D2.8 – VALIDATION FRAMEWORK SPECIFICATION

Deliverable ID	**D2.8**
Deliverable Title	**Validation Framework Specification**
Work Package	**WP2**
Dissemination Level	**PUBLIC**
Version	**1.0**
Date	**2018-06-29**
Status	**Final**
Lead Editor	**Bálint József Jánvári (SLAB)**
Main Contributors	**Regina Krisztina Bíró (SLAB), Bálint József Jánvári (SLAB)**

Published by the CPSwarm Consortium

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 731946.

Document History

Version	Date	Author(s)	Description
0.1	2017-08-10	Regina Krisztina Bíró (SLAB), Bálint József Jánvári (SLAB)	First Draft with TOC, introduction and brief description of methodology, as well as templates and a few examples in the security section
0.2	2017-09-05	Bálint József Jánvári (SLAB)	Replaced the term "use case" with "scenario" where appropriate, clarified the description of the goal "Ensuring industrial impact"
0.3	2017-11-24	Regina Krisztina Bíró (SLAB)	Integrated requirements for the Modelling Tool, Modelling Library, Optimization Tool and Optimization Simulator, added KPIs and test cases for the requirements describing the Modeling Tool and Library.
0.31	2018-01-11	Regina Krisztina Bíró (SLAB)	Refinement of test cases
0.32	2018-01-17	Etienne Brosse (SOFT)	Review of the KPIs and test cases related to the Modelling Tool
0.4	2018-04-20	Regina Krisztina Bíró (SLAB)	Integration of new requirements
0.5	2018-04-22	Regina Krisztina Bíró (SLAB), Bálint József Jánvári (SLAB)	Refined methodology
0.6	2018-05-24	Regina Krisztina Bíró (SLAB), Bálint József Jánvári (SLAB)	Added test cases and KPIs defined for the rest of the components, marked remaining open questions to be discussed
0.7	2018-06-14	Regina Krisztina Bíró (SLAB), Bálint József Jánvári (SLAB)	Matched evaluation dates to project milestones, changed methodology, refined some of the test cases
0.8	2018-06-18	Regina Krisztina Bíró (SLAB), Bálint József Jánvári (SLAB)	Added target maturity levels for milestones and made small adjustments to per test case target maturity levels.
1.0	2018-06-29	Regina Krisztina Bíró (SLAB), Bálint József Jánvári (SLAB)	Final check, integrated comments from internal reviewers

Internal Review History

Review Date	Reviewer	Summary of Comments
2018-06-26	Sisay Chala (FRAUNHOFER)	Approved with minor comments.
2018-06-28	Arthur Pitman (UNIKLU)	Approved with minor comments.

Table of Contents

1 Introduction

1.1 What is validation and verification?

Validation and verification, often used together in quality management systems, are independent procedures used to check whether a (software) product, service or system meets predefined requirements and specifications and fulfills its intended purpose.

Since the usage of these two terms varies - and sometimes they are used interchangeably - the objective of this chapter is to clarify what we mean by validation and verification. We intend to follow the usual definitions, simplified as:

- Validation:
  Are we building the right system?
- Verification:
  Are we building the system right?

More precisely, validation is concerned with assuring that a product, service or system meets the needs of its customers and other stakeholders, while verification is the evaluation of whether a product, service or system complies with its requirements and specifications ensuring that the product is well-engineered and error free. Verification is usually an internal process that helps determine whether the product is of high quality, but it does not ensure that the product is actually useful.

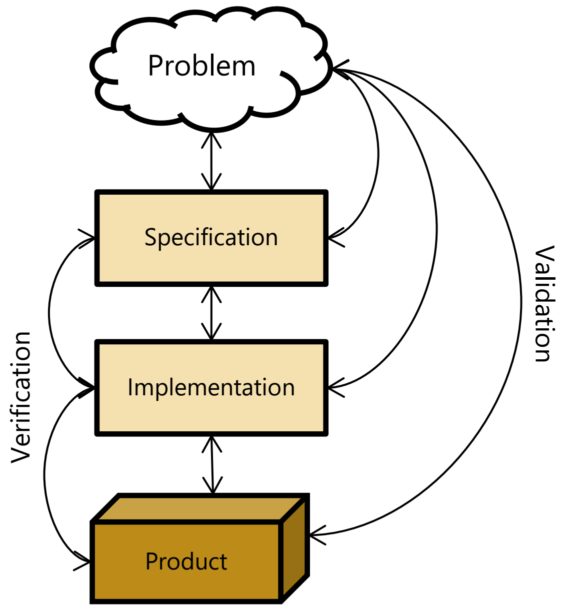

Figure 1 – Validation and verification

We can distinguish between the two terms by considering their respective roles with respect to the specification. Validation checks whether the specification captures the stakeholders' needs, while verification ensures that the product meets the specification. Evaluation items and activities also differ for the two terms as seen in Table 1.

Table 1 – Comparison of verification and validation

	Verification	Validation
Evaluation items	• Plans • Requirement specifications • Design specifications • Code • Test cases	• Products • Services • Systems
Activities	• Reviews • Walkthroughs • Inspections	• Testing

A commonly used, but less than ideal approach is that verification is only used to check that the product satisfies its requirements and validation is performed only at the beginning and end of the project: for requirements engineering and acceptance testing. However, we cannot assume that the stakeholders' needs can be captured at once in the beginning of the project, and that these requirements will not change while the product is being developed. Therefore, following the methodology depicted in Figure 2, both validation and verification will be applied throughout the lifecycle of the CPSwarm project.

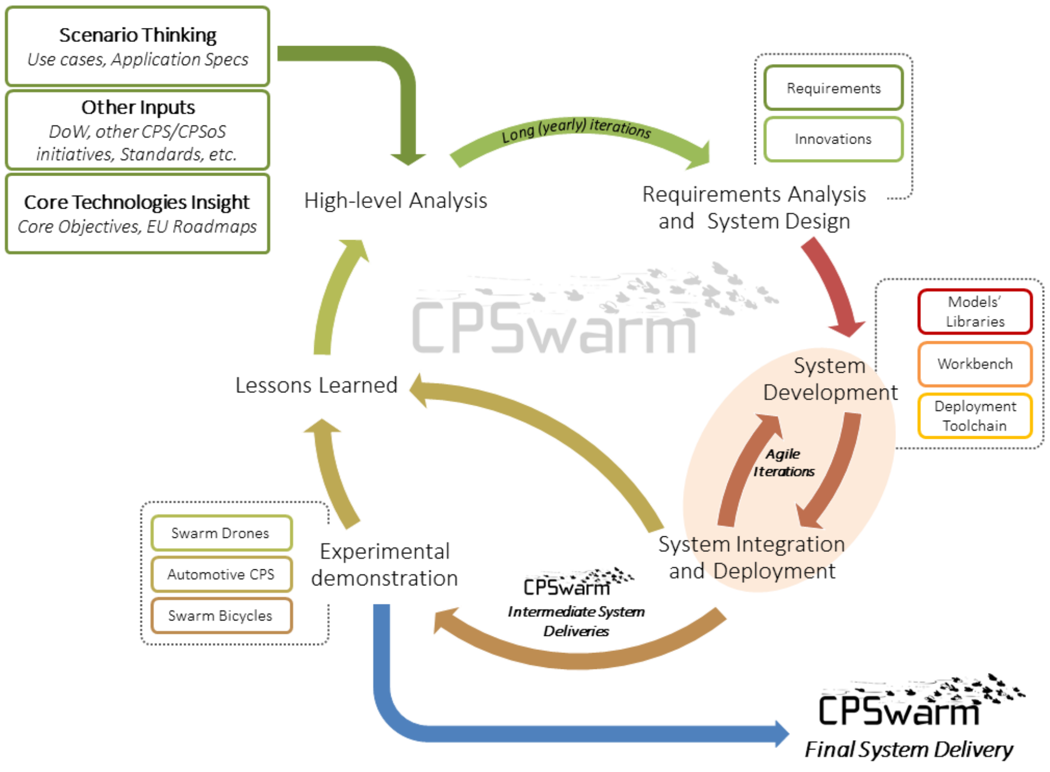

Figure 2 – CPSwarm project lifecycle

1.2 Goals

1.2.1 Ensuring industrial impact

From its inception, the CPSwarm project has been striving to create a platform that is relevant to the current state-of-the-art of the industry and that can offer a solution that is more effective and integrated than available alternatives.

Since each requirement is related directly or indirectly to one or more industrial scenarios, by verifying these requirements the Validation Framework also ensures that the development goals align with market requirements.

1.2.2 Continuous verification of project requirements

The Validation Framework establishes a reference for the continuous verification of project requirements. The framework provides a stable baseline for measuring the maturity of project components and the status of the project in general. Future validation and verification activities will be able to use this baseline to evaluate implemented functionality - including the evaluation of the finished product at the end of the project.

This continuous validation and verification can aid project management and software development by providing important feedback on the status of individual components and on the maturity of the project as a whole. Identification of requirements not yet met by components can also help software development teams focus their efforts on delivering a working solution.

1.2.3 Quality assurance

By defining ways to evaluate the system as a whole from the perspective of its end users, important aspects relevant to the quality of the product can be made measurable or at least verifiable. By verifying user experience requirements and providing continuous feedback on how well these high-level requirements are met, development efforts can focus on creating a product that not only works, but works well.

1.3 Related documents

ID	Title	Version	Date
D2.1	Initial Vision Scenarios	2.0	M4
D2.3	Initial Requirements Report	1.0	M6
D2.6	Updated Lessons Learned and Requirements Report	1.0	M14
D2.7	Final Lessons Learned and Requirements Report	N/A	M26
D3.1	Initial System Architecture Analysis & Design Specification	1.0	M6
D3.2	Updated System Architecture Analysis & Design Specification	N/A	M18
D3.3	Final System Architecture Analysis & Design Specification	N/A	M30
D8.7	Initial Validation results	N/A	M24
D8.8	Final Validation results	N/A	M36

2 Methodology

2.1 Types of metrics

2.1.1 Key Performance Indicator (KPI)

Key performance indicators are measurements that are used to evaluate the success of an organization or particular activities such as projects, programs or other initiatives. KPIs define a set of values against which to measure performance. These sets of values are called indicators, and can be divided into sub-categories like:

- *Quantitative indicators* that can be represented by a number
- *Qualitative indicators* that cannot be represented by a number
- *Input indicators* that measure the amount of resources used during the creation of the outcome or during the deployment of a use-case

For strategic development, KPIs can be viewed as objectives to be targeted that will bring the most value to the CPSwarm project. For deployment, KPIs can pose as a threshold for the definition of a successful mission concerning the use-cases.

In the case of the CPSwarm Workbench, Key Performance Indicators (KPIs) evaluate the success of the design and implementation of each component or the workbench as a whole according to the stakeholders' requirements and specifications. Target values for KPIs were established based on feedback from consortium members on the planned roadmap of each component.

2.1.2 Test case

Test cases are introduced in order to verify compliance with one or more requirements. The aim of running a test is to gain information, for example about whether a component will pass or fail the test. Test cases are the basis of quality management where they are designed to verify the quality, usability and behavior of the product. Test cases can be formal or informal - formal ones are defined with an input and an expected output, before the test is run. Formal test cases verify formal requirements in a way that for every requirement there are two test cases defined: one positive test and one negative test. If a component or scenario does not hold any formal requirements, informal test cases are introduced based on the normal behavior of a similar component or scenario.

2.1.3 Maturity

Maturity levels are designed in order to describe the progression and quality of each component and the workbench as a whole. Maturity is achieved by reaching specific goals and KPIs and by successfully passing test cases. We define five levels of maturity:

1. Proof of concept (demonstrates feasibility)
2. Working (core features are present)
3. Feature complete (all planned features are present)
4. Optimized (performance matches expectations, reasonably error free)
5. Production ready (meets standards, has documentation, easy to use)

Maturity levels signify milestones on our way to the finished product, and align with the iterative approach of the CPSwarm project lifecycle – driven by continuous feedback, components should move from being prototypes to being relevant and applicable in industrial scenarios. ML1 needs to be reached in order to show off a minimum viable product, while ML2 and ML3 each add new features and opportunities to apply the software. ML4 and ML5 are related to quality and applicability.

2.2 Establishing metrics

2.2.1 Identifying relevant KPIs and test cases for components and the whole project

Earlier deliverables, including D2.1 - *Initial Vision Scenarios and Use Case Definition* and D2.3 - *Initial Requirements Report*, have already done much to determine the stakeholder groups and their requirements as well as to segment the project into components. Building on that work, requirements can be grouped into three categories:

- Requirements related to individual components
- Requirements related to user experience
- Scenario specific requirements

The first two groups contain requirements common to all scenarios or considered generic enough to be included in the core set, while scenario specific requirements are only encountered in a specific industrial scenario. As the definition of requirements progressed, scenario specific requirements were incorporated into the core set of requirements.

These requirements need to be translated into measurable metrics - either by defining test cases that, when passed, imply that the requirement has been met, or by finding KPIs and setting their target value in a way that supports the assumption that the requirement has been met.

The Validation Framework will focus on a component-centric approach, with all test cases and KPIs inherently bound to one main component. Integration related requirements and user experience requirements will be grouped in such a way as to fit into this model.

2.2.2 Establish a maturity scoring system for KPIs and test cases

Meeting the target of a KPI or passing a test case indicates that the project is making progress - but to measure how much, these events need to be linked to specific maturity levels. A KPI might have different target values for different maturity levels, so for each KPI targets should be set for each maturity level. Tests, when passed, should also have a target maturity level. A component is considered to have reached a certain maturity level if all metrics linked to that maturity level have reached their targets. The workbench as a whole would take on the maturity level of its least mature component.

2.2.3 Set target dates - milestones - for reaching specific maturity levels

Building on the roadmap outlined in the project proposal and the three phases defined, a number of milestones can be set based on the planned due dates of relevant deliverables, setting target maturity levels for each component and the project as a whole at that specific milestone. While performing the test cases and evaluating the KPIs should be performed on a more regular basis to provide continuous feedback, a comprehensive report on the current maturity level of all components will only be prepared at the end of Phase 2 and Phase 3, thus targets will only be set for these milestones.

2.3 Continuous validation and verification

2.3.1 Track and validate changes to project requirements

As project requirements change, the changes need to be validated against the use cases. Once requirements have actually been changed, metrics need to be adjusted to correctly verify the changed requirements. New requirements should be carefully examined to ensure that they are related to actual industrial use cases, while in case of changing or removed requirements it must be ensured that the applicability of the end product in any of the use cases is not compromised.

2.3.2 Continuously gather data on KPIs and periodically perform test cases

As a follow up of the definition of the metrics above, and as part of T8.4 - *Use cases validation*, these metrics need to be gathered and evaluated periodically. If problems are encountered in meeting a KPI or passing a test case, resources may need to be reallocated or the requirements may need to be reevaluated or changed - either way, such incidents can be reported and requirements can be adjusted after validation as part of D2.6 - *Updated Lessons Learned and Requirements Report* and D2.7 - *Final Lessons Learned and Requirements Report*.

2.3.3 At each milestone, assess maturity and provide feedback

On each milestone, the maturity level of components and the system as a whole should be evaluated and compared to the target maturity level. If the project falls behind its target maturity levels, feedback should be given to project management on problematic areas and corrections should be made to catch up with the timeline. The result of these validation activities will be later documented in D8.7 - *Initial Validation results* and D8.8 - *Final Validation results*.

2.4 Templates

Metrics are defined in the standard format for each type as seen below. Each metric must have a unique name and must reference the requirements verified.

2.4.1 Template for KPIs

Metric name	*<name>*				
Verified requirements	*<list of relevant requirements>*				
Measurement	*<how to measure the metric>*				
Target values	ML1	ML2	ML3	ML4	ML5
	<target>	*<target>*	*<target>*	*<target>*	*<target>*
Notes	*<any further information, if required>*				

2.4.2 Template for formal test cases

Metric name	*<name>*	
Verified requirements	*<list of relevant requirements>*	
Maturity level	*<target maturity level>*	
Steps to perform	Positive test	Negative test
	<list of steps for positive test>	*<list of steps for negative test>*
Expected results	*<expected results for passing test>*	*<expected results for passing test>*
Notes	*<any further information, if required>*	

2.4.3 Template for informal test cases

Metric name	*<name>*
Verified requirements	*<list of relevant requirements>*
Maturity level	*<target maturity level>*
Steps to perform	*<list of steps>*
Expected results	*<expected results for passing test>*
Notes	*<any further information, if required>*

3 Components

This chapter collects the requirements identified concerning each component for the CPSwarm Workbench, as well as the Test Cases or KPIs and maturity levels identified for each requirement as part of the verification process. The structure of each sub-chapter is as follows:

- List of identified requirements for the given component
- Description of formal or informal test cases for them (if any)
- Description of the identified KPI and maturity level (if any)

3.1 Modelling Tool

The Modelling Tool is a graphical interface offering functions to model the swarm structure, behavior, environment and other necessary parameters. The Modelling Tool provides an easy way for swarm experts to design a swarm without having profound expertise in programming and/or hardware specific knowledge [1].

List of Requirements for the Modelling Tool
CRD-2 The Modelling Tool shall be able to use / reuse models from the Modelling Library
CRD-3 The Modelling Tool shall be able to model the structure of a swarm member
CRD-4 The Modelling Tool shall be able to model the behavior of a swarm member
CRD-6 The Modelling Tool shall be able to model the composition of a swarm
CRD-7 The Modelling Tool shall be able to model fitness function to define the goal of the swarm behavior
CRD-9 The Modelling Tool shall pass the end condition of simulation to the Optimization Tool
CRD-10 The Modelling Tool shall pass the environment model to the Optimization Tool
CRD-11 The Modelling Tool shall pass the swarm model to the Optimization Tool
CRD-12 The Modelling Tool shall pass fitness function to the Optimization Tool
CRD-13 The Modelling Tool shall pass the swarm composition to the Optimization Tool
CRD-21 The Modelling Tool should be able to present the structural diagram of a swarm member
CRD-30 The Modelling Tool shall enable users to create models and publish them in a private library
CRD-31 The Modelling Tool shall contain an editor to formulate the fitness function
CRD-32 The Modelling Tool shall be able to model the behavior of the swarm member using the swarm member behavior library
CRD-33 The Modelling Tool shall be able to model a local state as a part of the swarm member structure
CRD-54 The Modelling Tool shall be responsible for passing swarm member structure to the code generator
CRD-55 The Modelling Tool shall be responsible for passing swarm member behavior to the code generator
CRD-62 The Modelling Tool shall make it possible to define events

CRD-65 The Modelling Tool shall distinguish between swarm, member and component scope events, which are defined at their respective level in the model hierarchy
CRD-66 The Modelling Tool shall make it possible to trigger events based on the current value of the inputs and outputs defined for the low-level behavior of the current state
CRD-69 The Modelling Tool shall make it possible to add additional swarm scope events to each state transition that are triggered when the transition happens
CRD-77 The Modelling Tool shall make it possible to design systems with multiple behaviors where events can trigger a behavior change
CRD-87 The Modelling Tool shall let multiple high-level behaviors coexist within the same project
CRD-100 The Modelling Tool shall make it possible to specify event scope.
CRD-101 The Modelling Tool shall namespace component scope events to their respective component

Metric name	The Modelling Tool is able to use / reuse models from the Modelling Library
Verified requirements	CRD-2, CRD-30
Maturity level	ML1
Steps to perform	1. Open the Modelling Tool, create a CPSwarm project and then open the Modeling Library option. 2. Drag and drop models contained in the Modeling Library to the actual project created.
Expected results	The loaded models can be used as they are or can be tailored according to specific needs (see the following test cases).

Metric name	The Modelling Tool shall be able to model the structure of a swarm member
Verified requirements	CRD-3, CRD-21, CRD-33
Maturity level	ML1
Steps to perform	1. Open a CPSwarm project or create a new one in the Modelling Tool. 2. Assemble the model of the target swarm member using the Swarm Member Architecture diagram palette: add actuators, controllers, sensors, data flow indicators etc. to the model to represent the internal architecture of the swarm member.

Expected results	The created diagram represents the structure of the swarm member.

Metric name	The Modelling Tool shall be able to model the behavior of a swarm member
Verified requirements	CRD-4, CRD-32
Maturity level	ML1
Steps to perform	1. Open a CPSwarm project or create a new one in the Modelling Tool. 2. Assemble the behavioral model of the target swarm member using the Behavioral Modelling diagram palette: add states, transitions and pseudo-states to the model to represent the behavior of the swarm member.
Expected results	The created diagram represents the behavior of the swarm member as a state machine.

Metric name	The Modelling Tool shall be able to model the composition of a swarm
Verified requirements	CRD-6, **CRD-48**
Maturity level	ML1
Steps to perform	1. Open a CPSwarm project or create a new one in the Modelling Tool. 2. Assemble the model of the target swarm using the Swarm Architecture diagram palette: add swarms, swarm members, interfaces, attributes etc. to the model to represent the architecture of the swarm.
Expected results	The resulting diagram can represent the composition of the swarm.

Metric name	The Modelling Tool shall be able to model fitness function to define the goal of the swarm behavior
Verified requirements	CRD-7, CRD-31
Maturity level	ML2

Steps to perform	1. Open a CPSwarm project or create a new one in the Modelling Tool. 2. Establish the model of the fitness function using the Fitness Function Specification diagram palette: add the Fitness Function, parts representing the internal instantiations of components, ports for data flow communication between components, attributes etc. to the model to represent the fitness function.
Expected results	The fitness function can be passed to the Optimization Tool and the optimization can be generated (see CRD-12, CRD-20).

Metric name	The Optimization Tool is integrated with the Modeling Tool
Verified requirements	CRD-9, CRD-10, CRD-11, CRD-12, CRD-13, CRD-31
Maturity level	ML1
Steps to perform	The Modelling Tool has to pass the end condition of simulation, environment model, swarm model, fitness function and swarm composition to the Optimization Tool: 1. Define a fitness function that describes the goal of the swarm using the Modeling Tool 2. Generate a Optimization Project using the corresponding module in the Modelling Tool 3. Export the current project's parameters to the Optimization Tool. 4. The generated files containing the parameters defined in the Modeling Tool shall be saved in the dedicated folder from which the Optimization Tool can load and use them.
Expected results	The optimization should be able to run using the passed parameters.

Metric name	The Modelling Tool is responsible for passing swarm member structure to the code generator
Verified requirements	CRD-54, CRD-55
Maturity level	ML2
Steps to perform	1. When the project to be exported is ready, choose the option in the Modelling Tool which generates the definition of the swarm member structure and behavior in a standardized form.
Expected results	The files generated by the Modelling Tool can be used as valid inputs to the Code Generator.

Metric name	The Modelling Tool makes it possible to define events
Verified requirements	CRD-62, CRD-65, CRD-66, CRD-69, CRD-100, CRD-101
Maturity level	ML2
Steps to perform	1. When creating a low-level state machine for describing the behavior of a swarm member, create event trigger points connected to e.g. input/output values and define events that can refer to other behaviors in the high-level state machine. 2. Mark these events according to their scope – swarm, swarm member or component, where swarm and swarm member scope events have to be handled as privileged commands.
Expected results	The high and low-level state machines that describe the behavior of a swarm member accurately describe the input and output values that can trigger a change in behavior in different scopes including components, swarm members or the whole swarm.

Metric name	The Modelling Tool makes it possible to design swarm members with multiple behaviors
Verified requirements	CRD-77, CRD-87, **CRD-47**
Maturity level	ML2
Steps to perform	1. Define low level state machines for the desired behaviors of to the swarm member. 2. Start defining a high-level state machine and perform the steps described in the test "The Modelling Tool makes it possible to define events".
Expected results	The high-level state machine now defines the logic and transition between the different behaviors.

3.2 Modelling Library

The Modelling Library is a collection of reusable CPS models, swarm behavior algorithms, security guidelines etc. It enables high reusability and interoperability of core functions adopted in swarm development [1].

List of Requirements for the Modelling Library

CRD-1 The Modeling library will be a collection of different kinds of reusable components
CRD-22 The Modelling library shall include a library to help in designing a swarm member
CRD-23 The Modelling library shall include a library to help in designing an environment
CRD-24 The Modeling library shall include a library to help in designing a goal
CRD-25 The swarm member library shall contain models for the physical aspects of the swarm member
CRD-26 The swarm member library shall contain models for the behavior of a swarm member
CRD-28 The environment library shall contain models of environments
CRD-29 The goal library shall contain various fitness functions linked to different problems
CRD-34 The Swarm member library shall contain models for sensors and actuators to be used to design a swarm member
CRD-74 Components in the Modelling Library can have component scope events associated with them, which are imported when the component is added
CRD-84 The Modelling Library shall include behaviors specific to target hardware platforms that can be used as safe default contingency plans for each CPS model (soft shutdown)
CRD-86 The Modelling Library shall include a special behavior that switches over the CPS to manual remote control

Metric name	The Swarm member library contains models for sensors and actuators to be used to design a swarm member				
Verified requirements	CRD-34, CRD -25, CRD-22, CRD-1				
Measurement	Count the number of models for the sensors and actuators to be used to design a swarm member in the modelling library. Only include completed models which have successfully been used in an example/vision scenario. From ML3 the Modeling Library should include use-case specific solutions for sensor capabilities.				
Target values	ML1	ML2	ML3	ML4	ML5
	3	5	8	10	15

The above KPI partly measures the requirement CRD-25, however to fully verify it, it also has to reach the following target maturity levels below.

Metric name	The swarm member library contains models for the physical aspects of the swarm member				
Verified requirements	CRD-74, CRD-25, CRD-22, CRD-1				
Measurement	Count the number of models for the physical aspects (e.g. sensors, controllers) of the swarm member in the modelling library. Only include completed, working models. Each of these shall possess component-scope events attached, for example events determined by input/output values.				
Target values	ML1	ML2	ML3	ML4	ML5
	2	4	8	12	15

Metric name	The swarm member library contains models for the behavior of a swarm member				
Verified requirements	CRD-86, CRD-84, CRD-26, CRD-22, CRD-1				
Measurement	Count the number of models for the behavior of a swarm member in the modelling library. Only include completed, working models. The minimum viable behavior for ML2 is including the emergency exit example (or another toy-example), and from ML3 the Modeling Library should include behaviors connected to each of the use cases. ML4-5 should contain scenario and capability-specific contingency behaviors of a swarm member, including "Emergency stop shutdown" behaviors specific to the hardware platform used and a behavior that describes the transition to manual remote control.				
Target values	ML1	ML2	ML3	ML4	ML5
	1	5	8 From ML3 including at least 1 soft shutdown contingency behavior	12 Including soft shutdown behaviors for all hardware target platforms	18 Including a transitioning behavior to manual remote control

When reached, the above defined KPIs for the requirements CRD-34, CRD-25, CRD-26 and CRD-27 also describe requirement CRD-22 (namely the Modelling library shall include a library to help in designing a swarm member) with the minimum of all maturity levels of the four KPIs.

Metric name	The environment library shall contain models of environments				
Verified requirements	CRD-28, CRD-24, CRD-23, CRD-1				
Measurement	Count the number of models for environments in the modelling library. Only include completed, working models.				
Target values	ML1	ML2	ML3	ML4	ML5
	0	1	3	4	5

When reached, the above defined KPIs for the requirement CRD-28 also describe requirement CRD-23 (namely the Modelling library shall include a library to help in designing an environment) with the same maturity levels.

When the KPIs for CRD-22 and CRD-23 are reached, they also describe CRD-1 (namely the Modeling library will be a collection of different kinds of reusable components): the minimum of all maturity levels of the two KPIs.

Metric name	Number of different fitness functions related to different problems				
Verified requirements	CRD-29				
Measurement	Count the number of fitness functions related to different problems in the modelling library. Only include completed, working examples.				
Target values	ML1	ML2	ML3	ML4	ML5
	0	1 problem 1 fitness function	1 problem 1 fitness function	>1 problems at least 1 fitness function for each	>1 problems at least 1 fitness function for each

3.3 Optimization Tool

Due to the complexity of swarm behaviors, in many cases it is very difficult, if not impossible, to define the exact algorithm to be adopted for each individual member of a swarm. For this reason, an Optimization Tool is envisioned to exploit methods based on Darwinian evolution to optimize the algorithm automatically, according to the configuration given by users [1].

List of Requirements for the Optimization Tool
CRD-14 The Optimization Tool shall pass operational commands to the Optimization Simulator
CRD-20 The Optimization Tool shall optimize the algorithm according to the fitness score
CRD-56 The Optimization Tool shall pass the optimal behavior to the code generator
CRD-91 The Optimization Tool shall only optimize one behavior at a time, but shall let the simulation used include other behaviors

Metric name	The Optimization Tool passes operational commands to the Optimization Simulator
Verified requirements	CRD-14
Maturity level	ML3
Steps to perform	Start the optimization using the Optimization Tool and the Optimization Simulator together.
Expected results	The simulation can be performed and the simulated swarm members behave as indicated by the Optimization Tool.

Metric name	The Optimization Tool shall optimize the algorithm according to the fitness score
Verified requirements	CRD-20, CRD-91
Maturity level	ML2
Steps to perform	1. Create a fitness function that defines the goal of the swarm behavior 2. Start the optimization with the fitness function and other parameters that describe the swarm, including other behaviors to be included in the simulation, e.g. malicious behavior of some agents, hardware failure, etc.
Expected results	The Optimization Tool is able to rank the candidate controllers according to the fitness score, and the optimization stops when the maximum of the fitness function is reached.

Metric name	The Optimization Tool shall pass the optimal behavior to the Code Generator
Verified requirements	CRD-56
Maturity level	ML2
Steps to perform	1. After the optimization is done, export the state machine that describes the optimized behavior and load the file with the Code Generator.
Expected results	The Code Generator can generate target platform specific code that implements the optimized behavior.

3.4 Simulation Tool

In order to evaluate an algorithm, the Optimization Tool needs an Optimization Simulator to evaluate the performance a swarm population within a "controlled" environment. Thanks to the availability of the Optimization Simulator, different generations of algorithms proposed by the Optimization Tool are ranked and optimized across multiple simulations, on the basis of achieved performances [1].

List of Requirements for the Optimization Simulator and Simulation Manager
CRD-15 The Optimization Simulator shall simulate swarm composition, swarm member structure
CRD-16 The Optimization Simulator shall simulate environment model
CRD-17 The Optimization Simulator shall calculate fitness score for each simulation
CRD-18 The Optimization Simulator shall pass the fitness score to the Optimization tool
CRD-19 The Optimization Simulator shall pass the sensor data of each swarm member back to the Optimization Tool
CRD-88 The Simulation Manager shall support simulations where different swarm members have different behaviors
CRD-90 The Simulation Manager shall support simulations where different hardware components are faulty or where faults occur stochastically

Metric name	The Optimization Simulator enables simulations that describe realistic scenarios.
Verified requirements	CRD-15, CRD-16, CRD-19, CRD-88, CRD-90, **CRD-42**
Maturity level	ML4

Steps to perform	1. Define the simulation using the Simulation Manager: • Composition of the swarm – number of members, list of behaviors they can perform • Structure of swarm members – capabilities, hardware components • The model of the environment used for the simulation • Stochastic description of the occurrence of hardware faults 2. Start the simulation using the Optimization Simulator.
Expected results	The Optimization Simulator simulates the scenario described by the environment model, swarm composition, behaviors and malicious events such as hardware faults. The Optimization Simulator can feed back the simulated input data collected by the swarm members into the Optimization Tool.

Metric name	The Optimization Simulator creates and passes the fitness score to the Optimization Tool.
Verified requirements	CRD-17, CRD-18
Maturity level	ML2
Steps to perform	1. Start the optimization using the Optimization Tool and the Optimization Simulator together.
Expected results	After each of the iterations that simulate the behavior generated by the Optimization Tool, the Optimization Simulator calculates a fitness score describing it and passes it to the Optimization Tool.

3.5 Code Generation Tool

Algorithms designed and optimized through the CPSwarm components located at the higher logic-levels of the CPSwarm Workbench will finally be deployed on real-world CPS systems, e.g., robotic platforms. Optimized algorithms cannot be directly deployed on a target CPS as, on one hand, they are developed and optimized to be portable across platforms, and on the other hand, they are typically evolved in a behavior / swarm-centric manner, with less focus on platform-related details such as event delivery subsystems, sensor communication interfaces, etc.

List of Requirements for the Code Generation Tool
CRD-63 The Code Generator shall generate code that is readable and understandable by humans.
CRD-94 The Code Generator shall receive the model of the high-level behaviour as a state machine, with additional information passed about each state to define the inputs and outputs of the low-level behavior

that is being executed while that state is active
CRD-96 The Code Generator shall be configured to produce code for a specific platform.
CRD-97 The Code Generator shall integrate low-level behavior algorithms generated by the Optimization Tool
CRD-102 The Code Generator shall integrate low-level behavior algorithms implemented manually

Metric name	The Code Generator shall generate code for a multi-level state machine incorporating inputs from the Modelling Tool, the Optimization Tool and the user
Verified requirements	CRD-94, CRD-97, CRD-102
Maturity level	ML1
Steps to perform	1. Set up a project where some of the states in the behavior are from the Modelling Library, others are generated by the Optimization Tool and others are stubbed out and left for the user to implement 2. Have the Optimization Tool generate its own code, then implement the stubbed out states in order to produce valid code the Code Generator can integrate 3. Run the Code Generator
Expected results	The Code Generator should generate code that is a valid state machine and can call the implementations supplied by the Optimization Tool and the user
Notes	Relies on other workbench components to build the behavior.

Metric name	The code generated by the Code Generator is tidy and readable
Verified requirements	CRD-63
Maturity level	ML2
Steps to perform	Perform the steps in the test "The Code Generator shall generate code for a multi-level state machine incorporating inputs from the Modelling Tool, the Optimization Tool and the user"

Expected results	The generated code should have consistent formatting and naming conventions. Comments should be present to describe, at the least, each function, global variable and class.

Metric name	The Code Generator can target multiple platforms
Verified requirements	CRD-96
Maturity level	ML2
Steps to perform	Perform the steps in the test "The Code Generator shall generate code for a multi-level state machine incorporating inputs from the Modelling Tool, the Optimization Tool and the user" for at least two different hardware platforms
Expected results	For both platforms, the generated code is valid and can be deployed.

3.6 Deployment Tool

After the code is successfully generated, it must be deployed on different targets. To ease the efforts to execute and manage the deployment to a group of heterogeneous devices, the Deployment Tool automates the process according to the configuration provided by the system users. The initial design of the Deployment Tool offers an over-the-air (OTA) update mechanism to deliver software to swarm members on-the-go and at large scale.

List of Requirements for the Deployment Tool
CRD-58 The Deployment Tool shall deploy artefacts on swarm members
CRD-59 The Deployment Agent shall report the deployment status
CRD-60 The communication between the Deployment Agent running on swarm members and the Deployment Manager shall be authenticated, authorized, encrypted, and integrity checked
CRD-61 The Deployment Manager shall receive the configuration of the deployment task from the operator prior to deployment
CRD-72 The Deployment Manager shall sign all packages with an operator specific key
CRD-73 The Deployment Tool shall implement secure over-the-air update functionality.
CRD-75 The Deployment Agent shall verify the signatures of packages on boot and when updates are received
CRD-76 The Deployment Manager shall provide a way to generate, import and export operator specific keys for code signatures
CRD-78 The Deployment Agent shall use the list of trusted certificates supplied when the device is first

provisioned to validate signatures
CRD-79 The Deployment Agent shall be responsible for starting, stopping and monitoring the code that has been deployed, even during startups and shutdowns
CRD-103 The Deployment Tool shall provide the means to compile codes on target platforms
CRD-104 The Deployment Tool shall provide the means to cross-compile codes for the target platforms
CRD-105 The Deployment Tool shall provide the means to compile codes

Metric name	The Deployment Tool can deploy a new behavior on a swarm member
Verified requirements	CRD-58, CRD-59, CRD-61, CRD-79, **CRD-51**
Maturity level	ML1
Steps to perform	1. Start the Deployment Tool 2. Wait until the tool indicates that it has completed the enumeration or at most 1 minute 3. Select a swarm member 4. Initiate the deployment of a behavior package
Expected results	The Deployment Tool shows the progress of the deployment process, which ends successfully. The new behavior can be observed as active on the swarm member.
Notes	Relies on other workbench components to build the behavior.

Metric name	Deployed software artefacts are signed and their signatures are verified	
Verified requirements	CRD-72, CRD-73, CRD-75, CRD-76, CRD-78	
Maturity level	ML3	
Steps to perform	Positive test	Negative test
	1. Set up the trust relationship between the swarm members and the Deployment Tool 2. Perform a deployment as described in the test "The	1. Break or do not set up the trust relationship between the swarm members and the Deployment Tool. 2. Perform a deployment as

	Deployment Tool can deploy a new behavior on a swarm member" 3. While the swarm member is inactive, corrupt the signature of the software package 4. Start the swarm member	described in the test "The Deployment Tool can deploy a new behavior on a swarm member"
Expected results	The deployment itself should be successful. When the swarm member is activated after the signature has been corrupted, it should refuse to start its behavior and shut down immediately.	Deployment should fail.
Notes	For platforms requiring compilation on the device, the positive test should not test the effects of corrupted signatures, since no signature should be present on the final executable.	

Metric name	The Deployment Tool can compile code before deployment
Verified requirements	CRD-104, CRD-105
Maturity level	ML2
Steps to perform	Perform the steps of the test "The Deployment Tool can deploy a new behavior on a swarm member" with a package and platform combination that requires cross-compilation.
Expected results	The Deployment Tool should show the results of the compilation before deployment has begun.
Notes	Relies on other workbench components to build the behavior.

Metric name	The Deployment Tool can compile code after deployment
Verified requirements	CRD-103, CRD-105
Maturity level	ML2

Steps to perform	Perform the steps of the test "The Deployment Tool can deploy a new behavior on a swarm member" with a package and platform combination that requires on device compilation.
Expected results	The Deployment Tool should show the results of the compilation after deployment has begun.
Notes	Relies on other workbench components to build the behavior.

Metric name	The Deployment Tool and the Deployment Agent communicate over a secure channel
Verified requirements	CRD-60, CRD-73
Maturity level	ML3
Steps to perform	1. Start capturing swarm communications 2. Perform the steps of the test "The Deployment Tool can deploy a new behavior on a swarm member" 3. Stop capturing swarm communications 4. Analyze the captured packets
Expected results	The captured exchange meets state of the art cryptographic requirements.
Notes	This test is not as exact as most other tests. Analysis should focus on ensuring that no parts of the deployment package are transmitted without encryption and that all the necessary authentication handshakes take place. The test should be repeated at various stages of the established trust relationship to see if authentication fails if it is required to fail.

3.7 Abstraction Layer

To ease the process of generating code to be deployed on target CPS, the CPSwarm project defines a so-called CPS abstraction layer whose purpose is to decouple the implementation of swarm algorithms from platform / system-specific function calls and primitives. The CPSwarm abstraction layer is composed by a set of platform-specific libraries that provide a common, high-level API that enables generated programs to uniformly interact with concrete CPS functions and subsystems. Depending on the CPS nature and operating environment the abstraction layer might be implemented as a shared library, as adaptation middleware and so on. Several different implementations are foreseen mainly including the actual platforms considered by the project: STEM educational robots, ROS-powered drones and rovers, and automotive fog nodes.

List of Requirements for the Abstraction Layer

CRD-83 The Abstraction Layer shall have low level support for remote shutdown requests that work regardless the status of the current behavior
CRD-85 The Abstraction Layer shall implement a hardware specific safe remote shutdown behavior that cannot be overridden by the current behavior (hard shutdown)
CRD-98 The Abstraction Layer shall provide APIs to access/control/set-up sensors and actuator on CPSs
CRD-99 The Abstraction Layer shall provide primitives to activate and control high-level CPS routines

Metric name	Remote soft shutdown requests are handled by the Abstraction Layer if the behavior has no handler for them
Verified requirements	CRD-83
Maturity level	ML2
Steps to perform	1. Set up a swarm where members are running a behavior with no soft shutdown request handler 2. Start the Monitoring and Configuration Tool 3. Wait until the tool indicates that it has completed the enumeration or at most 1 minute 4. Issue a remote soft shutdown request to a swarm member
Expected results	The swarm member shuts down safely.
Notes	Relies on other workbench components to set up the swarm and issue the request.

Metric name	Remote soft shutdown requests are passed to the behavior by the Abstraction Layer
Verified requirements	CRD-83
Maturity level	ML3
Steps to perform	1. Set up a swarm where members are running a behavior which handles soft shutdown request in a distinctive manner 2. Start the Monitoring and Configuration Tool 3. Wait until the tool indicates that it has completed the enumeration or at most 1 minute 4. Issue a remote soft shutdown request to a swarm member

Expected results	The swarm member shuts down safely, in a manner consistent with the behavior specified.
Notes	Relies on other workbench components to set up the swarm and issue the request.

Metric name	Remote hard shutdown requests are handled by the Abstraction Layer
Verified requirements	CRD-83
Maturity level	ML2
Steps to perform	1. Start the Monitoring and Configuration Tool 2. Wait until the tool indicates that it has completed the enumeration or at most 1 minute 3. Issue a remote hard shutdown request to a swarm member
Expected results	The swarm member shuts down safely.
Notes	Relies on other workbench components to set up the swarm and issue the request.

Metric name	If the behavior is unresponsive, the Abstraction Layer translates the soft shutdown request into a hard shutdown request
Verified requirements	CRD-83, CRD-85
Maturity level	ML4
Steps to perform	1. Set up a swarm where members are running a purposefully unresponsive behavior 2. Start the Monitoring and Configuration Tool 3. Wait until the tool indicates that it has completed the enumeration or at most 1 minute 4. Issue a remote soft shutdown request to a swarm member
Expected results	The swarm member shuts down safely, in a manner consistent with how hard shutdown requests should be handled.

Notes	Relies on other workbench components to set up the swarm and issue the request.

Metric name	Number of sensors and actuators supported by the Abstraction Library				
Verified requirements	CRD-98				
Measurement	Sensors and actuators should be visible in the Modelling Library as building blocks. Count the number of building blocks that reference sensors and actuators.				
Target values	ML1	ML2	ML3	ML4	ML5
	3	5	8	11	14

Metric name	Number of high-level CPS routines supported by the Abstraction Library				
Verified requirements	CRD-99				
Measurement	High-level CPS routines should be visible in the Modelling Library as building blocks. Count the number of building blocks that reference behaviors implemented by the Abstraction Library.				
Target values	ML1	ML2	ML3	ML4	ML5
	1	2	3	4	5

3.8 Monitoring Tool

The Monitoring and Configuration Tool is responsible for the runtime configuration and reconfiguration of single CPS and multiple CPSs (CPS swarms), as well as for monitoring the critical system and mission parameters.

List of Requirements for the Monitoring and Configuration Tool
CRD-36 The Modelling Tool shall provide the type of swarm member, type of data and data source to the monitoring tool

CRD-37 The Monitoring and Configuration Tool shall provide the type and address of swarm member
CRD-89 The Monitoring and Configuration Tool shall be able to trigger remote events on individual swarm members
CRD-92 The Monitoring and Configuration Tool shall enable the user to launch an external tool to take remote control of a specific swarm member
CRD-93 The Monitoring and Configuration Tool shall be able to monitor events in all scopes as they are being triggered by or received on a swarm member

Metric name	The Monitoring and Configuration Tool can enumerate the members of a swarm
Verified requirements	CRD-36, CRD-37, **CRD-39, CRD-45**, **CRD-46**
Maturity level	ML1
Steps to perform	1. Start the Monitoring and Configuration Tool 2. Wait until the tool indicates that it has completed the enumeration or at most 1 minute
Expected results	The Monitoring and Configuration Tool shows all active swarm members.
Notes	The Monitoring and Configuration Tool should be started on a system that has already established a connection with the swarm or on a system that is capable of establishing such a connection using the features built into the tool itself. The swarm should have at least one active member.

Metric name	The Monitoring and Configuration Tool can enumerate properties of a swarm member
Verified requirements	CRD-36, CRD-37, **CRD-39, CRD-45**, **CRD-46**
Maturity level	ML2
Steps to perform	1. Perform the steps as defined in the test "The Monitoring and Configuration Tool can enumerate the members of a swarm" 2. Query one of the swarm members for its properties
Expected results	The Monitoring and Configuration Tool shows all properties of the swarm member, including the type of the property and whether it is read-only or writable.

Metric name	The Monitoring and Configuration Tool can issue commands to individual swarm members
Verified requirements	CRD-89, **CRD-41, CRD-43**, **CRD-44**, **CRD-45**
Maturity level	ML2
Steps to perform	1. Perform the steps as defined in the test "The Monitoring and Configuration Tool can enumerate the members of a swarm" 2. Issue a command to one of the swarm members
Expected results	The swarm member reacts to the command and performs the associated action.

Metric name	The Monitoring and Configuration Tool can enable the user to launch an external tool to take direct control of a swarm member
Verified requirements	CRD-92, **CRD-41**
Maturity level	ML4
Steps to perform	1. Perform the steps as defined in the test "The Monitoring and Configuration Tool can enumerate the members of a swarm" 2. Issue a request to take remote control of a swarm member 3. Wait until the response of approval and then launch the external tool
Expected results	An external tool is launched and can be used to control the swarm member directly.
Notes	Not all swarm members need to be compatible with this feature. Ensure that the selected swarm member has an associated external control tool and that handover is enabled on the device.

Metric name	The Monitoring and Configuration Tool can observe events as they happen on swarm members
Verified requirements	CRD-93, **CRD-39, CRD-45**, **CRD-46**
Maturity level	ML3
Steps to perform	1. Perform the steps as defined in the test "The Monitoring and Configuration Tool can enumerate the members of a swarm" 2. Trigger an event on one of the swarm members manually

Expected results	The Monitoring and Configuration Tool show the event as it happens.
Notes	A special behavior on the swarm member might be necessary to perform this test. The test should be repeated for each event scope to ensure that all event scopes are monitored correctly.

4 User Experience

These high-level user experience requirements are verified by the test cases defined for their relevant components (included in the list of verified requirements in bold):

List of Requirements for User Experience
CRD-39 The Swarm Operator should be able to monitor the swarm
CRD-41 The Swarm Operator should be able to change the mission on the go
CRD-42 Environment conditions should be simulated
CRD-43 The Mission Planner should be able to configure a mission
CRD-44 The Mission Planner should be able to start a mission
CRD-45 The Mission Planner would like to have a UI to configure a mission
CRD-46 The Swarm Operator would like to have a UI to monitor the swarm in play
CRD-47 The swarm can have heterogeneous or a homogeneous composition
CRD-48 The Swarm Designer should be able to define the composition of the swarm
CRD-51 The Swarm Designer should be able to assign role to swarm member

These requirements have no functional equivalents and as such will not be verified by any of the test cases defined:

List of Requirements for User Experience
CRD-38 The swarm should consist of self-organizing swarm members
CRD-49 All the swarm members of a swarm should act under only one mission at a time
CRD-50 The Mission Planner should be able to add constraints to a mission

5 Scenarios

Instead of defining a separate set of requirements for each vision scenario, use case partners have been active all the way through the process of defining the requirements for the whole workbench and have contributed their input and their insights to the requirements for the components. The requirements thus presented represent the complete set required to use the workbench in each of the vision scenarios.

6 Milestones

As defined in the project proposal, the CPSwarm project has three major phases – synchronized with the three years of the project. For Phase 1, we can already evaluate the maturity levels of the components using the metrics defined in this document, and for Phase 2 and Phase 3, we can plan ahead for each component to set realistic targets that can be met while the project progresses.

Since Phase 1 only includes a subset of the components being developed by the project, the rest were excluded from this first evaluation. Phase 1 components were evaluated as-is in M18 and this deliverable presents their maturity at the time this deliverable has been finalized.

		MS5 – Phase 1 (current)	MS9 – Phase 2 (planned)	MS13 – Phase 3 (planned)
Components	Modelling Tool	ML1	ML2	ML5*
	Modelling Library	ML1	ML2	ML5
	Optimization Tool	ML1	ML2	ML5*
	Simulation Tool	ML1	ML2	ML5*
	Code Generation Tool		ML2	ML5*
	Deployment Tool		ML2	ML5*
	Hardware Abstraction Layer		ML2	ML5
	Monitoring Tool		ML2	ML5
Project		ML1	ML2	ML5

Levels marked with an asterisk have no associated tests defined due to lack of more specific requirements – as the project progresses, new requirements will have to be defined (with respective test cases) to measure the progress of these components on higher maturity levels. This deliverable will be updated during the project progress and new results will be documented in the upcoming use case validation deliverables: D8.7 – Initial Validation Results and D8.8 – Final Validation Results.

Acronyms

Acronym	Explanation
KPI	Key Performance Indicator
ML	Maturity Level
CPS	Cyber-Physical System
OTA	Over-The-Air
UI	User Interface

List of figures

List of tables

References

[1] «D3.1 Initial System Architecture Analysis and Design Specification,» The CPSwarm Project.