developers is constantly making changes to it (Figure 7). Unlike continuous integration, testing and integrating phases are eliminated and the traditional process of code freeze is followed.

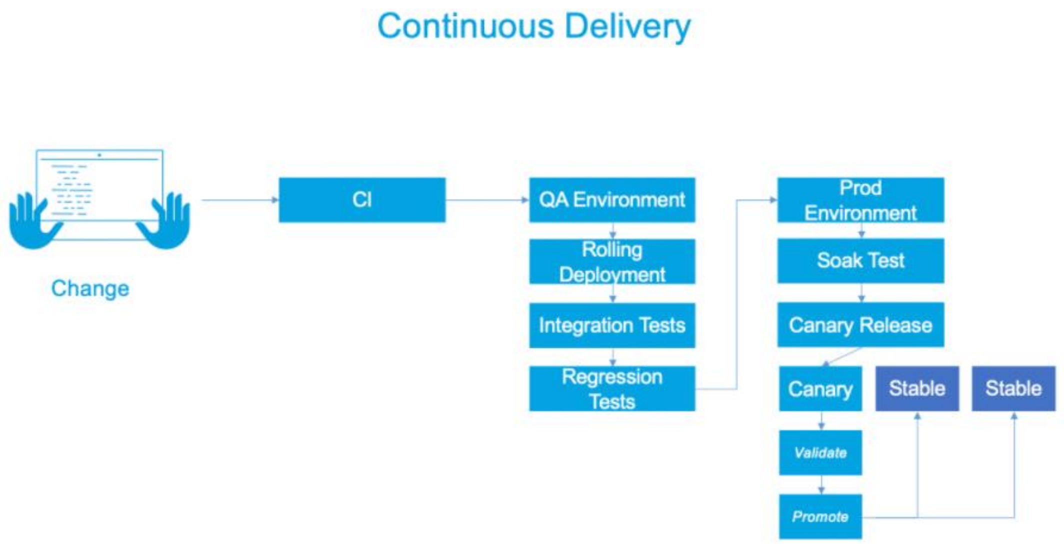


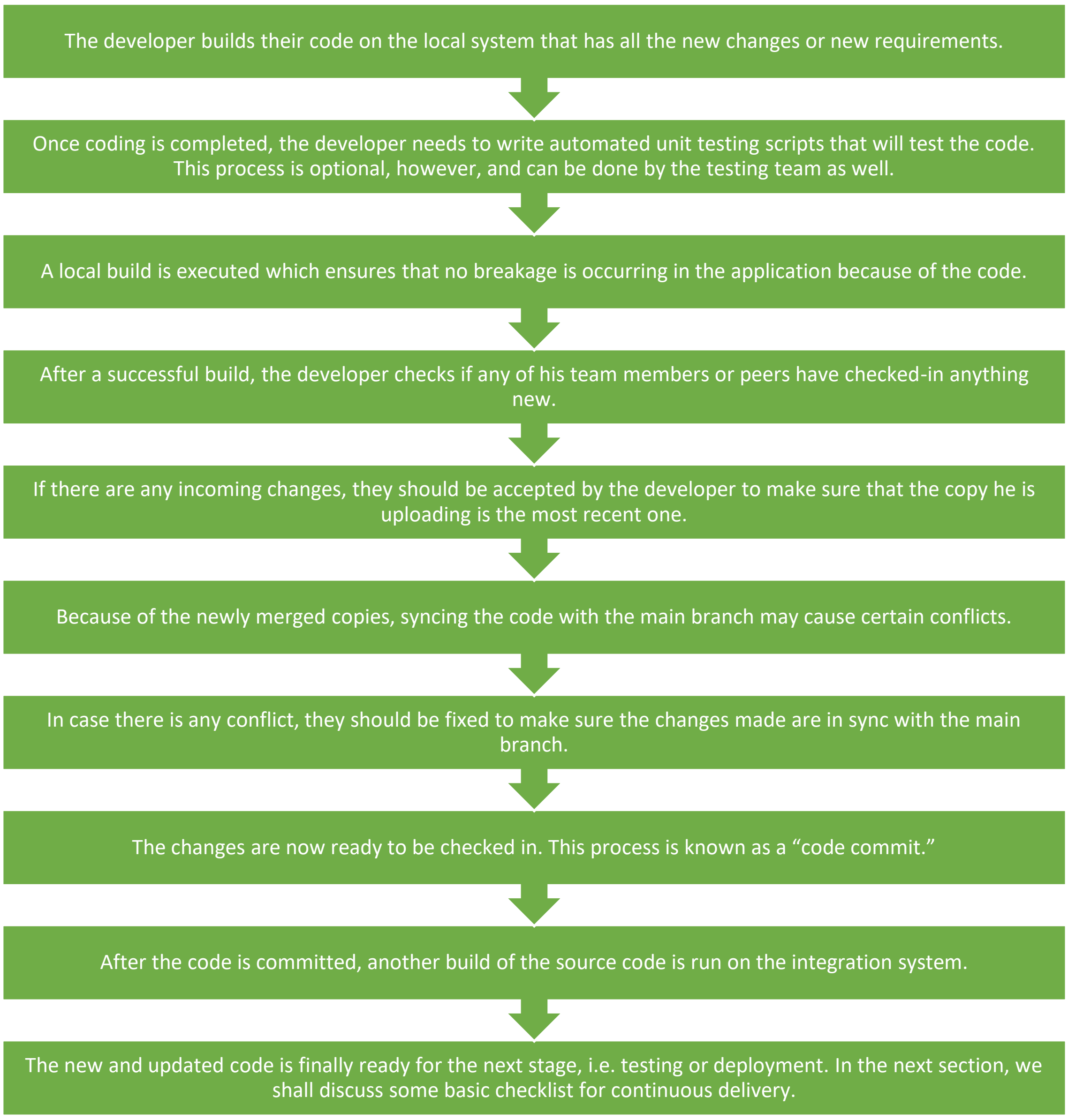
Figure 7 Flow diagram for Continuous Delivery

The benefits of continuous delivery for our application development lifecycle are listed below:

- **Reducing the Risk:** The main goal of Continuous Delivery is to make deployment easier and faster. Patterns like blue-green deployment make it possible to deploy the code at very low risk and almost no downtime, making deployment totally undetectable to the users.
- **High-Quality Application:** Most of the process is automated, testers now have a lot of time to focus on important testing phases like exploratory, usability, security and performance testing. These activities can now be continuously performed during the delivery process, ensuring a higher quality application.

- **Reduced Cost:** When an investment is made on testing, build and deployment, the product evolves quite a lot throughout its lifetime. The cost of frequent bug fixes and enhancements are reduced since certain fixed costs that are associated with the release is eliminated because of continuous delivery.
- **Happier Team and Better Product:** Since the aim of Continuous Delivery is to make a product release painless, the team can work in a relaxing manner. Because of frequent release, the team works closely with users and learn what ideas work and what new can be implemented to delight the users. Continuous user feedback and new testing methodologies also increase the product's quality.

The process flow for performing continuous delivery:



The developer builds their code on the local system that has all the new changes or new requirements.

Once coding is completed, the developer needs to write automated unit testing scripts that will test the code. This process is optional, however, and can be done by the testing team as well.

A local build is executed which ensures that no breakage is occurring in the application because of the code.

After a successful build, the developer checks if any of his team members or peers have checked-in anything new.

If there are any incoming changes, they should be accepted by the developer to make sure that the copy he is uploading is the most recent one.

Because of the newly merged copies, syncing the code with the main branch may cause certain conflicts.

In case there is any conflict, they should be fixed to make sure the changes made are in sync with the main branch.

The changes are now ready to be checked in. This process is known as a "code commit."

After the code is committed, another build of the source code is run on the integration system.

The new and updated code is finally ready for the next stage, i.e. testing or deployment. In the next section, we shall discuss some basic checklist for continuous delivery.

8.4. Common Practice

Best practices for Continuous Integration, Continuous Deployment and Continuous Delivery that should be followed by all software professionals as well as organizations are the following:

- **Keep a Central Repository:** A large project involves multiple developers constantly pulling and pushing codes that are organized together to build the application. A revision control system should be kept that will help the team to get the latest clean code from the repository at any point of time during the development cycle.
- **Automated Deployment and Build**: Automated build ensures that the team only gets the latest source code available in the repository and it is compiled every time before the final product is built. Automated Build cycle also allows the developers to push the code into different environments quickly, saving a lot of time.
- **Include Automated Unit Testing**: This will help the team to detect bugs before the code is pushed in the repository. Unit testing, as well as interface testing, have greater clarity on the product's state before it is released. Testing phase becomes easier and issues can be fixed rapidly.
- **Test in the Production's Clone:** Often an application that has passed all testing scenarios fails when it is deployed in production because of the environment is different. To prevent this, testing should be executed in an environment that is exactly the same as the production environment. This will allow testers and developers to understand how the application behaves before it is deployed into production.
- **Commit the Code Everyday**: To prevent any conflicts, developers should make it a mandatory practice to commit the code every day in the repository. It provides very little scope to look for errors occurring due to conflicts. It also improves the communication between the team members and allows developers to divide their work into small sections and track the progress of their code.
- **Build Faster:** Continuous integration fundamental purpose is to get feedback instantly after a build. A quick and perfect build keeps the development team ahead and prevents any bottleneck that may occur during unit testing.
- **Everyone can see what others are doing**: Continuous Integration and Continuous Delivery essential goal is to make the communication between team members smooth and effective. Everyone should have a clear idea regarding the state of the application and the latest changes that are made on it. Builds that have failed should be reported immediately to the stakeholders who can then make the relevant changes. IMs, Emails and other monitoring tools are used by various organizations to monitor the state of the builds.

The first step is to maintain a common repository. If possible, each component of the BIECO will have its own placeholder in the repository, where each developer can commit its sources or so-called artifacts. This is needed so a Continuous Integration tool (CI) can retrieve these sources to provide it to a build system. During continuous integration, continuous deployment, and continuous delivery the process is designed to deal both with tools that can provide the source-code (preferable from the common repository), as well as already built, proprietary tools.

The second step is to have one or more build systems, one for each component. These build systems will be provided with the sources, retrieved from the repository by the CI. The build will be invoked by the CI as well. The CI then can verify, looking at the standard out or standard error whether the build succeeded or not.

Lastly, the third step is used to plug-in test and quality frameworks. The CI will invoke the configured frameworks, so each framework can fulfill its duty. The CI can also collect the results from these frameworks for presentation and reporting purposes.

The steps above are repeated at least once a day. It is common practice to apply an incremental approach to speed-up the process and have quick feedback.

8.5. Continuous Integration Tools

For Continuous Integration [8] there are the following tools:

- **Jenkins**: An open-source Java-based CI tool that is platform independent. The best part is, it can be configured both using a console or a graphical user interface.
- **Team City:** This is a cloud-based CI server, developed by JetBrains. Although the enterprise edition is paid, there is a free version as well that allowed 3 build agents and a maximum of 100 builds.
- **Travis CI:** One of the oldest Continuous Integration and Continuous Delivery solution, the tool is free for all projects that are open source. It is hosted on GitHub and based on the usage you can choose the appropriate package from several options.
- **Gitlab:** The CI developed by GitLab is cloud-based, hosted on their official website. It is supported on multiple platforms and has both free and paid versions.
- **Circle CI:** A cloud-based CI tool, it supports GitHub and languages like Node.js, Java, Ruby, Python, Scala, Haskell, and PHP. It allows the parallel building of your code.
- **Codeship:** This is also another hosted tool that comes with basic as well as enterprise editions. The basic version comes with several packages and with expensive enterprise edition, it brings you more options to run parallel builds.
- **SonarQube:** This is a Code Quality Assurance tool that collects and analyzes source code, and provides reports for the code quality of your project. It combines static and dynamic analysis tools and enables quality to be measured continually over time.

To enable continuous integration, BIECO employs Jenkins. Jenkins is an automation server that enables one to automate repetitive actions during software development. Jenkins highly integrates with version control tools and build systems. It is capable of executing scripts on external nodes, making it a powerful tool to be the center of integration for a software project.

To apply the measuring method, the code quality framework SonarQube which integrates with Jenkins, build systems, repositories and numerous code checkers. Depending on the used programming language, different code checkers can be employed to measure the TQI.

In Figure 8 is represented the BIECO Collaboration framework:

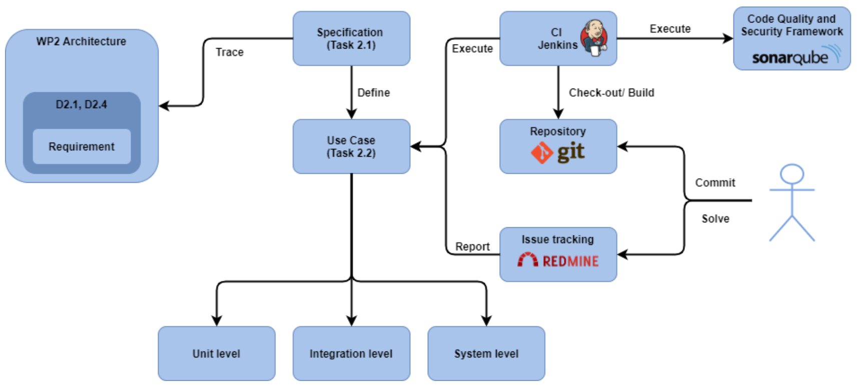


Figure 8 Collaboration framework

9. Verification Strategy

This Section recalls the general goals of the project, already introduced in D2.1, with the aim of deriving from them the specific goals of UC1, as well as requirements and KPIs related to the ICT GW Use Case that are presented in the following sections. The general goals are listed below:

- G1 – Providing a framework that will allow the reinforcement of trust in ICT supply chains;
- G2 – Performing advanced vulnerability assessment over ICT supply chains;
- G3 – Achieving resilience in ecosystems formed by unreliable components;
- G4 – Extending auditing process to evaluate interconnected ICT systems;
- G5 – Providing advanced risk analysis and mitigation strategies that support a view of the complete ICT supply-chain;
- G6 – Performing evidence-based security assurance and a harmonized certification for ICT systems;
- G7 – Industrial validation of BIECO's framework within IoT ecosystems.

9.1. Test Plan for BIECO Platform

BIECO Platform has 3 actors:

- Tool Developer;
- End-User;
- Platform Administrator.

The scenarios for each of the actors is depicted in Figure 9.

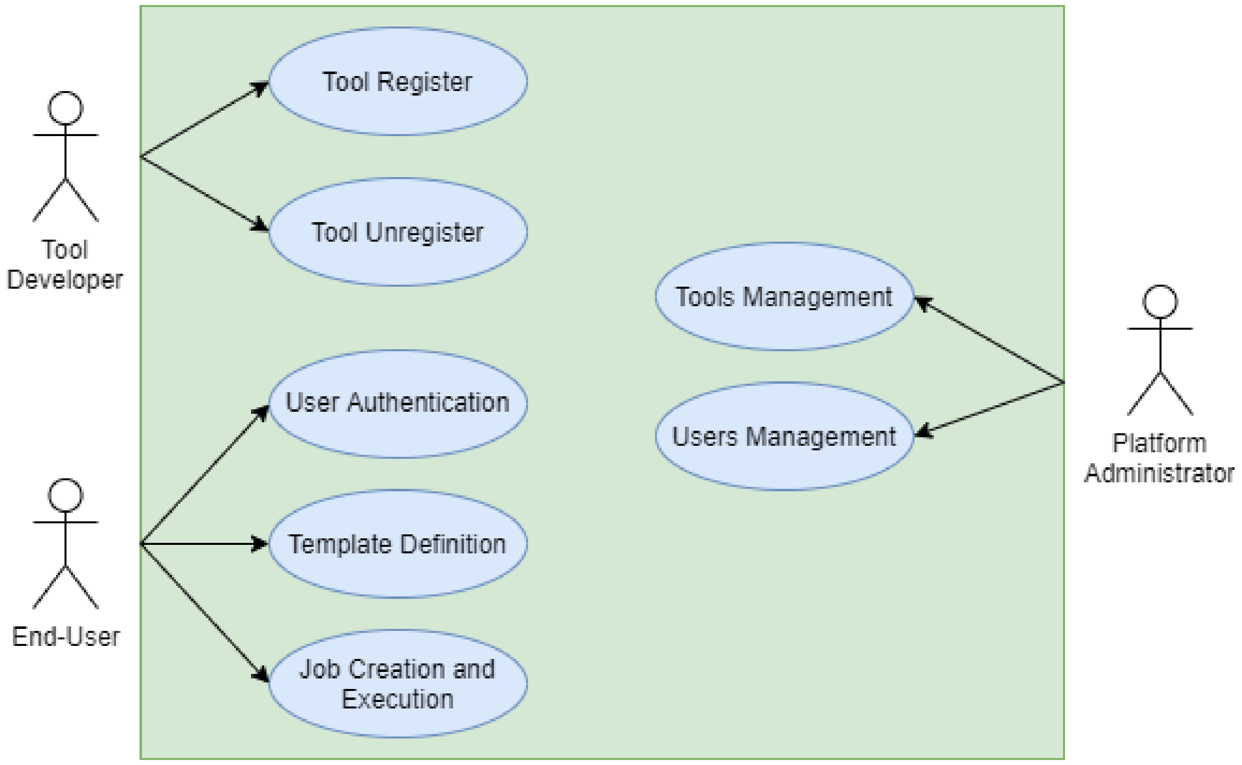


Figure 9 Actor scenarios for BIECO Platform

The test scenarios described in the next sections are based on this figure.

The tools involved in the Test Plan for BIECO Platform:

Tool	How it is involved
jUnit[6]	Unit tests are written for all of the components. At each run, all tests must pass.
SonarQube[7]	This tool will provide quality and security assessment for all the source code written for the platform. Vulnerabilities and bugs will be discovered easier and solved before proceeding to the next steps of development and testing.
Selenium[8]	Automated Selenium based tests will be designed and deployed in order to assure system integration and platform validity.

9.1.1. Test scenario identifier 1

Test Scenario ID	HS-TS-01
Test Scenario Name	User Authentication and Validation
Test Case Description	Testing the functionalities associated with the user (actor) access to the platform. This includes: user register, user activation, user authentication.
Actors	End-User, Tool Developer, Platform Administrator
Pre-Conditions	Actor must have a valid email address and access to it.
Post-Condition	Actor will have a valid user account and be able to login to the platform.
Associated goal	G1

9.1.1.1 Test-case-identifier 1.1

Test Case ID	HS-TC-01-1
Test Case Description	Actor can register for a new End-User account.
Pre-Conditions	Actor must have a valid email address. Actor must introduce information in all required fields. Email address must be valid and accessible by the Actor
Test Steps	- Introduction of valid data into the fields; - Introduction of valid data into the fields with a repeated email address; - Introduction of data that does not comply with platform requirements.
Test Data	Both valid and invalid faked user information.
Expected Result	- Platform must allow the existence of only one copy of an email address; - Platform must invalidate and prevent registration for users that enter invalid data.
Post Condition	- Actor must have an activate and valid account, if valid data was used; - Registration must be prevented if invalid data or repeated email address was used.
Actual Result	The users table must have only valid information entered into it.

[6] https://junit.org/junit5/
[7] https://www.sonarqube.org/
[8] https://www.selenium.dev/

9.1.1.2. Test-case-identifier 1.2

Test Case ID	HS-TC-01-2
Test Case Description	Actor can login into the BIECO platform.
Pre-Conditions	Actor must have an activated account.
Test Steps	- Introduction of valid data into the email and password fields; - Introduction of invalid data into the email and password fields; - Check when the fields are blank and submit button is clicked.
Test Data	Both valid and invalid (faked) actor credentials.
Expected Result	- Platform must allow user to login only if the email and password entered are valid and actor has an activated account; - When the required fields are not entered correctly the user should not be able to login and an error message should be displayed.
Post Condition	- Actor is successfully logged in the platform; - Login must be prevented if invalid data was used.
Actual Result	The user has successfully logged in or not.

9.1.2. Test scenario identifier 2

Test Scenario ID	HS-TS-02
Test Scenario Name	Template Definition and Visualisation
Test Case Description	Testing the functionalities associated with the template creating and editing. This includes: template create, save, edit, delete and visualisation.
Actors	End-User, Tool Developer, Platform Administrator
Pre-Conditions	An activated and authenticated user
Post-Condition	Actor will have a valid template and will be able to use it in the Job section.
Associated goal	G2, G3, G4, G5

9.1.2.1. Test-case identifier 2.1

Test Case ID	HS-TC-02-1
Test Case Description	Actor can create and edit his own template
Pre-Conditions	Actor must be correctly authenticated in the platform.
Test Steps	- Create a valid template where all the tools have inputs and outputs; - Attempt to create an invalid template where no tools are defined.
Test Data	Both valid and invalid template creation
Expected Result	- Platform allows the creation of a valid template; - Error message when the template is not valid, preventing the template to be saved; - The save button saves the valid template in the database; - The actor can set the visibility of the template created between Public and Private possibilities; - Actor must be able to edit only the templates created by him.

Post Condition	Actor will have a valid template which will be use after in the Job section.
Actual Result	The actor has successfully created a template

9.1.2.2. Test-case identifier 2.2

Test Case ID	HS-TC-02-2
Test Case Description	Actor can view public templates and the ones created by him.
Pre-Conditions	Actor must have an activated and authenticated account.
Test Steps	- Visualisation the templates created by actor; - Visualisation of public templates; - When the template is private the platform must not allow the other user to view or use it.
Test Data	- User and public templates.
Expected Result	- Actor is able to visualise the public templates and the ones created by him; - The Private templates should not be displayed to other users.
Post Condition	Actor must successfully view the correct templates.
Actual Result	Actor can successfully view the correct templates.

9.1.3. Test scenario identifier 3

Test Scenario ID	HS-TS-03
Test Scenario Name	Job Creation and Execution
Test Case Description	Testing the functionalities associated with the job creating and execution. This includes: the job definition, save, edit, view and run, the history information regarding the execution of the job.
Actors	End-User, Tool Developer, Platform Administrator
Pre-Conditions	An activated and authenticated user
Post-Condition	Actor will have a defined Job.
Associated goal	G2, G3, G4, G5

9.1.3.1. Test-case identifier 3.1

Test Case ID	HS-TC-03-1
Test Case Description	Actor can create a Job.
Pre-Conditions	Actor must have an activated and authenticated account.
Test Steps	- Create a new Job, introduce the name of the job and select the desired Template; - Save the new Job if all fields are completed;
Test Data	- Job name and desired Template
Expected Result	Correct generation of Job with the name and Template provided.
Post Condition	Existence of generated Job.
Actual Result	The Job has the name and Template provided by the actor.

9.1.3.2. Test-case identifier 3.2

Test Case ID	HS-TC-03-2
Test Case Description	Actor can run a Job.
Pre-Conditions	Actor must have an activated and authenticated account.
Test Steps	- Opening an existing Job; - Input the needed data in the fields; - Input invalid and/or incomplete data; - Click the run button;
Test Data	Depending on tools used in the Template
Expected Result	- Prevention of Job start if the data is incomplete or invalid; - Job is running in parameters if all the provided information is correct.
Post Condition	Job information from the Platform is presented to the actor.
Actual Result	The table of events and other data is visible in real-time on the UI.

9.1.4. Test scenario identifier 4

Test Scenario ID	HS-TS-04
Test Scenario Name	Tool Registration
Test Case Description	Testing the functionalities associated with the tool registration and un-registration. This includes: the tool definition, save, edit, view.
Actors	Tool Developer, Platform Administrator
Pre-Conditions	An activated and authenticated user
Post-Condition	Actor will have a registered Tool.
Associated goal	G2, G3, G4, G5

9.1.4.1. Test-case identifier 4.1

Test Case ID	HS-TC-04-1
Test Case Description	Actor can register a Tool.
Pre-Conditions	Actor must have an activated and authenticated account.
Test Steps	- Registration of a new Tool, introduce the name of the Tool and introduce the inputs required for the Tool functionality; - Save the new Tool if all fields are completed;
Test Data	Tool name and inputs required
Expected Result	Correct registration of Tool with the name and inputs provided.
Post Condition	Existence of a registered Tool.
Actual Result	The Tool has the name and inputs provided by the actor.

9.1.4.2. Test-case identifier 4.2

Test Case ID	HS-TC-04-2
Test Case Description	Actor can unregister a Tool
Pre-Conditions	Actor must have an activated, authenticated account and Tool Registered.
Test Steps	- Check that actor is able to delete his Tool; - Check that Tool get deleted or not.
Test Data	Registered Tool
Expected Result	Correct unregistered Tool.
Post Condition	Delete a Tool from database
Actual Result	The Tool has unregistered and deleted from the database

9.2. Test Plan for Use Case 1: The ICT Gateway

Regarding UC1, the main goal is analyzing the behavior of the ICT GW from the security perspective, improving its trustworthiness and making it resilient against attacks and failures. This is particularly important, not only from the ICT GW point of view, which is the software that can be used for validating some of the BIECO solutions and tools, but also to avoid the propagation of vulnerabilities to the smart grid and the other systems interconnected with the ICT GW itself.

In BIECO, the goals related to the ITC GW are:

- UC1_G1 - detecting vulnerabilities that might exist in the software, and determine if a possible vulnerability of the gateway could propagate to other software components of the smart grid (WP3);
- UC1_G2 - performing self-checks that allow to detect residual vulnerabilities, software and hardware failures; using simulation tools that enable a virtual evaluation when an adversary is influencing clock synchronization (WP4);
- UC1_G3 - auditing and monitoring the integration with other third-party systems and components, as well as the correct runtime behavior of the gateway and its subsystems (WP5);
- UC1_G4 - performing a risk assessment and threat modelling of the status of the system, taking into account also how a vulnerability in the ICT GW could impact the smart grid (WP6);
- UC1_G5 - obtaining guarantees that certify the security of the software (WP7).

The tools involved in this use case:

Tool	How it is involved

ResilBlockly (RES)	Modelling of the ICT GW in ResilBlockly;
	Specification of MUD-compliant communication rules for the ICT GW;
	Model-based Risk Assessment of the ICT GW;

Vulnerability Detection tool (GRAD)	Vulnerability Detection in the ICT GW
Vulnerability Propagation tool (GRAD)	Vulnerability Propagation within the ICT GW
Exploitability forecasting tool (GRAD)	Vulnerability Exploitability Forecasting in the ICT GW
SafeTBox (IESE)	Determine mitigations for the ICT GW model
Periodic self-checking of SW failures tool (RES)	Introduction of a self-checking mechanism into the ICT GW

9.2.1. Test scenario identifier 1

Test Scenario ID	UC1-TS-01
Test Scenario Name	Vulnerability Detection in the ICT GW
Test Case Description	Detection and identification of any existing vulnerability in the source code of the ICT GW
Actors	ICT GW provider (RES), Tool Developer (GRAD)
Pre-Conditions	The vulnerability detection tool is installed or runs in a RES server where the ICT GW is deployed. ICT GW source code language is compatible with detection tool.
Post-Condition	All vulnerabilities are identified, the result is not ambiguous and correctly interpreted
Associated goal	UC1_G1
Associated Requirement	UC1_FR1

9.2.1.1. Test-case-identifier 1.1

Test Case ID	UC1-TC-01-1
Test Case Description	Detect vulnerabilities to all applicable attack tests envisioned in the relevant UC1 scenarios
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	The files to be tested
Expected Result	The vulnerabilities are output
Post Condition	The run was all the way to the end
Actual Result	

9.2.2. Test scenario identifier 2

Test Scenario ID	UC1-TS-02
Test Scenario Name	Vulnerability Propagation within the ICT GW
Test Case Description	Determine the propagation of an identified vulnerability in the source code of the ICT GW
Actors	ICT GW provider (RES), Tool Developer (GRAD)
Pre-Conditions	The vulnerability propagation tool is installed or runs in a RES server where the ICT GW is deployed. ICT GW source code language is compatible with propagation tool.
Post-Condition	The propagation of the vulnerability in the source code is determined, and the result is not ambiguous and correctly interpreted
Associated goal	UC1_G1
Associated Requirement	UC1_FR2

9.2.2.1. Test-case-identifier 2.1

Test Case ID	UC1-TC-02-1
Test Case Description	Study vulnerability propagation (e.g., paths and possible level of risk) among the ICT GW
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	The propagation graph is shown
Expected Result	The vulnerabilities are output
Post Condition	The run was all the way to the end
Actual Result	-

9.2.3. Test scenario identifier 3

Test Scenario ID	UC1-TS-03
Test Scenario Name	Vulnerability Exploitability Forecasting in the ICT GW
Test Case Description	Forecasting the exploitability of an identified vulnerability in the source code of the ICT GW
Actors	ICT GW provider (RES), Tool Developer (GRAD)
Pre-Conditions	The exploitability forecasting tool is installed or runs in a RES server where the ICT GW is deployed. ICT GW source code language is compatible with detection tool.
Post-Condition	The exploitability of an identified vulnerability is predicted, the result is not ambiguous and correctly interpreted
Associated goal	UC1_G1
Associated Requirement	-

9.2.3.1. Test-case-identifier 3.1

Test Case ID	UC1-TC-03-1
Test Case Description	Study vulnerability Exploitability Forecasting among the ICT GW
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	The Exploitability Forecasting is shown
Expected Result	The vulnerabilities are output
Post Condition	The run was all the way to the end
Actual Result	-

9.2.4. Test scenario identifier 4

Test Scenario ID	UC1-TS-04
Test Scenario Name	Modelling of the ICT GW in ResilBlockly
Test Case Description	Modelling of the ICT GW and its Smart Grid Ecosystem with ResilBlockly Model Designer
Actors	ResilBlockly end-user (Model Designer User)
Pre-Conditions	The information about the ICT GW system architecture is available and sufficiently detailed for modelling. The end-user is familiar with the Tool (e.g., has read the user guide). The profile used for modelling exists.
Post-Condition	The end-user creates a model of the ICT GW (and eventually of the Smart Grid Ecosystem surrounding it).
Associated goal	UC1_G4
Associated Requirement	UC1_FR5

9.2.4.1. Test-case-identifier 4.1

Test Case ID	UC1-TC-04-1
Test Case Description	Modelling of the ICT GW in ResilBlockly
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	-
Expected Result	The model of ICT GW in ResilBlockly
Post Condition	The run was all the way to the end
Actual Result	-

9.2.5. Test scenario identifier 5

Test Scenario ID	UC1-TS-05
Test Scenario Name	Specification of MUD-compliant communication rules for the ICT GW
Test Case Description	Specification of extended MUD-compliant communication rules for the ICT GW components/interfaces
Actors	ResilBlockly end-user (Model Designer User), Security Expert
Pre-Conditions	The model of the ICT GW has been created or imported within ResilBlockly. The structure of the extended MUD is available.
Post-Condition	The model of the ICT GW is provided with communication rules, compliant with an extended MUD model, specifying the behaviour of its components/interfaces.
Associated goal	UC1_G4
Associated Requirement	UC1_FR5

9.2.5.1. Test-case-identifier 5.1

Test Case ID	UC1-TC-05-1
Test Case Description	Specification of MUD-compliant communication rules for the ICT GW
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	-
Expected Result	The specification of MUD-compliant communication rules for the ICT GW
Post Condition	The run was all the way to the end
Actual Result	-

9.2.6. Test scenario identifier 6

Test Scenario ID	UC1-TS-06
Test Scenario Name	Model-based Risk Assessment of the ICT GW in ResilBlockly
Test Case Description	The already modelled ICT GW is analysed leveraging the risk assessment features of ResilBlockly
Actors	ResilBlockly end-user (Model Designer User), Security Expert
Pre-Conditions	The model of the ICT GW has been created or imported within ResilBlockly.
Post-Condition	The model is enriched with weaknesses and vulnerabilities, and for each of them a risk assessment is conducted.
Associated goal	UC1_G4
Associated Requirement	UC1_FR5, UC1_FR6

9.2.6.1. Test-case-identifier 6.1

Test Case ID	UC1-TC-06-1
Test Case Description	Model-based Risk Assessment of the ICT GW in ResilBlockly
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	-
Expected Result	The model-based Risk Assessment of the ICT GW in ResilBlockly
Post Condition	The run was all the way to the end
Actual Result	-

9.2.7. Test scenario identifier 7

Test Scenario ID	UC1-TS-07
Test Scenario Name	Determine mitigations for the ICT GW model
Test Case Description	The model of the ICT GW is given as input to SafeTbox and complemented with mitigations
Actors	SafeTbox user, Security Expert
Pre-Conditions	The model of the ICT GW, together with the risk assessment results, is imported from ResilBlockly into SafeTbox
Post-Condition	SafeTbox provides mitigations for the identified vulnerabilities, weaknesses or attack scenarios
Associated goal	UC1_G4
Associated Requirement	UC1_FR7

9.2.7.1. Test-case-identifier 7.1

Test Case ID	UC1-TC-07-1
Test Case Description	Determine mitigations for the ICT GW model
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	-
Expected Result	The mitigations for the ICT GW model
Post Condition	The run was all the way to the end
Actual Result	-

9.2.8. Test scenario identifier 8

Test Scenario ID	UC1-TS-08
Test Scenario Name	Introduction of a self-checking mechanism into the ICT GW
Test Case Description	The ICT GW is provided with a self-checking mechanism for detecting residual software vulnerabilities
Actors	ICT GW use case owner, tool developer
Pre-Conditions	The specific self-checking mechanism has been designed and developed
Post-Condition	The ICT GW is provided with the mechanism and is therefore capable of performing self-checks for residual software vulnerabilities
Associated goal	UC1_G2
Associated Requirement	UC1_FR3

9.2.8.1. Test-case-identifier 8.1

Test Case ID	UC1-TC-08-1
Test Case Description	Introduction of a self-checking mechanism into the ICT GW
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	-
Expected Result	The introduction of a self-checking mechanism into the ICT GW
Post Condition	The run was all the way to the end
Actual Result	-

9.3. Test Plan for Use Case 2: AI Investments

The main goal to achieve in BIECO from the AI Investments use case perspective is analyzing the behavior of the AI Investments application (AII application) from the security perspective and making it resilient against attacks and failures. This is particularly important, not only from the AII point of view, which is the software that can be used for validating BIECO, but also to avoid the propagation of vulnerabilities to the stock brokers and the other systems interconnected with the AII application.
In BIECO the AII application will be used to:

- UC2_G1 - detect software vulnerabilities that might exist in the software, and determine how a possible vulnerability of the application could propagate to the stock brokers (WP3);
- UC2_G2 - perform self-checks that allow to detect residual vulnerabilities, software and hardware failures; using simulation tools that enable a virtual evaluation when an adversary is influencing clock synchronization (WP4);
- UC2_G3 - audit and monitor the integration with other third-party systems and components, as well as the correct runtime behavior of the application and its subsystems (WP5)
- UC2_G4 - perform a risk assessment and threat modelling of the status of the system, taking into account also how a vulnerability in the AII application could impact the stock brokers and financial assets (WP6);
- UC2_G5 - obtain guarantees that certify the adherence of the software to its expected behavior (WP7).

The tools involved in this use case:

Tool	How it is involved

ResilBlockly (RES)	Modelling of the AI Investments application in ResilBlockly;
	Model-based Risk Assessment of the AI Investments application;

Vulnerability Detection tool (GRAD)	Vulnerability detection in the AI Investments application;
Vulnerability Propagation tool (GRAD)	Vulnerability propagation in the AI Investments application;
SafeTBox (IESE)	Determine mitigations for the AI Investments application model;
Periodic self-checking of SW failures tool (RES)	Introduction of a self-checking mechanism into the AI Investments application.

9.3.1. Test scenario identifier 1

Test Scenario ID	UC2-TS-01
Test Scenario Name	Vulnerability detection in the source code of AI Investments application
Test Case Description	Detection and identification of any existing vulnerability in the source code of the AI Investments application
Actors	AI Investments application developer (7b), Tool Developer (GRAD)
Pre-Conditions	The vulnerability detection tool is installed or runs in a 7b server where the AI Investments application is deployed. AI Investments application source code language is compatible with detection tool.
Post-Condition	All vulnerabilities are identified, the result is not ambiguous and correctly interpreted
Associated goal	UC2_G1
Associated Requirement	UC2_FR1

9.3.1.1. Test-case-identifier 1.1

Test Case ID	UC2-TC-01-1
Test Case Description	Detect vulnerabilities to all applicable attack tests envisioned in the relevant UC2 scenarios
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	The files to be tested
Expected Result	The vulnerabilities are output
Post Condition	The run was all the way to the end
Actual Result	-

9.3.2. Test scenario identifier 2

Test Scenario ID	UC2-TS-02
Test Scenario Name	Vulnerability propagation among the AI Investments application components.
Test Case Description	Determine the propagation of an identified vulnerability in the source code of the components in AI Investments application.
Actors	AI Investments application developer (7b), Tool Developer (GRAD)
Pre-Conditions	The vulnerability propagation tool is installed or runs in a 7b server where the AI Investments application is deployed.
Post-Condition	The propagation of the vulnerability in the source code is determined, and the result is not ambiguous and correctly interpreted
Associated goal	UC2_G1
Associated Requirement	UC2_FR2

9.3.2.1. Test-case-identifier 2.1

Test Case ID	UC2-TC-02-1
Test Case Description	Study vulnerability propagation (e.g., paths and possible level of risk) among the AI Investments application components.
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	The propagation graph is shown
Expected Result	The vulnerabilities are output
Post Condition	The run was all the way to the end
Actual Result	-

9.3.3. Test scenario identifier 3

Test Scenario ID	UC2-TS-03
Test Scenario Name	Modelling of the AI Investments application in ResilBlockly
Test Case Description	Modelling of the AI Investments application with ResilBlockly Model Designer
Actors	ResilBlockly end-user (Model Designer User)
Pre-Conditions	The information about the AI Investments application system architecture is available and sufficiently detailed for modelling. The end-user is familiar with the Tool (e.g., has read the user guide). The profile used for modelling exists.
Post-Condition	The end-user creates a model of the AI Investments application.
Associated goal	UC2_G4
Associated Requirement	UC2_FR5

9.3.3.1. Test-case-identifier 3.1

Test Case ID	UC2-TC-03-1
Test Case Description	Modelling of the AI Investments application in ResilBlockly
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	AI Investments application
Expected Result	The modelled application in ResilBlockly was created successfully
Post Condition	The run was all the way to the end
Actual Result	-

9.3.4. Test scenario identifier 4

Test Scenario ID	UC2-TS-04
Test Scenario Name	Model-based Risk Assessment of the AI Investments application in ResilBlockly
Test Case Description	The already modelled AI Investments application is analysed leveraging the risk assessment features of ResilBlockly
Actors	ResilBlockly end-user (Model Designer User), Security Expert
Pre-Conditions	The model of the AI Investments application has been created or imported within ResilBlockly.
Post-Condition	The model is enriched with weaknesses and vulnerabilities, and for each of them a risk assessment is conducted.
Associated goal	UC2_G4
Associated Requirement	UC2_FR5, UC2_FR6

9.3.4.1. Test-case-identifier 4.1

Test Case ID	UC2-TC-04-1
Test Case Description	Model-based Risk Assessment of the AI Investments application in ResilBlockly
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	AI Investments application
Expected Result	The model-based Risk Assessment of the AI Investments application in ResilBlockly
Post Condition	The run was all the way to the end
Actual Result	-

9.3.5. Test scenario identifier 5

Test Scenario ID	UC2-TS-05
Test Scenario Name	Determine mitigations for the AI Investments application
Test Case Description	The model of the AI Investments application is given as input to SafeTbox and complemented with mitigations
Actors	SafeTbox user, Security Expert
Pre-Conditions	The model of the AI Investments application together with the risk assessment results, is imported from ResilBlockly into SafeTbox
Post-Condition	SafeTbox provides mitigations for the identified vulnerabilities, weaknesses or attack scenarios
Associated goal	UC2_G4
Associated Requirement	UC2_FR7

9.3.5.1. Test-case-identifier 5.1

Test Case ID	UC2-TC-05-1
Test Case Description	Determine mitigations for the AI Investments application
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	AI Investments application
Expected Result	Determination of mitigations for the AI Investments application
Post Condition	The run was all the way to the end
Actual Result	-

9.3.6. Test scenario identifier 6

Test Scenario ID	UC2-TS-06
Test Scenario Name	Introduction of a self-checking mechanism into the AI Investments application components.
Test Case Description	The AI Investments application components are provided with a self-checking mechanism for detecting residual software vulnerabilities
Actors	AI Investments application use case owner (7b), tool developer
Pre-Conditions	The specific self-checking mechanism has been designed and developed
Post-Condition	The AI Investments application provided with the built-in feature capable of performing self-checks for vulnerabilities in source code or running application/component.
Associated goal	UC2_G2
Associated Requirement	UC2_FR3

9.3.6.1. Test-case-identifier 6.1

Test Case ID	UC2-TC-06-1
Test Case Description	Introduction of a self-checking mechanism into the AI Investments application components.
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	AI Investments application
Expected Result	The potential attacks and failures are displayed
Post Condition	The run was all the way to the end
Actual Result	-

9.4. Test Plan for Use Case 3: Smart Microfactory and FOTA

For UC3, the FOTA implementation (according to the UPTANE guidelines) has been chosen as a representative and particularly relevant case of security for communication in the automotive field, with the aim to extend it to the whole industrial production, management and maintenance environment of the microfactory.
Likely, due to restrictions on IP in the automotive industrial sector and company policy, a Hardware-in-the-loop approach will be adopted and a realistic cyberattack situation will be evaluated and possible mitigation strategies and countermeasures will be studied and proposed.
For the UPTANE/FOTA system, particular goals have been derived from the general ones of the project (see section 4.7.1). The UC3 specific goals are given hereafter:

- UC3_G1 (WP3) – analyse the system to find any possible vulnerability or weakness of any HW node and SW component in the platform. Propagation of anomalous behaviour from one element to another in the networked system will also be studied.
- UC3_G2 (WP4) – test the implemented protocols to verify signature, authentication, integrity etc. on metadata and firmware image, thus providing a vulnerability assessment.
- UC3_G3 (WP5) – audit and monitor the network data traffic to detect anomalies or issues which can lead to critical situation with the rise of new vulnerabilities and provide the tools to enhance resilience.
- UC3_G4 (WP6) – carry out the risk analysis and provide mitigation strategies based on previous tests results; eventually, this will allow threat modelling as well.
- UC3_G5 (WP7) – gather test results to support the definition of the security level implemented in the system and its certification.

9.4.1. Test scenario 1

Test Scenario ID	UC3-TS-01
Test Scenario Name	Detect vulnerabilities to all applicable attack tests envisioned in the relevant UC3 scenarios
Test Case Description	Detect vulnerabilities to all applicable attack tests envisioned in the relevant UC3 scenarios
Actors	User
Pre-Conditions	User authenticated
Post-Condition	
Associated goal	UC3_G1
Associated Requirement	UC3_FR1

9.4.1.1. Test-case-identifier 1.1

Test Case ID	UC3-TC-01-1
Test Case Description	Detect vulnerabilities to all applicable attack tests envisioned in the relevant UC3 scenarios
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	The files to be tested
Expected Result	The vulnerabilities are output
Post Condition	The run was all the way to the end
Actual Result	

9.4.2. Test scenario 2

Test Scenario ID	UC3-TS-02
Test Scenario Name	Study vulnerability propagation (e.g., paths and possible level of risk) among HW nodes and SW components of the UPTANE-FOTA platform
Test Case Description	Study vulnerability propagation (e.g., paths and possible level of risk) among HW nodes and SW components of the UPTANE-FOTA platform
Actors	User
Pre-Conditions	User authenticated
Post-Condition	-
Associated goal	UC3_G1
Associated Requirement	UC3_FR2

9.4.2.1. Test-case-identifier 2.1

Test Case ID	UC3-TC-02-1
Test Case Description	Study vulnerability propagation (e.g., paths and possible level of risk) among HW nodes and SW components of the UPTANE-FOTA platform
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	The propagation graph is shown
Expected Result	The vulnerabilities are output
Post Condition	The run was all the way to the end
Actual Result	

9.4.3. Test scenario 3

Test Scenario ID	UC3-TS-03
Test Scenario Name	Verify cross-checking protocols that allow to counter potential attacks and failures during the remote FW update and detect behavioural anomalies which can open breaches to further vulnerabilities or failures
Test Case Description	Verify cross-checking protocols that allow to counter potential attacks and failures during the remote FW update and detect behavioural anomalies which can open breaches to further vulnerabilities or failures
Actors	User
Pre-Conditions	User authenticated
Post-Condition	
Associated goal	UC3_G3
Associated Requirement	UC3_FR3

9.4.3.1. Test-case-identifier 3.1

Test Case ID	UC3-TC-03-1
Test Case Description	Verify cross-checking protocols that allow to counter potential attacks and failures during the remote FW update and detect behavioural anomalies which can open breaches to further vulnerabilities or failures
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	
Expected Result	The potential attacks and failures are displayed
Post Condition	The run was all the way to the end
Actual Result	

9.4.4. Test scenario 4

Test Scenario ID	UC3-TS-04
Test Scenario Name	Audit/Monitor the behaviour of the networked communications with ongoing attack attempts or failures, in particular with those described in the scenarios.
Test Case Description	Audit/Monitor the behaviour of the networked communications with ongoing attack attempts or failures, in particular with those described in the scenarios.
Actors	User
Pre-Conditions	User authenticated
Post-Condition	
Associated goal	G4
Associated Requirement	UC3_FR4

9.4.4.1. Test-case-identifier 4.1

Test Case ID	UC3-TC-04-1
Test Case Description	Audit/Monitor the behaviour of the networked communications with ongoing attack attempts or failures, in particular with those described in the scenarios.
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	
Expected Result	The behaviour of the networked communications with ongoing attack attempts or failures
Post Condition	The run was all the way to the end
Actual Result	

9.4.5. Test scenario 5

Test Scenario ID	UC3-TS-05
Test Scenario Name	Collect evidences of possible weaknesses and vulnerabilities to support the evaluation of the overall security of the system
Test Case Description	Collect evidences of possible weaknesses and vulnerabilities to support the evaluation of the overall security of the system
Actors	User
Pre-Conditions	User authenticated
Post-Condition	
Associated goal	G5
Associated Requirement	UC3_FR5

9.4.5.1. Test-case-identifier 5.1.

Test Case ID	UC3-TC-05-1
Test Case Description	Collect evidences of possible weaknesses and vulnerabilities to support the evaluation of the overall security of the system
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	-
Expected Result	The evidences of possible weaknesses and vulnerabilities to support the evaluation of the overall security of the system
Post Condition	The run was all the way to the end
Actual Result	

9.4.6. Test scenario 6

Test Scenario ID	UC3-TS-06
Test Scenario Name	Assess weakness and vulnerability risks of relevant attacks, with reference to those indicated in UC3 scenarios.
Test Case Description	Assess weakness and vulnerability risks of relevant attacks, with reference to those indicated in UC3 scenarios.
Actors	User
Pre-Conditions	User authenticated
Post-Condition	
Associated goal	G5
Associated Requirement	UC3_FR6

9.4.6.1. Test-case-identifier 6.1

Test Case ID	UC3-TC-06-1
Test Case Description	Assess weakness and vulnerability risks of relevant attacks, with reference to those indicated in UC3 scenarios.
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	
Expected Result	The weakness and vulnerability risks of relevant attacks
Post Condition	The run was all the way to the end
Actual Result	

9.4.7. Test scenario 7

Test Scenario ID	UC3 TS-07
Test Scenario Name	Find possible mitigation actions including those needed to counter the attacks described in UC3 scenarios.
Test Case Description	Find possible mitigation actions including those needed to counter the attacks described in UC3 scenarios.
Actors	User
Pre-Conditions	User authenticated
Post-Condition	
Associated goal	G5
Associated Requirement	UC3_FR7

9.4.7.1. Test-case-identifier 7.1

Test Case ID	UC3-TC-07-1
Test Case Description	Find possible mitigation actions including those needed to counter the attacks described in UC3 scenarios.
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	
Expected Result	The possible mitigation actions
Post Condition	The run was all the way to the end
Actual Result	

9.4.8. Test scenario 8

Test Scenario ID	UC3-TS-08
Test Scenario Name	Carry out WP7 security certification methodology and risk assessment to the UPTANE-FOTA system.
Test Case Description	Carry out WP7 security certification methodology and risk assessment to the UPTANE-FOTA system.
Actors	User
Pre-Conditions	User authenticated
Post-Condition	
Associated goal	G6
Associated Requirement	UC3_FR8

9.4.8.1. Test-case-identifier 8.1

Test Case ID	UC3-TC-08-1
Test Case Description	Carry out WP7 security certification methodology and risk assessment to the UPTANE-FOTA system.
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	
Expected Result	The UPTANE-FOTA system IS CERTIFIED ACCORDING TO THE METHODOLOGY
Post Condition	The run was all the way to the end
Actual Result	

9.4.9. Test scenario 9

Test Scenario ID	UC3-TS-09
Test Scenario Name	After implementing mitigation actions, evaluate and certify UC3 security level
Test Case Description	After implementing mitigation actions, evaluate and certify UC3 security level
Actors	User
Pre-Conditions	User authenticated
Post-Condition	
Associated goal	G6
Associated Requirement	UC3_FR9

9.4.9.1. Test-case-identifier 9.1

Test Case ID	UC3-TC-09-1
Test Case Description	After implementing mitigation actions, evaluate and certify UC3 security level
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	
Expected Result	UC3 security level is certified
Post Condition	The run was all the way to the end
Actual Result	

9.5. Test Plan for Use Case 4: Autonomous Navigation

In UC4, the main goal is analysing the behaviour of an autonomous navigation system. Concretely, BIECO will apply its tools and methodology in order to find possible safety and security vulnerabilities in the local planning of mobile robots. Improving trustworthiness and resilience against attacks and failures.

In BIECO, the goals related to the ITC GW are:

- **UC4_G1** (WP3) – analyse the system to find any possible vulnerability or weakness of any SW component in the controlled environment. Chain propagation of anomalous behaviour from one element to another in the ROS based networked system will also be studied.
- **UC4_G2** (WP4) – test the implemented protocols to verify the behaviour of the local planner module to ensure safe behaviour.

- **UC4_G3** (WP5) – audit and monitor each entity (robots and stations) in presence of attempts of attacks (I.e., the ones identified in the scenarios) and the consequences for other third-party systems and components.
- **UC4_G4** (WP6) – carry out the risk analysis and provide mitigation strategies based on previous tests results; eventually, this will allow threat modelling as well.
- **UC4_G5** (WP7) – gather test results to support the definition of the security level implemented in the system and its certification.

The tools involved in this use case:

Tool	How it is involved
Vulnerability Detection tool (GRAD)	Vulnerability Detection in the ROS based navigation system of UC4
Vulnerability Propagation tool (GRAD)	Vulnerability Propagation within the ROS network of UC4
SafeTBox (IESE)	Determine mitigations for the UC4 model
TOOLNAME (IESE)	Perform continuous behaviour analysis that can create breaches and vulnerabilities in the navigation software
Periodic self-checking of SW failures tool (RES)	Introduction of a self-checking mechanism that confirms all ROS SW modules.

9.5.1. Test scenario identifier 1

Test Scenario ID	UC4-TS-01
Test Scenario Name	Vulnerability Detection in the ROS network of UC4
Test Case Description	Detection and identification of any existing vulnerability in the source code of the navigation system in UC4
Actors	UC4 provider (UNI), Tool Developer (GRAD)
Pre-Conditions	The vulnerability detection tool is installed or runs in a controlled environment where UC4 is deployed. ROS SW code source code language is compatible with detection tool.
Post-Condition	All vulnerabilities are identified, the result is not ambiguous and correctly interpreted
Associated goal	UC4_G1
Associated Requirement	UC4_FR1

9.5.1.1. Test-case-identifier 1.1

Test Case ID	UC4-TC-01-1
Test Case Description	Detect vulnerabilities to all applicable attack tests envisioned in the relevant UC4 scenarios
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	The files to be tested
Expected Result	The vulnerabilities are output
Post Condition	The run was all the way to the end
Actual Result	

9.5.2. Test scenario identifier 2

Test Scenario ID	UC4-TS-02
Test Scenario Name	Vulnerability Propagation within the ROS network of UC4
Test Case Description	Determine the propagation of an identified vulnerability in the source code of the ROS SW modules and its impact on the complete ROS network
Actors	UC4 provider (UNI), Tool Developer (GRAD)
Pre-Conditions	The vulnerability detection tool is installed or runs in a controlled environment where UC4 is deployed. ROS SW code source code language is compatible with detection tool.
Post-Condition	The propagation of the vulnerability in the source code is determined, and the result is not ambiguous and correctly interpreted
Associated goal	UC4_G1
Associated Requirement	UC4_FR2

9.5.2.1. Test-case-identifier 2.1

Test Case ID	UC4-TC-02-1
Test Case Description	Study vulnerability propagation (e.g., paths and possible level of risk) within the ROS network of UC4
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	The propagation graph is shown
Expected Result	The vulnerabilities are output
Post Condition	The run was all the way to the end
Actual Result	

9.5.3. Test scenario identifier 3

Test Scenario ID	UC4-TS-03
Test Scenario Name	Determine mitigations for the UC4 model
Test Case Description	The model of the UC4 is given as input to SafeTbox and complemented with mitigations
Actors	SafeTbox user, Security Expert
Pre-Conditions	The model of the UC4, together with the risk assessment results, is imported from ResilBlockly into SafeTbox
Post-Condition	SafeTbox provides mitigations for the identified vulnerabilities, weaknesses or attack scenarios
Associated goal	UC4_G4
Associated Requirement	UC4_FR6

9.5.3.1. Test-case-identifier 3.1

Test Case ID	UC4-TC-03-1
Test Case Description	Determine mitigations for the UC4 model
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	
Expected Result	The mitigations for the UC4 model
Post Condition	The run was all the way to the end
Actual Result	

9.5.4. Test scenario identifier 4

Test Scenario ID	UC4-TS-04
Test Scenario Name	Perform continuous behaviour analysis
Test Case Description	Use a digital twin detect anomalous behaviour in the navigation software
Actors	Test Case provider (UNI), Tool developer (IESE)
Pre-Conditions	A digital twin of the navigation software is created
Post-Condition	Tool provides continuous behaviour detection that will mitigate possible safety and security issues.
Associated goal	UC4_G3
Associated Requirement	UC4_FR4

9.5.4.1. Test-case-identifier 4.1

Test Case ID	UC4-TC-04-1
Test Case Description	Perform continuous behaviour analysis
Pre-Conditions	User authenticated
Test Steps	The user loads/creates a template and then it runs it
Test Data	
Expected Result	The analysis of continuous behaviour
Post Condition	The run was all the way to the end
Actual Result	

9.6. Test Plan for Data collection and pre-processing tool

This tool makes use in BIECO of the following tools:

Data collection tool has two parts:

1. **Web Application**
    a. Client Application
    b. Admin Application
2. **REST API**

REST API tests has been done using Postman and full documentation related to the syntax and examples are presented in Document *D3.2 - Dataset with software vulnerabilities - final version.docx -3.3 API Specification*. Example of using DCT API using Python and cURL are documented in *D3.2 - Dataset with software vulnerabilities - final version.docx – Annex A. API Code Snippets.*

Tool	How it is involved
jUnit	Provides tools, classes and methods to ease the task of performing unit tests. For each component in part a unit test is written.
Postman	Platform for testing the REST APIs.
Selenium	Provides the possibility to run automated tests. Used in the integration phase to test integration between components.

9.6.1 Test scenario identifier 1

Test Scenario ID	UTC-TS-01
Test Scenario Name	Manual usage of Data Collection and pre-processing tool for Client Application
Test Case Description	Testing the Data Collection and pre-processing tool from a user's point of view
Actors	End-User
Pre-Conditions	An activated and authenticated user
Post-Condition	Actors will have access to Data Collection and pre-processing tool
Associated goal	-
Associated Requirement	-

9.6.1.1. Test-case-identifier 1.1

Test Case ID	UTC-TC-01-1
Test Case Description	**Client Application** HOME functionality
Pre-Conditions	-
Test Steps	Users access the client webpage: – Click on Home
Test Data	-
Expected Result	Should display a statistic of the CVSS score distribution for all vulnerabilities (number and percentage) as well as a chart with the score distribution.
Post Condition	-
Actual Result	

9.6.1.2. Test-case-identifier 1.2

Test Case ID	UTC-TC-01-2
Test Case Description	**Client Application** Products- Vendor Search functionality
Pre-Conditions	-
Test Steps	Users access the client webpage: – Click on Public data – Products- Vendor Search
Test Data	-
Expected Result	Should display the list of vendors and the number of products per each vendor and vulnerabilities added. It should also offer the possibility to search based on a vendor name.
Post Condition	-
Actual Result	

9.6.1.3. Test-case-identifier 1.3

Test Case ID	UTC-TC-01-3
Test Case Description	**Client Application** Products Search functionality
Pre-Conditions	-
Test Steps	Users access the client webpage: – Click on Public data – Products- Version Search
Test Data	-
Expected Result	Should display the list of products, the vendor, the number of the CVE Entries and the Product Type. Should have a search area that offers the possibility to search for a specific product.
Post Condition	-
Actual Result	

9.6.1.4. Test-case-identifier 1.4

Test Case ID	UTC-TC-01-4
Test Case Description	Client Application Vulnerabilities by Date functionality
Pre-Conditions	-
Test Steps	Users access the client webpage: – Click on Public Data – Vulnerabilities – By Date
Test Data	-
Expected Result	Should display a statistic of all the vulnerabilities per year and a statistic chart per all years and a table with the number of vulnerabilities for each year is available, and if clicking on a specific month from a specific year, the list of vulnerabilities will be displayed for that year and month with the possibility to search based on all CVE criteria.
Post Condition	-
Actual Result	

9.6.1.5. Test-case-identifier 1.5

Test Case ID	UTC-TC-01-5
Test Case Description	**Client Application** Vulnerabilities – Weaknesses functionality
Pre-Conditions	-
Test Steps	Users access the client webpage: – Click on Public Data – Vulnerabilities – Weaknesses
Test Data	-
Expected Result	Should display the list of CWE and the corresponding fields: CWE Id, Name, Description, Status. Also, it offers the possibility to do a search based on these fields in the CWE list.
Post Condition	-
Actual Result	

9.6.1.6. Test-case-identifier 1.6

Test Case ID	UTC-TC-01-6
Test Case Description	**Client Application** Public Data – Exploits functionality
Pre-Conditions	-
Test Steps	Users access the client webpage: – Click on Public Data – Exploits
Test Data	-
Expected Result	Should display the records from the Exploits database. The following fields should be displayed, and it should offer the possibility to do an advanced search based on them: Name, Type, CVE, Platform, Author and Date.
Post Condition	-
Actual Result	

9.6.1.7. Test-case-identifier 1.7

Test Case ID	UTC-TC-01-7
Test Case Description	**Client Application** Public Data – MUD Files functionality
Pre-Conditions	-
Test Steps	Users access the client webpage: – Click on Public Data – MUD Files
Test Data	-
Expected Result	Should display the records from the MUD database. The following fields should be displayed, and it should offer the possibility to do an advanced search: Product name, Manufacturer, ZIP file, MUD file, Signature file and Date. Also, it should offer the possibility to download the zip file, JSON MUD file and signature file.
Post Condition	-
Actual Result	

9.6.1.8. Test-case-identifier 1.8

Test Case ID	UTC-TC-01-8
Test Case Description	**Client Application** Internal Data – Components and Dependencies functionality
Pre-Conditions	-
Test Steps	Users access the client webpage: – Click on Internal Data – Profile Information - Components and Dependencies
Test Data	-
Expected Result	Should display the Components and Dependencies records. The page should display the name, type and details for each component. The possibility to do a search based on all these fields should also be available.
Post Condition	-
Actual Result	

9.6.1.9. Test-case-identifier 1.9

Test Case ID	UTC-TC-01-9
Test Case Description	**Client Application** Public Data – Software bugs functionality
Pre-Conditions	-
Test Steps	Users access the client webpage: – Click on Internal Data – Profile Information – Software bugs
Test Data	-
Expected Result	Should displays the list of bugs from the use cases. The following fields should be displayed, and the view should offer the possibility to do an advanced search based on all these fields: Key, Summary, Issue Type, Status, Priority, Resolution, Assignee, Reporter, Created.
Post Condition	-
Actual Result	

9.6.2 Test scenario identifier 2

Test Scenario ID	UTC-TS-02
Test Scenario Name	Manual usage of Data Collection and pre-processing tool for Admin Application
Test Case Description	Testing the Data Collection and pre-processing tool from an admin's usage point of view
Actors	Administrator
Pre-Conditions	An activated and authenticated account
Post-Condition	Actors will have access to Data Collection and pre-processing tool
Associated goal	-
Associated Requirement	-

9.6.2.1 Test-case-identifier 2.1

Test Case ID	UTC-TC-02-1
Test Case Description	**Admin Application** - User Authentication functionality
Pre-Conditions	Admin user must have an activated account.
Test Steps	- Introduction of valid data into the username and password fields. - Introduction of invalid data into the username and password fields. - Introduction of empty data for either username or password
Test Data	Both valid and invalid faked user information.
Expected Result	- Platform must allow user to login only if the username and password entered are valid and actor has an activated account. - When the required fields are not entered correctly the user should not be able to login and an error message should be displayed
Post Condition	Admin is successfully logged in the platform.
Actual Result	The user has successfully logged in or not.

9.6.2.2 Test-case-identifier 2.2

Test Case ID	UTC-TC-02-2
Test Case Description	**Admin Application -** administrate the CVE/CPE/CWE data
Pre-Conditions	Admin user must be correctly authenticated in the platform and access the CVE/CPE/CWE Page
Test Steps	Click on CVE/CPE/CWE Page
Test Data	–
Expected Result	To be able to manage all the CVE/CPE/CWE records and perform CRUD operations on them. It also provides the possibility to do an advanced search for each field in part. The functionality actions menu should be visible in the top area of the data management table, and the associated record actions should be displayed in the last column.
Post Condition	Admin should successfully manage all the CVE/CPE/CWE records.
Actual Result	The admin user has successfully performed the desired action on the CVE/CPE/CWE records.

9.6.2.3. Test-case-identifier 2.3

Test Case ID	UTC-TC-02-3
Test Case Description	**Admin Application -** administrate the Exploits/MUD files/ Software bugs data
Pre-Conditions	Admin user must be correctly authenticated in the platform and access the Exploits/MUD files/Software bugs/Components Page
Test Steps	Click on Exploits/MUD files/Software bugs/Components Page
Test Data	–
Expected Result	To be able to manage all the Exploits/MUD files/Software bugs/Components records and perform CRUD operations on them. It should also provide the possibility to do an advanced search for each field in part. The functionality actions menu should be visible in the top area of the data management table, and the associated record actions should be displayed in the last column.
Post Condition	Admin should successfully manage all the Exploits/MUD files/Software/Components bugs records.
Actual Result	The admin user has successfully performed the desired action on the Exploits/MUD files/Software bugs/Components records.

9.7. Test Plan for Vulnerability Detection tool

The objective of the tool is the detection of vulnerabilities that might exist in the ICT GW (software implementation, used libraries and technologies) which may provoke the successful execution of attacks. The tool, through static analysis and Machine Learning algorithms, locates sections of the code which contain vulnerabilities or those that are more prone to contain one.

This tool makes use in BIECO of the following tools:

Tool	How it is involved
jUnit	Provides tools, classes and methods to ease the task of performing unit tests.
SonarQube	Provides static source code analysis, identifying susceptible points, such as vulnerabilities and bugs, which will be solved before proceeding to the next step of development and testing.
Safety	Checks installed dependencies for known security vulnerabilities, using a proprietary database by default.

9.7.1. Test scenario identifier 1

Test Scenario ID	GRAD-TS-01
Test Scenario Name	Checking for vulnerabilities in the source code.
Test Case Description	Checking if the source code has any vulnerability.
Actors	End-User, Tool Developer and Platform Administrator.
Pre-Conditions	The language of the source code to be examined must be compatible with the languages implemented in this tool.
Post-Condition	Actors must correctly understand the program's output.

9.7.1.1. Test-case-identifier 1.1

Test Case ID	GRAD-TC-01-1
Test Case Description	Identifying the existing vulnerabilities in the source code
Pre-Conditions	The language of the code must be compatible with those implemented in the tool.
Test Steps	Being provided with a source code with an existing vulnerability, verify if the tool is able to detect it.
Test Data	Source code of an existing vulnerability.
Expected Result	Identification of all vulnerabilities.
Post Condition	The result must be correctly interpreted by the actors.
Actual Result	

9.7.1.2. Test-case-identifier 1.2

Test Case ID	GRAD-TC-01-2
Test Case Description	If the source code has no vulnerabilities, it does not identify any.
Pre-Conditions	The language of the code must be compatible with those implemented in the tool.
Test Steps	Run the tool on a source code with no vulnerabilities and check that there are no vulnerabilities.
Test Data	Source code without any vulnerability.
Expected Result	It does not identify any vulnerability.
Post Condition	The result must be correctly interpreted by the actors.
Actual Result	

9.8. Test Plan for Vulnerability Propagation tool

The objective of the tool is to study vulnerability propagation, such as paths and possible level of risk, in the source code to the ICT GW components.

This tool makes use in BIECO of the following tools:

Tool	How it is involved
jUnit	Provides tools, classes and methods to ease the task of performing unit tests.
SonarQube	Provides static source code analysis, identifying susceptible points, such as vulnerabilities and bugs, which will be solved before proceeding to the next step of development and testing.
Safety	Checks installed dependencies for known security vulnerabilities, using a proprietary database by default.

9.8.1. Test scenario identifier 1

Test Scenario ID	GRAD-TS-02
Test Scenario Name	Detecting the propagation of an already identified vulnerability.
Test Case Description	Knowing in advance the propagation of a vulnerability in the source code, check that the tool detects the propagation of the vulnerability correctly.
Actors	End-User, Tool Developer and Platform Administrator.
Pre-Conditions	The language of the source code to be examined must be compatible with the languages implemented in this tool.
Post-Condition	Actors must correctly understand the program's output.

9.8.1.1. Test-case-identifier 1.1

Test Case ID	GRAD-TC-02-1
Test Case Description	Identifying the propagation of an existing source code vulnerability.
Pre-Conditions	The language of the code must be compatible with those implemented in the tool.
Test Steps	Obtain the source code vulnerability and its identified propagation path. Execute the tool and verify that the propagation path is the expected.
Test Data	Source code with a vulnerability and its propagation path.
Expected Result	Propagation of the vulnerability in the source code.
Post Condition	The result is correctly interpreted by the different actors.
Actual Result	

9.9. Test Plan for Exploitability forecasting tool

The objective of the tool is the forecasting of the exploitability of vulnerabilities in the ICT GW. Specifically, the tool, through Machine Learning algorithms, provides an estimate of the period of time in which a vulnerability could be exploited (e.g., within the next 12 months).
This tool makes use in BIECO of the following tools:

Tool	How it is involved
jUnit	Provides tools, classes and methods to ease the task of performing unit tests.
SonarQube	Provides static source code analysis, identifying susceptible points, such as vulnerabilities and bugs, which will be solved before proceeding to the next step of development and testing.
Safety	Checks installed dependencies for known security vulnerabilities, using a proprietary database by default.

9.9.1. Test scenario identifier 1

Test Scenario ID	GRAD-TS-03
Test Scenario Name	Forecasting exploitability of an already identified vulnerability.
Test Case Description	Having identified the vulnerability in the source code, determine its exploitability.
Actors	End-User, Tool Developer and Platform Administrator.
Pre-Conditions	The language of the source code must be compatible with the language of the tool.
Post-Condition	Actors must correctly interpret the program's output.

9.9.1.1. Test-case-identifier 1.1

Test Case ID	GRAD-TC-03-1
Test Case Description	Identification of the predicted exploitability of an existing vulnerability in the source code.
Pre-Conditions	The data entered in the model must have all the required characteristics.
Test Steps	Having the information about when a vulnerability became public and when it became exploitable, execute the tool on that vulnerability and verify that the result matches the predicted exploitability of the vulnerability.
Test Data	Source code of a vulnerability and the time between publication and exploitation of the vulnerability.
Expected Result	Anticipated exploitability of an already identified vulnerability.
Post Condition	The result must be correctly interpreted by the actors.
Actual Result	

9.10. Test Plan for Vulnerabilities forecasting tool

This tool makes use in BIECO of the following tools:

Tool	How it is involved
Postman	Platform for testing the REST APIs
Data Collection Tool	Provides information regarding the known vulnerabilities.
	Provides the use case components and dependencies.
	Provides the use cases bug history.

9.10.1. Test scenario identifier 1

Test Scenario ID	UTC-TS-03
Test Scenario Name	Vulnerabilities forecasting.
Test Case Description	Forecasting the number of vulnerabilities that will be discovered in certain time frame, for certain use case.
Actors	Tool Developer and Platform Administrator.
Pre-Conditions	The use case profile must be complete and correct.
Post-Condition	Actors must understand and correctly follow the API rules.

9.10.1.1. Test-case-identifier 1.1

Test Case ID	UTC-TC-03-1
Test Case Description	Obtaining the time evolution of the use case vulnerabilities from the Data Collection Tool or NVD.
Pre-Conditions	The use case profile is completely specified.
Test Steps	Provide the time frame and verify if the tool correctly delivers the time evolution of the number of vulnerabilities, bugs, or both.
Test Data	The vulnerability information in the Data Collection Tool or National Vulnerability Database and the use case profile.
Expected Result	The time evolution of the use case vulnerabilities, bugs or both.
Post Condition	
Actual Result	

9.10.1.2. Test-case-identifier 1.2

Test Case ID	UTC-TC-03-2
Test Case Description	Evaluate the forecasting algorithm accuracy
Pre-Conditions	The use case profile is completely specified, and there is sufficient vulnerability information in the Data Collection Tool or NVD.
Test Steps	Provide a past time frame and obtain a forecast for the number of vulnerabilities, software bugs or both. Compare the result with the available historical data.
Test Data	A fraction of the available vulnerability information in the Data Collection Tool or National Vulnerability Database.
Expected Result	Good average accuracy for the one step forecasting.
Post Condition	
Actual Result	

9.11. Test Plan for periodic self-checking of HW/SW failures tool

This tool makes use in BIECO of the following tools:

Tool	How it is involved
JUnit	Framework used for unit testing
Postman	Platform for testing the REST APIs
STL	Self-Test Libraries

9.11.1. Test scenario identifier 1

Test Scenario ID	RES-TS-01
Test Scenario Name	Periodic self-check of failures
Test Case Description	Monitoring of data stream and periodic check of failures
Actors	Tool developers, end-user
Pre-Conditions	
Post-Condition	

9.11.1.1. Test-case-identifier 1.1

Test Case ID	RES-TC-01-1
Test Case Description	Periodic self-check of Hardware failure
Pre-Conditions	A Self-test library is available
Test Steps	Relying on STL, perform test instructions of HW features and components
Test Data	
Expected Result	Boolean output on the correct functioning of HW components
Post Condition	The hardware is checked for failures
Actual Result	

9.11.1.2. Test-case-identifier 1.2

Test Case ID	RES-TC-01-2
Test Case Description	Periodic self-check of software failures
Pre-Conditions	The data stream to be monitored, and a signature of the SW execution exist
Test Steps	Monitoring of the data stream.
Test Data	
Expected Result	Boolean output on the correctness of the Software control flow
Post Condition	The software is checked for software failures and the boolean result is available
Actual Result	

9.12. Test Plan for Co-Simulation tool

This tool makes use in BIECO of the following tools:

Tool	How it is involved
FERAL	FERAL couples different simulation models and executes them for a given scenario

9.12.1. Test scenario identifier 1

Test Scenario ID	IESE-TS-01
Test Scenario Name	Input/output Interface Test
Test Case Description	For enabling coupling of simulation models, FERAL will be tested for being able to read data from the standardized interface.
Actors	FERAL developers
Pre-Conditions	An interface for connecting various simulation models is specified
Post-Condition	All data from the standardized interface can be read by FERAL

9.12.1.1. Test-case-identifier 1.1

Test Case ID	IESE-TC-01-1
Test Case Description	Input Interface test
Pre-Conditions	For enabling coupling of simulation models, FERAL will be tested for its readiness to read data from the compatible Interface.
Test Steps	Simulation model is provided in the format compatible with the BIECO framework.
Test Data	Functional Mock-up Unit
Expected Result	FERAL reads the data from the FMU
Post Condition	
Actual Result	The simulation model is specified in a format readable by FERAL/ FERAL is ready to execute simulation models specified in accordance to a well standardized interface.

9.12.1.2. Test-case-identifier 1.2

Test Case ID	IESE-TC-01-2
Test Case Description	Out Interface test. For enabling coupling of simulation models, FERAL will be tested for its readiness to write data in a component compatible with the standard interface.
Pre-Conditions	A standard interface is defined (Active MQ/ FMI).
Test Steps	Functional Mock-up Unit
Test Data	Component executed by FERAL
Expected Result	Data is delivered in a format compatible with the standard interface.
Post Condition	The simulation model is specified in a format readable by FERAL/ FERAL is ready to export results in accordance to a well standardized interface
Actual Result	FERAl interoperability with BIECO.

9.13. Test Plan for Forecasting systems failures tool

This tool makes use in BIECO of the following tools:

Tool	How it is involved
Postman	Platform for testing the REST APIs
Self-checking of vulnerabilities and failures	Provide information about the running system

9.13.1. Test scenario identifier 1

Test Scenario ID	UTC-TS-02
Test Scenario Name	Failure prediction
Test Case Description	Predict the probability of an upcoming failure.
Actors	Tool Developer and Platform Administrator.
Pre-Conditions	Enough information about the running process is available (the time evolution of the parameters, sensor data and failures).
Post-Condition	-

9.13.1.1. Test-case-identifier 1.1

Test Case ID	UTC-TC-02-1
Test Case Description	Evaluate the prediction accuracy
Pre-Conditions	Availability of enough historical data, in order to perform a good prediction.
Test Steps	Simulate dangerous and safe conditions and test the tool response.
Test Data	The time evolution of the system parameters, sensor data and failures.
Expected Result	Ability of the tool to differentiate dangerous from safe situations.
Post Condition	-
Actual Result	-

9.14. Test Plan for ResilBlockly tool

This tool makes use in BIECO of the following tools:

Tool	How it is involved
JUnit	Framework used for unit testing of ResilBlockly
Postman	Platform for testing the REST APIs of ResilBlockly
Mockito	Mocking Framework used for unit testing of ResilBlockly
Spring Framework (SpringBoot Test)	Framework used for integration testing of ResilBlockly

9.14.1. Test scenario identifier 1

Test Scenario ID	RES-TS-02
Test Scenario Name	User Authentication
Test Case Description	The user authenticates with the given credentials (email address and password)
Actors	ResilBlockly end-user, Tool Provider
Pre-Conditions	The tool provider has created an account for the user and the credentials have been communicated
Post-Condition	The user is authenticated and signed in

9.14.1.1. Test-case-identifier 1.1

Test Case ID	RES-TC-02-1
Test Case Description	The user is not registered or the inserted credentials are not correct.
Pre-Conditions	The user reaches the ResilBlockly login URL
Test Steps	The user inserts erroneous credentials.
Test Data	
Expected Result	The authentication is not successful. The user cannot access the tool
Post Condition	Login attempt is not successful. The tool cannot be accessed.
Actual Result	

9.14.1.2. Test-case-identifier 1.2

Test Case ID	RES-TC-02-2
Test Case Description	The registered user enters the correct credentials and is successfully authenticated
Pre-Conditions	The tool provider has created an account for the user and the credentials have been communicated. The user reaches the ResilBlockly login URL
Test Steps	The user inserts the correct email address and the corresponding password
Test Data	
Expected Result	The authentication is successful
Post Condition	The user is authenticated and signed in
Actual Result	

9.14.2. Test scenario identifier 2

Test Scenario ID	RES-TS-03
Test Scenario Name	Design of a Profile
Test Case Description	The user designs a ResilBlockly profile by importing an existing file or creating a new one from scratch
Actors	ResilBlockly end-user (profile or domain expert)
Pre-Conditions	User is authenticated.
Post-Condition	A profile is available in ResilBlockly and can be modified or instantiated in a model.

9.14.2.1. Test-case-identifier 2.1

Test Case ID	RES-TC-03-1
Test Case Description	Import of an existing ecore Profile
Pre-Conditions	User is authenticated. A profile exists in the file system and is in .ecore UML format
Test Steps	The user selects the import ecore feature. The ecore file is retrieved from the file system. Possible validation errors are removed by the user. The profile is saved with a new name.
Test Data	
Expected Result	The ecore file is successfully imported and the profile saved
Post Condition	A profile with the same information available in the ecore is available in ResilBlockly and can be modified or instantiated in a model.
Actual Result	

9.14.2.2. Test-case-identifier 2.2

Test Case ID	RES-TC-03-2
Test Case Description	Creation of a new Profile
Pre-Conditions	User is authenticated.
Test Steps	The user creates a new profile, resolve possible validation errors, and saves it with a new name.
Test Data	
Expected Result	The profile is successfully saved
Post Condition	A profile is available in ResilBlockly and can be modified or instantiated in a model.
Actual Result	

9.14.3. Test scenario identifier 3

Test Scenario ID	RES-TS-04
Test Scenario Name	Design of a Model
Test Case Description	The user instantiates a profile in a model
Actors	ResilBlockly end-user
Pre-Conditions	User is authenticated. At least one profile exists in ResilBlockly and has been selected
Post-Condition	A model is available in ResilBlockly for the user, and can be modified or exported (as ecore or JSON workspace).

9.14.3.1. Test-case-identifier 3.1

Test Case ID	RES-TC-04-1
Test Case Description	Import of an existing ResilBlockly Model
Pre-Conditions	User is authenticated. At least one profile exists in ResilBlockly and has been selected. A ResilBlockly Model exists in the file system.
Test Steps	The ResilBlockly model (workspace JSON file) is retrieved from the file system. The model is saved with a new name.
Test Data	
Expected Result	The existing ResilBlockly model (workspace JSON file) is successfully imported and the model is saved
Post Condition	A model is available in ResilBlockly for the user, and can be modified or exported (as ecore or JSON workspace).
Actual Result	

9.14.3.2. Test-case-identifier 3.2

Test Case ID	RES-TC-04-2
Test Case Description	Creation of a new Model
Pre-Conditions	User is authenticated. At least one profile exists in ResilBlockly and has been selected.
Test Steps	The user realizes a model and saves it with a new name.
Test Data	
Expected Result	The new ResilBlockly model is successfully saved
Post Condition	A model is available in ResilBlockly for the user, and can be modified or exported (as ecore or JSON workspace).
Actual Result	

9.14.4. Test scenario identifier 4

Test Scenario ID	RES-TS-05
Test Scenario Name	MUD communication rules
Test Case Description	
Actors	ResilBlockly end-user
Pre-Conditions	User is authenticated. A ResilBlockly Model is available for the user and it is selected. A MUD JSON related to one of the model components exists on the file system.
Post-Condition	The model includes the MUD communication rules and can be exported as extended MUD JSON

9.14.4.1. Test-case-identifier 4.1

Test Case ID	RES-TC-05-1
Test Case Description	Import of an existing MUD JSON
Pre-Conditions	User is authenticated. A ResilBlockly Model is available for the user and it is selected. A MUD JSON related to one of the model components exists on the file system.
Test Steps	The user imports the MUD JSON. The model is updated and saved.
Test Data	
Expected Result	The existing MUD JSON is imported and the communication rules contained are associated to the ResilBlockly model and the model is saved
Post Condition	A model is available in ResilBlockly for the user, and it includes the MUD communication rules.
Actual Result	

9.14.4.2. Test-case-identifier 4.2

Test Case ID	RES-TC-05-2
Test Case Description	Specification and export of MUD communication rules
Pre-Conditions	User is authenticated. A ResilBlockly Model is available for the user and it is selected.
Test Steps	The user selects a component interface. The user introduces the inputs (e.g., rule name, connection type, port, MUD-URL, etc.) through the ResilBlockly dedicated GUI. The saves the rules and the model. The user exports the extended MUD JSON.
Test Data	
Expected Result	The user introduces the inputs through the ResilBlockly dedicated GUI. The model is updated and saved. The user can export and save the obtained, extended MUD JSON.
Post Condition	A model is available in ResilBlockly for the user, and it includes the MUD communication rules. The corresponding extended MUD JSON is exported.
Actual Result	

9.14.5. Test scenario identifier 5

Test Scenario ID	RES-TS-06
Test Scenario Name	Identification and association of threats with a ResilBlockly model
Test Case Description	Threats (i.e., Weaknesses and vulnerabilities) are identified and associated to model elements (e.g., interfaces)
Actors	ResilBlockly end-user
Pre-Conditions	User is authenticated. A ResilBlockly model is available for the user and it is selected.
Post-Condition	The model includes the associated weakness(es)/vulnerabilities.

9.14.5.1. Test-case-identifier 5.1

Test Case ID	RES-TC-06-1
Test Case Description	Identification and association of weaknesses with a ResilBlockly model
Pre-Conditions	User is authenticated. A ResilBlockly model is available for the user and it is selected.
Test Steps	The user opens the Risk Assessment functionality. The user selects the Weaknesses tab. The user selects a block of the model (e.g., an interface) The user identifies one or more weaknesses (either by searching into CWE catalogue, or searching into CAPEC and retrieving CWEs, or defining a custom weakness).
Test Data	
Expected Result	The identified weakness(es) is associated to the model.
Post Condition	A model is available in ResilBlockly for the user, and it includes the associated weakness(es). The tool shows the Attack Tree related to the identified CWE weaknesses.
Actual Result	

9.14.5.2. Test-case-identifier 5.2

Test Case ID	RES-TC-06-2
Test Case Description	Identification and association of vulnerabilities with a ResilBlockly model
Pre-Conditions	User is authenticated. A ResilBlockly model is available for the user and it is selected.
Test Steps	The user opens the Risk Assessment functionality. The user selects the Vulnerabilities tab. The user selects a block of the model (e.g., an interface) The user identifies one or more vulnerabilities (either by searching into CVE catalogue, or defining a custom vulnerability).
Test Data	
Expected Result	The identified vulnerabilities are associated to the model.
Post Condition	A model is available in ResilBlockly for the user, and it includes the associated weakness(es).
Actual Result	-

9.14.6. Test scenario identifier 6

Test Scenario ID	RES-TS-07
Test Scenario Name	Model-based Risk Assessment with ResilBlockly
Test Case Description	Threats associated with model elements are analysed for determining the risk
Actors	ResilBlockly end-user, security expert
Pre-Conditions	User is authenticated. A ResilBlockly model is available for the user and it is selected. Weaknesses and Vulnerabilities have been identified and associated with the model elements
Post-Condition	The model includes the associated weakness(es)/vulnerabilities and the related risk assessment.

9.14.6.1. Test-case-identifier 6.1

Test Case ID	RES-TC-07-1
Test Case Description	Risk assessment of model weaknesses
Pre-Conditions	User is authenticated. A ResilBlockly model is available for the user and it is selected. Weaknesses have been identified and associated with the model elements.
Test Steps	The user opens the Risk Assessment functionality. The user selects the Risk tab. The user selects the Weaknesses inner tab. The user selects a block of the model (e.g., an interface) The tool shows the associated weaknesses. The user determines severity and likelihood of one or more weaknesses.
Test Data	
Expected Result	The tool determines the risk for the weaknesses
Post Condition	A model is available in ResilBlockly for the user, and it includes the associated weakness(es) and the related risk assessment.
Actual Result	-

9.14.6.2. Test-case-identifier 6.2

Test Case ID	RES-TC-07-2
Test Case Description	Risk assessment of model vulnerabilities
Pre-Conditions	User is authenticated. A ResilBlockly model is available for the user and it is selected. Vulnerabilities have been identified and associated with the model elements.
Test Steps	The user opens the Risk Assessment functionality. The user selects the Risk tab. The user selects the Vulnerabilities inner tab. The user selects a version of the CVSS. The user selects a block of the model (e.g., an interface) The tool shows the associated vulnerabilities. The tool retrieves the CVSS base score for the vulnerabilities of the block. The user determines the likelihood of one or more vulnerabilities.
Test Data	
Expected Result	The tool determines the risk for the vulnerabilities
Post Condition	A model is available in ResilBlockly for the user, and it includes the associated vulnerabilities and the related risk assessment.
Actual Result	-

9.15. Test Plan for safeTBox tool

This tool makes use in BIECO of the following tools:

Tool	How it is involved
Resilblockly	Provides input to safeTbox in the form of generated models.

9.15.1. Test scenario identifier 1

Test Scenario ID	IESE-TS-02
Test Scenario Name	Validate safeTbox input
Test Case Description	Validate that the input received (system models and analysis results) is acceptable for safeTbox.
Actors	IESE, RES
Pre-Conditions	-
Post-Condition	-

9.15.1.1. Test-case-identifier 1.1

Test Case ID	IESE-TC-02-1
Test Scenario Name	Validate safeTbox output
Test Case Description	Confirm that the output from safeTbox (generated mitigation strategies) are valid.
Actors	IESE
Pre-Condition	-
Post-Condition	-

9.16. Test Plan for Accountability through Blockchain tool

This tool does not rely on other BIECO tools.

9.16.1. Test scenario identifier 1

Test Scenario ID	7B-TS-01
Test Scenario Name	Confidentiality and integrity
Test Case Description	Validate that the communication between the logging host and the tool is confidential and integral.
Actors	- Logging process/host - Tool process/host - Eavesdropping process/host Injecting process/host
Pre-Conditions	Both sides are set up to exchange messages.
Post-Condition	Messages are exchanged

9.16.1.1. Test-case-identifier 1.1

Test Case ID	7B-TC-01-1
Test Case Description	Validate that the intercept able stream of data cannot be deciphered.
Pre-Conditions	Communicated data between application and the tool.
Test Steps	- Set up an eavesdropper and collect the stream of data. Verify the stream of data is encrypted and, despite the knowledge of the protocol, cannot be easily deciphered.
Test Data	n/a
Expected Result	The stream could not be deciphered.
Post Condition	Encrypted stream.
Actual Result	Encrypted stream.

9.16.1.2. Test-case-identifier 1.2

Test Case ID	7B-TC-01-1
Test Case Description	Validate that an impersonating stream of data is not accepted by the tool.
Pre-Conditions	Communicated data between application and the tool.
Test Steps	- Set up an impersonator and send messages to the tool claiming it is the logging process/host. Verify the tool rejects these messages as not being authentic.
Test Data	n/a
Expected Result	The false stream is rejected by the tool.
Post Condition	Only correct data accepted by tool.
Actual Result	Only correct data accepted by tool.

9.16.2. Test scenario identifier 2

Test Scenario ID	7B-TS-02
Test Scenario Name	Accountability
Test Case Description	Validate that the collected data allows to detect changes in logs.
Actors	- Logging process/host Tool process/host
Pre-Conditions	Both sides are set up to exchange messages and some logs metadata has been deposited already.
Post-Condition	Both sides are set up to exchange messages and some logs metadata has been deposited already.

9.16.2.1. Test-case-identifier 2.1

Test Case ID	7B-TC-02-1
Test Case Description	Validate changes in original data cause accountability errors.
Pre-Conditions	-
Test Steps	- Modify logs on the logging host. Ask the tool to verify the accountability.
Test Data	n/a
Expected Result	The tool should report inconsistencies.
Post Condition	Reported inconsistencies.
Actual Result	Reported inconsistencies.
Post Condition	Only correct data accepted by tool.
Actual Result	Only correct data accepted by tool.

9.17. Test scenario for the Runtime phase

Referring to the deliverable D2.3 and D 5.1 for more details, in the following a summary of the activities related to the runtime phase and it the relative testing scenario are provided.

Auditing Framework Activity

The target of the Auditing Framework is to monitor functional and non-functional properties when a new device or component is integrated into an existing System of

Systems (SoS), facilities so as to assess and prevent anomalous and dangerous situations

Therefore, the Auditing Framework will be validated in order to assure that it is able to:

- collect and analyze data coming from the different SoS sources (e.g., sensors, components or devices);
- assess the run time SoS (components or devices) behavior;
- promptly rise up alarms in case of violations.

The integration and testing of the Auditing Framework inside an SoS environment may involve the participation of different stakeholders such as end users, tools developers, and platform administrators (referred as SoS domain experts, device developers and monitoring experts in D5.1 respectively). In particular, two possible alternatives are considered:

1. **Using Inferred knowledge (UIK):** i.e., deriving monitoring knowledge by exploiting general, available information about the device and the relative SoS or Controlled Environment (CE). This includes:
    1. the formal representation of the knowledge about SoS, Ecosystems and devices available in literature and in practice (e.g., ontologies, requirements, guidelines, standards or behavioral models);
    2. the formal representation of the knowledge of SoS and Ecosystems derived from the BIECO 's Design Phase (e.g., knowledge extracted from the Blueprints, Security and Privacy Claims or Vulnerabilities and risk analysis).
2. **Using Explicit knowledge (UEK)**: i.e., deriving monitoring knowledge by exploiting an already available behavioral specification of the device and the relative Controlled Environment, provided by the BIECO users.

Integration and testing activity target therefore the overall process of the Auditing Framework as shown in D5.1 Figure 4 reported here below for completeness.

As reported in the picture, the main testing and integration activity will involve different sub-processes.

Main Scenario

The main test scenario will involve the validation of a request from the BIECO End User. In this case, the scenario will simulate the interaction with the BIECO framework GUI so as to start the BIECO Runtime Phase and consequently the setting up of the Auditing Framework and the Controlled Environment.

As in **Error! Reference source not found.** below (see Figure 4 of D5.1) each interaction t hrough the BIECO Framework (GUI) for setting up the Auditing Framework will be verified with specific test cases taken from the BIECO Use Cases.

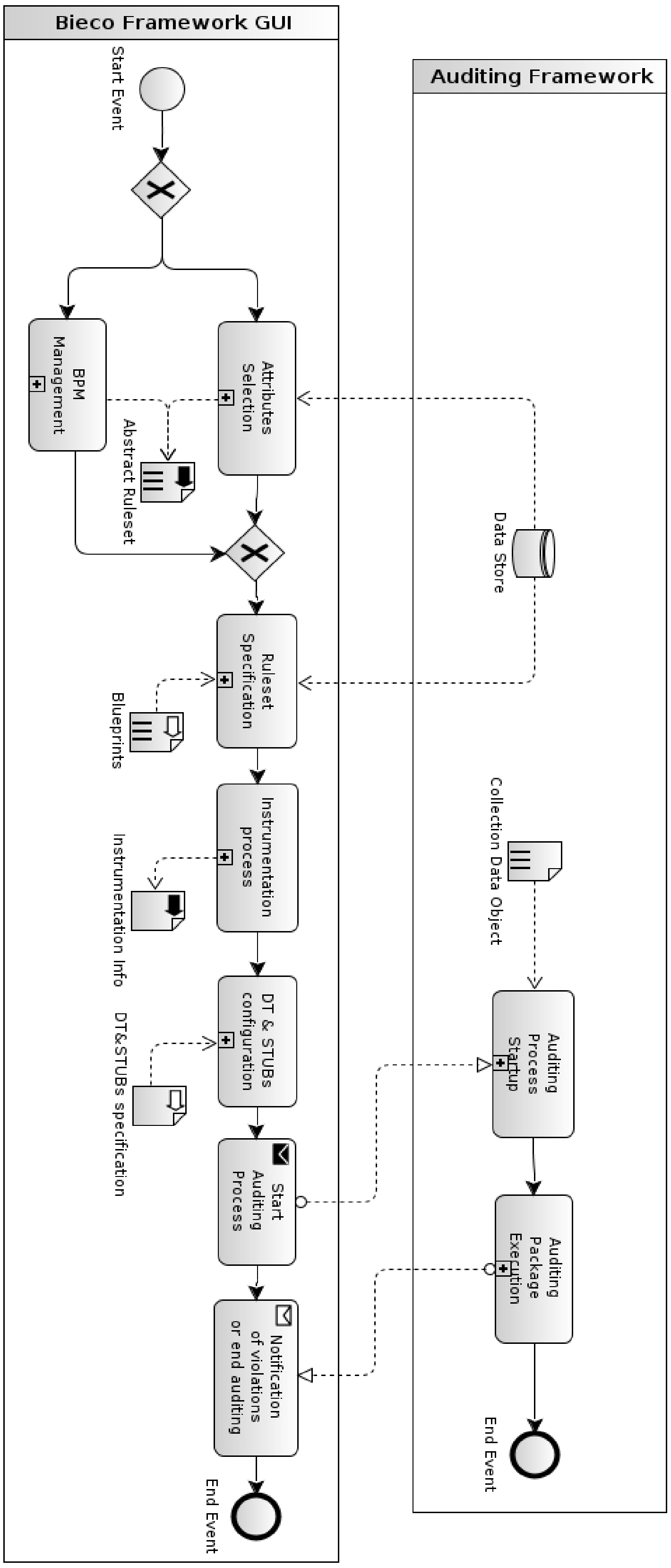


Figure 10 Audit framework main behaviour (taken from D5.1 Figure 4).

Specifically, the following sub-processes will be verified with the following scenarios:

Test sub-scenario 1 (UIK)

Attributes selection subprocess is the starting activity of the **Using Implicit knowledge flow (UIK).**

- **Test sub-sub-scenario 1.1** peculiarities and the requirements of the device and the Ecosystem specified. This scenario will include the verification of the management of the ontology-based representation of the different ecosystems and monitoring knowledge and targets the selection of the attributes (such as time-duration, number-of-connections) related to the device.
- **Test sub-sub-scenario 1.2** verify that after the execution of sub-sub-scenario 1.1 a set of rules called "*Abstract Ruleset*" will be generated and correctly stored.

Test sub-scenario 2 (UEK)

"*BPM Managemen*t" is verified in order to check the starting sub-process of the **Using Explicit knowledge flow (UEK).** This scenario enables the user to:

- **Test sub-sub-scenario 2.1** verify the generation and the loading of business process models that represent the behavior of the device and the CE that is going to be monitored.
- **Test sub-sub-scenario 2.2** Verify that the set of activities can be enriched with parameters.
- **Test sub-sub-scenario 2.3** Verify that it is possible to express functional and non-functional properties about a specific activity.
- **Test sub-sub-scenario 2.4** Verify that the properties will be used for generating the rules that will be monitored at runtime.

Test sub-scenario 3 (RuleSet)

The results of the two alternative flows (Test scenario 1 and 2) starts the sub-process *RuleSet Specification*.

- **Test sub-sub-scenario 3.1** Verify that the *Abstract Ruleset* and the *Blueprints* generated during the Design Runtime phase execution of the BIECO platform can be retrieved.
- **Test sub-sub-scenario 3.2** Verify that the BIECO Platform allows the user to detail the *Abstract Ruleset* with *Blueprints*.

Test sub-scenario 4 (Instrumentation)

This test scenario is related with *Instrumentation process* to the probe injection within the device under test. This activity will provide guidelines to the user for instrumenting code or for using an automatic instrumentation tool for getting continuous information about the Device under test through delivery of events. Therefore, the objective is to verify that the instrumentation will take place.

Test sub-scenario 5 (Digital Twin)

verify that the subprocess *DT&STUBs configuration* will take place.

- **Test sub-sub-scenario 5.1** verify that the configuration of the Digital Twin involved within the *Conformity Monitoring*.
- **Test sub-sub-scenario 5.2** verify the configuration of the STUBs that simulates the external services involved within the device execution

Test sub-scenario 6 (Auditing Framework set up)

Verify the initial setup of the auditing process. It includes the communication channels creation and setup.

Test sub-scenario 7 (Auditing framework information retrieval)

Verify that the audit framework retrieves the necessary data as well as input data provided as *Blueprints*.

Test sub-scenario 8 (Auditing framework execution)

Verify that the Auditing Framework is ready to start its activities.

- **Test sub-sub-scenario 8.1** verify that the subprocess called *Auditing Process Startup* is invoked from the *BIECO Framework GUI* (see *Start Auditing Process* activity) when all the data is ready for being used for instantiating the Monitoring and the Predictive Simulation components within the Auditing Framework.
- **Test sub-scenario 8.2** After the completion of the startup phase, verify that the *Auditing package Execution* process can be executed.

Test sub-scenario 9 (Auditing framework notification)

Verify the notification about violation related to the functionals or non-functionals properties generated from Blueprints and information gathered in rules are forwarded to the BIECO Framework GUI.

10. Conclusions

The present deliverable was prepared in the frame of WP8 Platform Integration and testing, with the main goal to precisely describe the testing strategy for the foreseen BIECO platform. Another goal of the present deliverable was to present the main approach that will be adopted during the development and testing of the BIECO platform, in order to continuously measure the extent to which the final functional and non-functional requirements are met.

The detailing methodology regarding the test validation of project results will be described in D8.2 BIECO Assessment methodology.

Three test levels were identified that specify the process in that particular scope (Figure 11). Generalized overall test implementation principles describe processes that are applicable to all test levels. Added to those test levels, Non-functional testing is instructed to aid testing in scope of fulfilment of the non-functional requirements, but also functional.

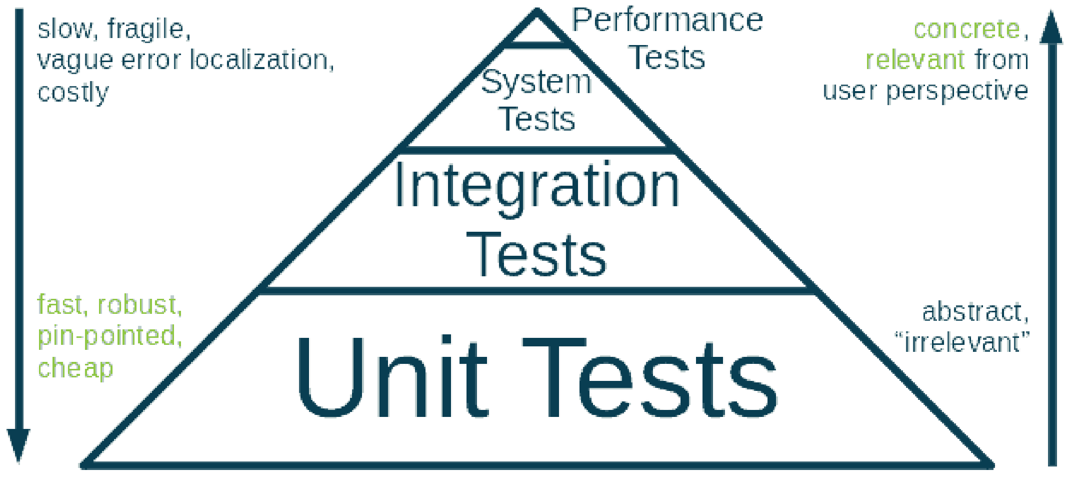

slow, fragile, vague error localization, costly
fast, robust, pin-pointed, cheap
Performance Tests
System Tests
Integration Tests
Unit Tests
concrete, relevant from user perspective
abstract, "irrelevant"

Figure 11 Software Testing Pyramid

Continuous integration will be applied in order to have a short and fast feedback loop and enable the BIECO partners to collaborate together to develop the BIECO platform. The deliverable defines also the infrastructure to support and facilitate the tests.

The remainder of the deliverable presents the test scenarios and test cases needed to test and validate:

- BIECO Platform;
- The four Use Cases;
- BIECO Tools.

11. References

[1] Gorton, Ian. "Software quality attributes." Essential Software Architecture. Springer, Berlin, Heidelberg, (2011). 23-38.

[2] M. Felderer and E. Fourneret, “A Systematic Classification of Security Regression Testing Approaches, ”International Journal on Software Tools for Technology Transfer, vol. 17, no. 3, pp. 305–319, 2015

[3] Index, T. I. O. B. E. "Tiobe-the software quality company." *TIOBE Index| TIOBE–The Software Quality Company [Electronic resource]. Mode of access: https://www. tiobe.com/tiobe-index/-Date of access* 1 (2018).

[4] Mccabe, Thomas. "Cyclomatic complexity and the year 2000." IEEE Software 13.3 (1996): 115-117.

[5] Fowler, Martin, and Matthew Foemmel. "Continuous integration."

[6] Shahin, Mojtaba, Muhammad Ali Babar, and Liming Zhu. "Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices." IEEE Access 5 (2017): 3909-3943.

[7] Ståhl, Daniel, and Jan Bosch. "Cinders: The continuous integration and delivery architecture framework." Information and Software Technology 83 (2017): 76-93.

[8] Meyer, Mathias. "Continuous integration and its tools." IEEE software 31.3 (2014): 14-16.