D2.7 – FINAL LESSONS LEARNED AND REQUIREMENTS REPORT

Deliverable ID	**D2.7**
Deliverable Title	**Final Lessons Learned and Requirements Report**
Work Package	**WP2**
Dissemination Level	**PUBLIC**
Version	**1.1**
Date	**2019-08-02**
Status	**Final**
Lead Editor	**FRAUNHOFER**
Main Contributors	Sarah Suleri, René Reiners, Farshid Tavakolizadeh **(FRAUNHOFER),** Etienne Brosse **(SOFTEAM),** Melanie Schranz **(LAKE),** Arthur Pitman **(UniKLU)**, Judit Torma **(SLAB),** Davide Conzon **(LINKS),** Andreas Eckel **(TTTECH, TTA)**

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	2018-08-31	Sarah Suleri (FRAUNHOFER)	First Draft with TOC
0.2	2019-01-16	Sarah Suleri (FRAUNHOFER)	Section 4.1 (Non-functional Requirements)
0.3	2019-01-28	Etienne Brosse (Softeam)	Section 5.1, 5.2
		Davide Conzon (LINKS)	Section 5.4, 5.5, 5.6
		Arthur Pitman (UniKLU)	Section 5.3
		Farshid Tavakolizadeh (FRAUNHOFER)	Section 5.7, 5.9
0.4	2019-02-04	Andreas Eckel (TTTech)	Section 5.8
		Judith Torma (Slab)	Section 4.2
		Rene Reiners (FRAUNHOFER)	Section 6
0.5	2019-02-05	Sarah Suleri (FRAUNHOFER)	Accommodating changes suggested by internal review 1.
1.0	2019-02-14	Sarah Suleri (FRAUNHOFER)	Accommodating changes suggested by internal review 2.
1.1	2019-08-02	René Reiners (FRAUNHOFER)	Integrated Appendix A

Internal Review History

Review Date	Reviewer	Summary of Comments
2019-02-12	Angel Soriano (ROBOTNIK)	Minor changes
2019-02-05	Melanie Schranz (LAKE)	Minor changes

Executive Summary

The present document is a deliverable of the CPSwarm project, funded by the European Commission's Directorate-General for Research and Innovation (DG RTD), under its Horizon 2020 Research and innovation program (H2020), reporting the results of the activities carried out by WP2 – Use cases, requirements engineering and business models. The main objective of the CPSwarm project is to develop a workbench that aims to fully design, develop, validate and deploy engineered swarm solutions. More specifically, the project revolves around three vision scenarios; Swarm Drones, Swarm Logistics Assistant and Automotive CPS. The scenarios were outlined in the proposal and are refined within the engineering efforts alongside the project, driven by WP2.

WP2 manages and undertakes the work of carrying out the iterative engineering of requirements, which focuses on the engineering process of initial requirements and reengineering after the end of each iteration cycle. The purpose of this work package is thus to maintain a continuous discovery and analysis of user centric requirements, needs and prospects, to be used in the design, development, implementation and validation of the CPSwarm workbench.

The main objective of this deliverable is to describe the reiteration of the requirements elicited and documented in D2.3 and D2.6. The goal of this document is to define a list of CPSwarm requirements exploiting the "Volere" approach. These requirements have been continuously updated and refined through an iterative process that lead to the production of a total of three releases of this document, due respectively in Project Months M6, M14 and M26. Additionally, D2.7 also documents lessons learned during design and development of various components of the CPSwarm workbench.

Furthermore, this deliverable formulates the foundation for the validation results to be specified in D8.8 and final system architecture analysis and design specifications to be documented in D3.3 in WP3, and later for the remaining technical WPs (WP3 up to WP7), towards the demonstration (WP8).

Table of Contents

1 Introduction

This deliverable documents the results of Task 2.1 *Vision scenarios, use cases and initial requirements*. The purpose of this deliverable is to refine **user needs** and **technical requirements** identified and described in D2.3 *Initial requirements report* and D2.6 *Initial Lessons Learned and Requirements Report.*

This document describes the activities to support the identified workbench workflow, adapting it to the different environments involved in the CPSwarm project and provides a thorough analysis of the requirements. These high-level requirements will guide the development phases within the technical work packages, and therefore, this deliverable will be a common reference point for the CPSwarm consortium with relevance to architectural (WP3) questions and impacts on implementation (WP7 and WP8) as well as exploitation (WP9) efforts.

The **main objectives** of the activities that were performed by Task T2.1 so far are listed in the following:

- Requirements and user needs reiteration
- Documentation of lessons learned

Over time, while documenting and requirements formulated in this deliverable, demands for **safety requirements** arose from demos, during reviews and from general feedback. Especially for the implementation tasks in WP8, knowledge of norms, rules and regulations is necessary for designing a cyber-physical system within the actual domains. Since most work is put into role process and requirements definition for **constructing and using** the CPSwarm Workbench, we stick to this primary field.
In addition, this document provides an overview of rules and norms that need to be taken care of when creating the concept of a CPS. These kinds of norms and rules must be provided by swarm and application designers and be available during design time. This kind of knowledge is not directly part of the requirements to be collected in this document. Still, the need for being aware of this knowledge is a requirement and thus, norms and rules are listed as one extensible library of the CPSwarm workbench.

Appendix A summarizes the application experts' knowledge as a table with references to adequate norms and rules.

The development of this deliverable was coordinated by FRAUNHOFER with contribution of SOFTEAM, LAKE, SearchLab, LINKS, UniKLU and TTTech. The outcome of this deliverable will be used for deliverable D3.3: *Final System Architecture Analysis & Design Specification* and D8.8: *Final Validation Results*, due in M30 and M36 respectively.

1.1 Related documents

ID	Title	Reference	Version	Date
D2.1	Initial Vision Scenarios and Use Case Definition		2.0	M4
D2.3	Initial Requirements Report		1.0	M6
D3.1	Initial System Architecture Analysis & Design Specification		1.0	M6
D2.6	Initial Lessons Learned and Requirements Report		1.0	M14
D2.2	Final Vision Scenarios and Use Case Definition		1.0	M16

D3.2	Updated System Architecture Analysis & Design Specification		1.0	M18
D2.8	Validation Framework Specification		1.0	M18
D3.3	Final System Architecture Analysis & Design Specification		1.0	M30
D8.8	Final Validation Results		1.0	M36

2 Approach and Methodology

As depicted in Figure 1, the development cycle for the CPSwarm Workbench starts from the top left with a scenario thinking methodology accompanied by collecting other kinds of input such as related work, documents, standards or available technologies. Once some understanding of the context has been reached, requirements are derived from it. These requirements, especially in the beginning, take the form of user requirements, i.e. what the user needs from the system. When the system starts to take a concrete shape, these user needs are transformed into technical requirements, i.e. what the system must offer or how the architecture should look like.

In long-term iterations, system design, integration of technologies and knowledge as libraries take place that are then implemented in an incremental manner and validated later on. The results from the validation are then fed back into the scenarios and collection of available knowledge base. New findings, corrections and additions are then incorporated into the existing documents and requirements as well as ideas for innovations are updated. This way, the cycle starts again, affecting all technical developments, which, in the end, are validated again. This methodology allows for step-wise knowledge acquisition and development allowing for adjustments alongside conception and development.

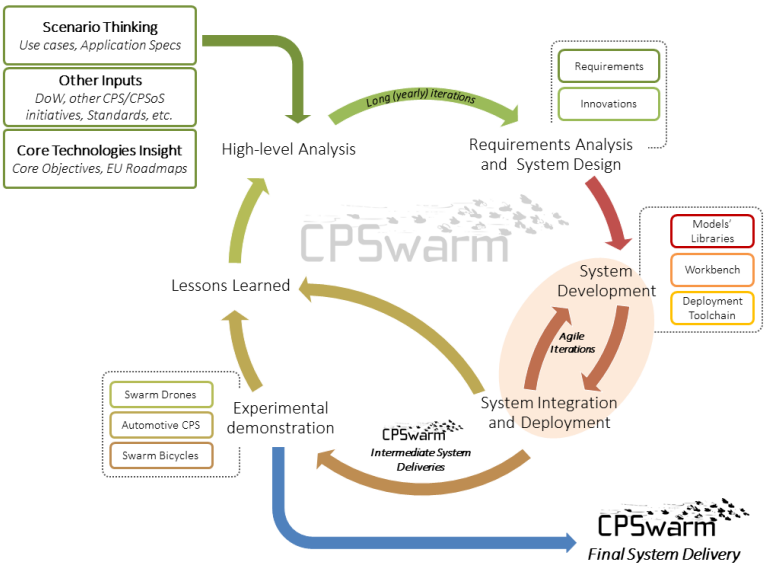

Figure 1: The CPSwarm Workbench development lifecycle

The work reported in this deliverable is located in the top right corner and follows a user centred approach for requirements elicitation. This document also enlists the lessons learned during the system development cycle shown in the bottom right corner. The document's structure is as follows:

- **Chapter 3** describes the Volere requirements approach followed throughout this deliverable for requirement reiteration. This section explains various attributes of the Volere Requirement Shell and additionally, explains the adaptation of this shell used for CPSwarm requirement specification. It also explains details of online support provided for requirement specification and management.

- **Chapter 4** enlists the updated requirements

- **Chapter 5** describes the lessons learned and recommendations related to each component of the CPSwarm workbench.

- **Chapter 6** explains the design pattern library, its current state and plans to proceed further.

3 Requirements Engineering Approach

CPSwarm is using the Volere Requirements approach described by Robertson and Robertson (cf. Ref. [9] [10] [11]). Volere is a proven and widely used general-purpose approach to requirements elicitation, including both the process of eliciting requirements as well as the format for representing them. Section 3.1 provides an overview of key elements of the Volere approach.

3.1 Volere Requirements Approach

The Volere requirements approach is described by Robertson and Robertson (cf. Ref [9] [10]). There is a web-site dedicated to the Volere approach as well: http://www.volere.co.uk/. One of the various resources available on this site is the "Volere Requirements Specification Template" (cf. Ref [11]) known as the "Requirements Shell". This format is further explained in subsection 3.1.1.

3.1.1 Requirements Shell

Figure 2 reproduces the Volere "Requirements Shell" by Robertson and Robertson (cf. Ref [11]). While the "Requirements Shell" mimics an index card, it is meant as the definition of a representational format that should be used with appropriate technical support for authoring requirements.

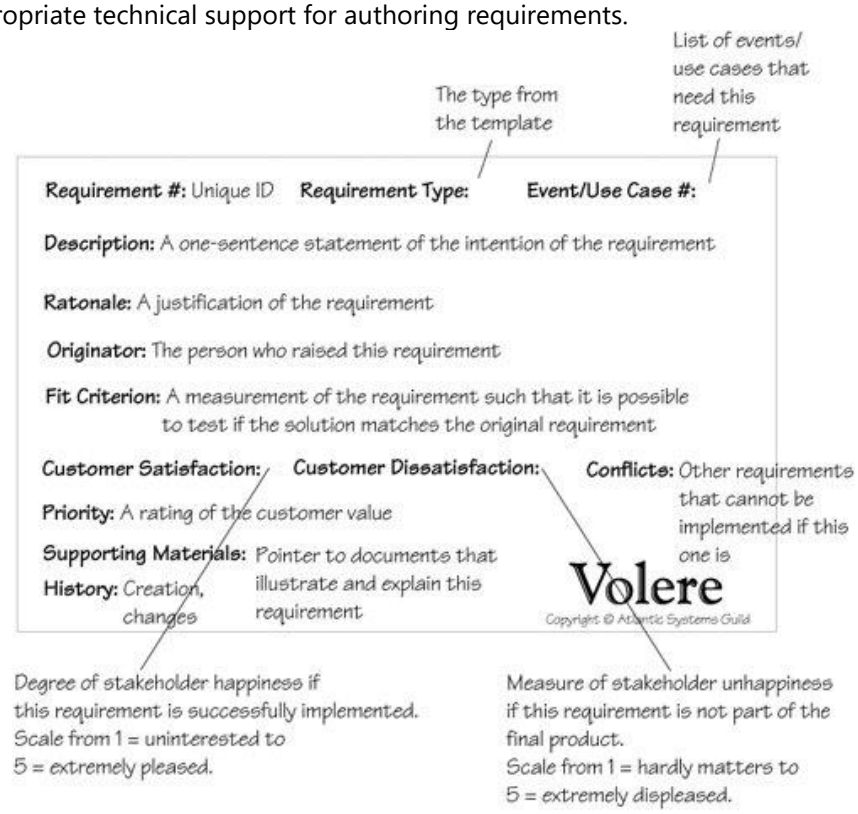

Figure 2: The Volere "Requirements Shell" for representing atomic requirements [11]

3.2 Requirements Management

For the creation and management of information elements of a design process, a number of different approaches have been suggested by Stufflebeam et al. [12] and Penna et al. [7]. In the authors' experience most tools that go beyond Microsoft (MS) Word and Excel have little prospect of being used on a broad basis among a heterogeneous group of partners in international R&D projects. While MS Word and Excel are

certainly adequate for representing a set of user needs or requirements, they have not proven effective in sustainably supporting a continuous and iterative design process.

As mentioned in D2.3, we used GitLab Issue Tracker for requirement specification and management. For the initial specification of user needs and requirements with limited attributes, the GitLab Issue tracker sufficed but for the reiteration of user needs and requirements, the GitLab could not support all the attributes of the Volere requirements shell. Therefore, we changed the platform to JIRA for requirement specification and management. The CPSwarm JIRA project space can be found at the following address and is hosted by Fraunhofer:

https://jira.fit.fraunhofer.de/jira/projects/CRD/issues/

Two general issue types are currently available in the JIRA space of the CPSwarm Requirements and Development project.
The first is a "**User Need**" that documents user stories based on the information acquired from the requirement engineering workshops conducted with our application partners. The creation dialog for user needs in JIRA is shown in Figure 3.

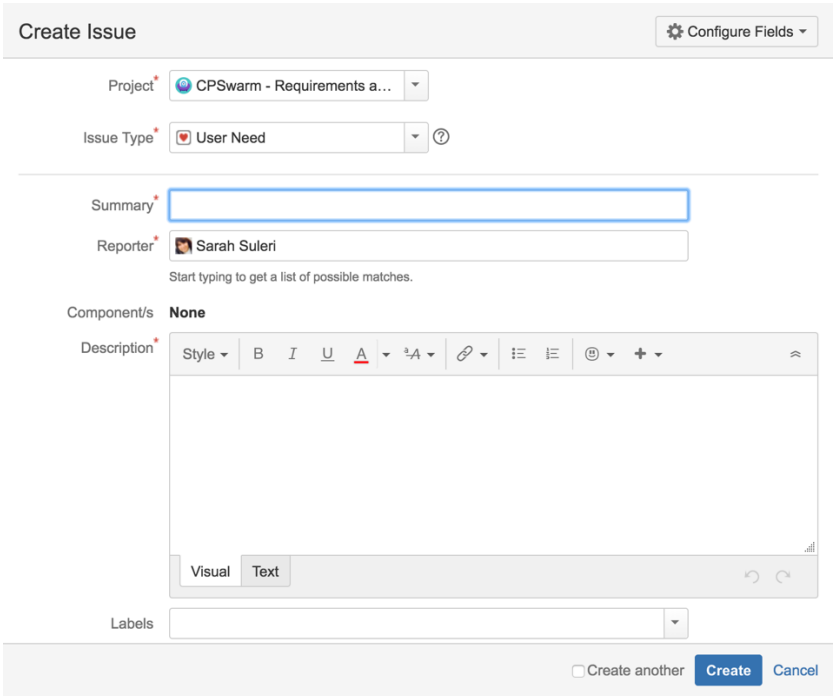

Figure 3: Screenshot of the user need creation dialog in CPSwarm

The second is a "**Volere Requirement**", which is used to describe various capabilities of different identified components of the workbench. This type of requirements also defines the data flow between these components. Figure 4 shows a screenshot of the Volere Requirement creation dialog in JIRA.

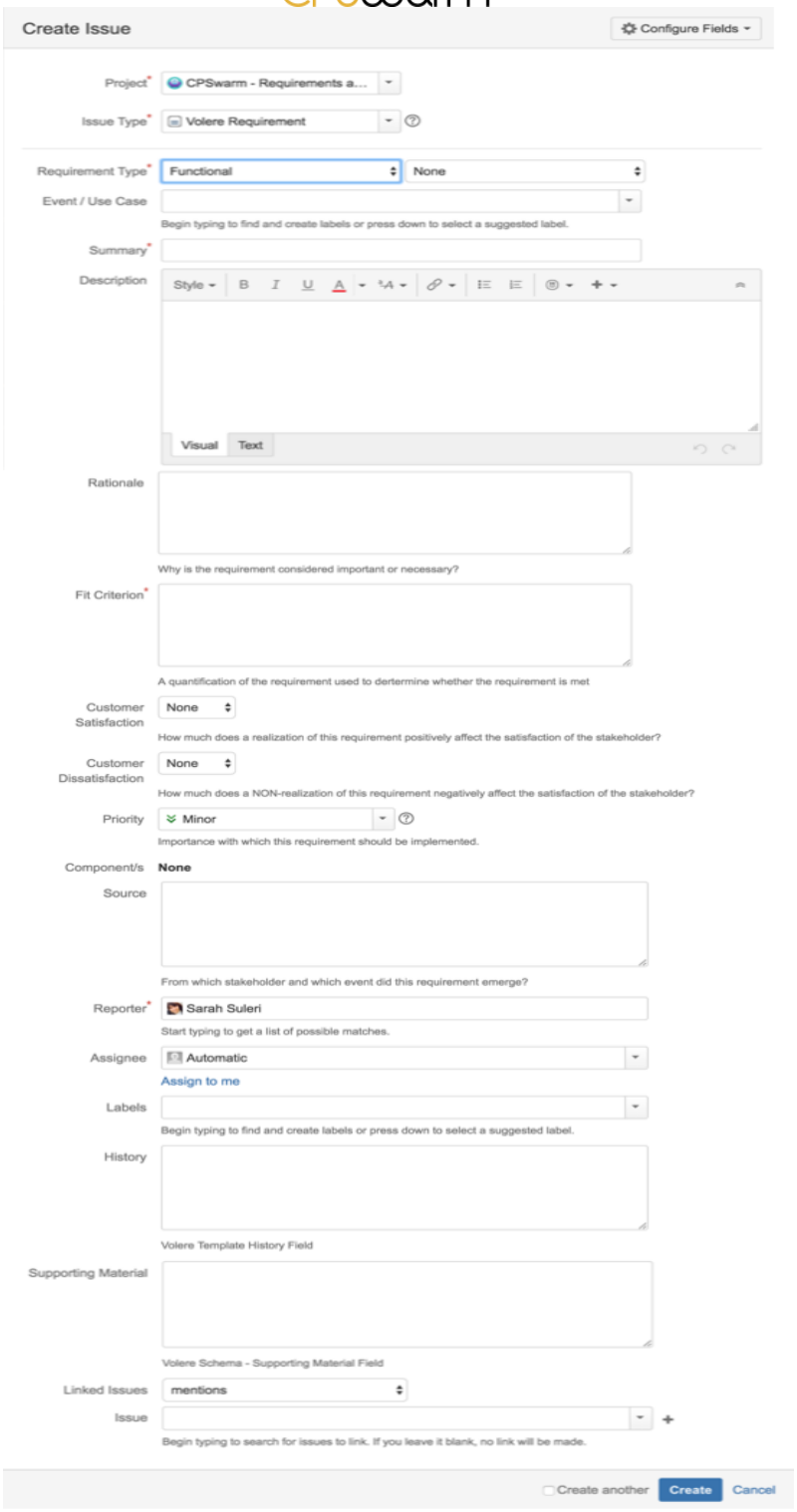

Figure 4: Screenshot of the Volere Requirement creation dialog in CPSwarm

A state diagram defining possible states of an issue and appropriate transitions between states has been implemented for both types of issues, "User Need" and "Volere Requirement", in the CPSwarm requirements engineering process. The state diagrams are shown in Figure 5 and Figure 6.

In particular Figure 5 shows that as soon as a user need has been created it is in an "open" state. After an "open" user need has passed the quality check should be set to "quality check passed" state. The state

"duplicate" can be assigned from all states (here the two mentioned above) and means that this user need actually is redundant to another user need, i.e. it duplicates another existing issue.

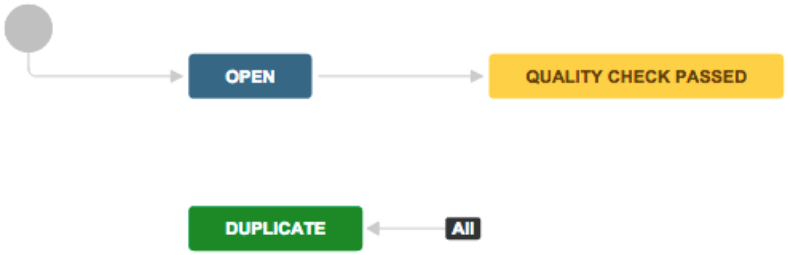

Figure 5: State diagram of issue type "User Need"

Logic depicted in Figure 6 is initially the same as for the previous description. Once the quality check is passed for a requirement, it can become a part of the specification. After the implementation is complete, it acquires the status of "implemented". After that, it is validated. This three step process is iterative. The figure also shows that once a requirement is open, it can also be rejected based on a legitimate reason.

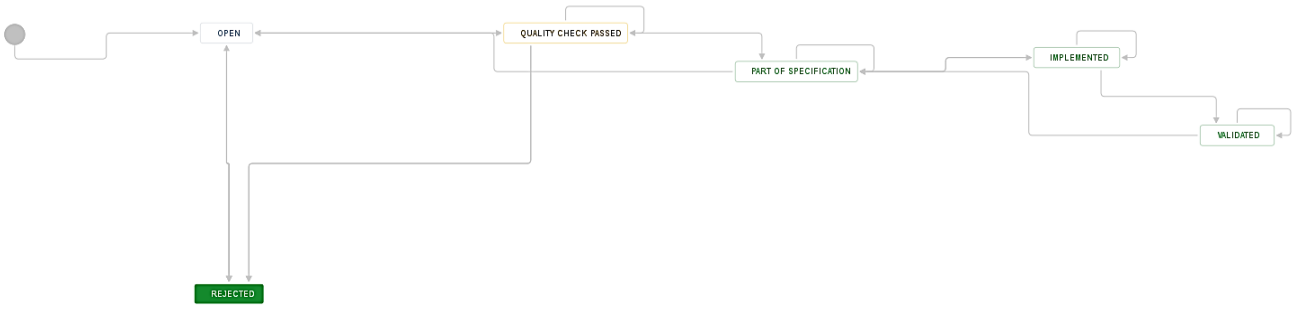

Figure 6: State diagram of issue type "Volere Requirement"

4 CPSwarm Requirement Specification

As mentioned in the previous section, there are two types of requirements elicited for CPSwarm; User Needs and Volere Requirements. User needs are explained from the perspective of various user roles. Whereas, the Volere requirements are described from the perspective of various components of the workbench. In this last iteration of requirements, we have enhanced the requirements collection by adding non-functional requirements.

4.1 Non-functional Requirements

The initial set of technical requirements are documented in D2.3 and D2.6. Below are the details of reiterated Volere requirements.

[CRD-115] The system shall be able to show/visualise relevant information in an understandable manner			
Description:	The requirement is related to the need of the user to have a complete overview of the current situation.		
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Major
Reporter:	Sarah Suleri	**Assignee:**	Miguel Cantero
Labels:	NFR		
Requirement Type:	Non-Functional - Usability and Humanity		
Rationale:	This is essential for the usability of the system.		
Fit Criterion:	The user can understand the relevant information shown by the system.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	Very high		
Source:	User Needs Analysis		

[CRD-116] System should provide guides and other material for training of users			
Description:	Guides and material should be directed at different users, e.g., detailed guides for professional users and simpler instructions for novices. This could be as Frequently Asked Questions.		
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Major
Reporter:	Sarah Suleri	**Assignee:**	Angel Soriano
Labels:	NFR		
Requirement Type:	Non-Functional - Usability and Humanity		
Rationale:	It is important that the different types of users can learn to use the system easily and correctly.		
Fit Criterion:	Guides, instructions and other material for training is provided.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	Neutral		
Source:	User Needs Analysis		

[CRD-117] All the components of the system shall be well integrated			
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Major
Reporter:	Sarah Suleri	**Assignee:**	Omar Morando
Labels:	NFR		
Requirement Type:	Non-Functional – Operational		
Rationale:	It is important that the various components of the system are coherent and well integrated to ensure an uninterrupted flow of activities.		
Fit Criterion:	All the components of the system are well integrated.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	High		
Source:	User Needs Analysis		

[CRD-118] The system should be able to interface and interoperate with existing systems			
Description:	The system must be easy to install and integrate with other software, equipment and systems.		
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Minor
Reporter:	Sarah Suleri	**Assignee:**	Omar Morando
Labels:	NFR		
Requirement Type:	Non-Functional - Maintainability and Support		
Rationale:	This is important to make the CPSwarm workbench easy to integrate with existing systems.		
Fit Criterion:	The system interfaces and interoperations with existing systems		
Customer Satisfaction:	Neutral		
Customer Dissatisfaction:	Neutral		
Source:	User Needs Analysis		

[CRD-119] Data processing and management must comply with relevant regulations			
Description:	Data processing must comply with GDPR and national regulations.		
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Critical
Reporter:	Sarah Suleri	**Assignee:**	Omar Morando
Labels:	NFR		
Requirement Type:	Non-Functional - Legal		
Rationale:	This is required to prevent misuse of data.		
Fit Criterion:	Data processing and management complies with GDPR and national regulations.		
Customer Satisfaction:	Very high		
Customer Dissatisfaction:	Very high		
Source:	User Needs Analysis		

[CRD-121] Any interface between the user and the platform must have a reasonable response time			
Description:	More concrete value shall be defined by pilot partners.		
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Major
Reporter:	Sarah Suleri	**Assignee:**	Omar Morando
Labels:	NFR		
Requirement Type:	Non-Functional - Performance		
Rationale:	Long response times can be very annoying for the users.		
Fit Criterion:	All interfaces have reasonable response times.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	High		
Source:	User Needs Analysis		

[CRD-122] The system shall be scalable to support massive growth in the number of users/devices, etc.			
Description:	The system should be easy to extend with additional (types of) devices, sensors, etc.		
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Medium
Reporter:	Sarah Suleri	**Assignee:**	Miguel Cantero
Labels:	NFR		
Requirement Type:	Non-Functional - Operational		
Rationale:	This is inherent in a large-scale system.		
Fit Criterion:	The system is scalable to support massive growth in the number of users/devices/etc.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	Neutral		
Source:	User Needs Analysis		

[CRD-123] The solution should be in compliance with GDPR as well as national policies			
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Critical
Reporter:	Sarah Suleri	**Assignee:**	Angel Soriano
Labels:	NFR		
Requirement Type:	Non-Functional - Operational		
Rationale:	All solutions must comply with national and international rules and regulations.		
Fit Criterion:	GDPR as well as national policies and regulations are considered.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	High		
Source:	User Needs Analysis		

[CRD-124] The system shall be evaluated by 85% of the professional users to be easy to use			
Description:	This relates to the overall evaluation of the system.		
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Medium
Reporter:	Sarah Suleri	**Assignee:**	Miguel Cantero
Labels:	NFR		
Requirement Type:	Non-Functional - Usability and Humanity		
Rationale:	It is important that the user interface is logical and intuitive.		
Fit Criterion:	85% of the professional users find the workbench easy to use.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	High		
Source:	User Needs Analysis		

[CRD-125] The system shall not generate additional workload for the professional users			
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Major
Reporter:	Sarah Suleri	**Assignee:**	Omar Morando
Labels:	NFR		
Requirement Type:	Non-Functional - Usability and Humanity		
Rationale:	It is important that the system is perceived as a help not as a burden.		
Fit Criterion:	The system does not generate additional workload for the professional users.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	High		
Source:	User Needs Analysis		

[CRD-126] Accessing sensitive data must be logged (User ID, Timestamp, etc.)			
Description:	Sensitive data as defined in GDPR.		
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Major
Reporter:	Sarah Suleri	**Assignee:**	Miguel Cantero
Labels:	NFR		
Requirement Type:	Non-Functional – Legal		
Rationale:	This is a legal requirement.		
Fit Criterion:	Accessing sensitive data is logged (User ID, Timestamp, etc.).		
Customer Satisfaction:	High		
Customer Dissatisfaction:	High		
Source:	User Needs Analysis		

[CRD-127] Attempts at accessing sensitive data by unauthorised users must be logged			
Description:	Surveillance of attempts at intrusion Should be achieved: • on the system level where sensitive data are stored (collection of system logs, evaluation and automatic prevention resp. alerting the owner of sensitive data repository) • on the network traffic level, it is possible to identify target of attempt by unauthorised user (Identity & Access Management, Intrusion Prevention resp. Anomaly Detection)		
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Medium
Reporter:	Sarah Suleri	**Assignee:**	Angel Soriano
Labels:	NFR		
Requirement Type:	Non-Functional - Security		
Rationale:	This is part of the built-in system security and privacy.		
Fit Criterion:	Attempts at accessing sensitive data by unauthorised users are logged.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	High		
Source:	User Needs Analysis		

[CRD-128] The system shall be protected against cyber attacks			
Description:	Protection against cyber-attack should be achieved: • on the system level (using anti-malware and system firewall) • within a network, on the traffic from/to the system (using firewalls and gateways, log collection and evaluation with automatic preventive measures [blocking] resp. at least alerting system owners about unusual behaviour)		
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Critical
Reporter:	Sarah Suleri	**Assignee:**	Omar Morando
Labels:	NFR		
Requirement Type:	Non-Functional - Security		
Rationale:	This is a must for online systems with sensitive data.		
Fit Criterion:	The system is protected against cyber attacks.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	High		
Source:	User Needs Analysis		

[CRD-129] The system shall not use picture icons that could be considered offensive in any country where the system is used			
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Medium
Reporter:	Sarah Suleri	**Assignee:**	Angel Soriano
Labels:	NFR		
Requirement Type:	Non-Functional - Look & Feel		
Rationale:	This is a must for interoperability.		
Fit Criterion:	The product does not use picture icons that could be considered offensive in any country where the system is used.		
Customer Satisfaction:	Neutral		
Customer Dissatisfaction:	High		
Source:	User Needs Analysis		

[CRD-130] Installing an upgrade shall not modify existing configuration values.			
Description:	An exception is made for any values that the new version uses in different ways from the previous version.		
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Nice to have
Reporter:	Sarah Suleri	**Assignee:**	Miguel Cantero
Labels:	NFR		
Requirement Type:	Non-Functional - Maintainability and Support		
Rationale:	This refers to the installability of the system. The motivation for this requirement is to avoid wasting the time of users who have spent considerable time configuring the system to suit themselves.		
Fit Criterion:	Installing an upgrade does not modify existing configuration values.		
Customer Satisfaction:	Neutral		
Customer Dissatisfaction:	High		
Source:	User Needs Analysis		

[CRD-131] When a new version of the main system is released, it shall be possible to upgrade to it from any previous version.			
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Nice to have
Reporter:	Sarah Suleri	**Assignee:**	Angel Soriano
Labels:	NFR		
Requirement Type:	Non-Functional - Maintainability and Support		
Rationale:	This refers to the installability of the system. The motivation for this requirement is to avoid wasting the time of users who have spent considerable time configuring the system to suit themselves.		
Fit Criterion:	New version of the main system can be upgraded from any previous version.		
Customer Satisfaction:	Neutral		
Customer Dissatisfaction:	Neutral		
Source:	User Needs Analysis		

[CRD-132] No piece of text that might be displayed to a user shall reside in source code.			
Description:	That is, every piece of text that a user might see must be modifiable without changing source code.		
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Nice to have
Reporter:	Sarah Suleri	**Assignee:**	Angel Soriano
Labels:	NFR		
Requirement Type:	Non-Functional - Maintainability and Support		
Rationale:	This refers to the modifiability of the system.		
Fit Criterion:	No piece of text that might be displayed to a user resides in source code.		
Customer Satisfaction:	Neutral		
Customer Dissatisfaction:	Neutral		
Source:	User Needs Analysis		

[CRD-133] The system shall not be shut down for maintenance more than once in a 24-hour period.			
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Medium
Reporter:	Sarah Suleri	**Assignee:**	Miguel Cantero
Labels:	NFR		
Requirement Type:	Non-Functional - Maintainability and Support		
Rationale:	This refers to the maintainability of the system.		
Fit Criterion:	The system does not shut down for maintenance more than once in a 24-hour period.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	Neutral		
Source:	User Needs Analysis		

[CRD-134] Provisions shall be made for the future usage of multiple languages.			
Description:	Provision shall include at least the following: 1)The structure of the data store shall be such that multi-lingual support shall not necessitate additional components or the need to replace current components, and 2) A user shall be able to nominate their preferred language when entering their personal information.		
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Medium
Reporter:	Sarah Suleri	**Assignee:**	Omar Morando
Labels:	NFR		
Requirement Type:	Non-Functional - Cultural and Political		
Rationale:	This refers to the flexibility of the system.		
Fit Criterion:	Provisions are made for the future usage of multiple languages.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	Neutral		
Source:	User Needs Analysis		

[CRD-135] The system shall be useable by users after nominal training.			
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Minor
Reporter:	Sarah Suleri	**Assignee:**	Omar Morando
Labels:	NFR		
Requirement Type:	Non-Functional - Usability and Humanity		
Rationale:	This refers to the learnability of the system.		
Fit Criterion:	The system is useable by users after nominal training.		
Customer Satisfaction:	Neutral		
Customer Dissatisfaction:	Neutral		
Source:	User Needs Analysis		

[CRD-136] People with no training and no understanding of English shall be able to use the product.			
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Minor
Reporter:	Sarah Suleri	**Assignee:**	Miguel Cantero
Labels:	NFR		
Requirement Type:	Non-Functional - Usability and Humanity		
Rationale:	This refers to the learnability of the system.		
Fit Criterion:	People with no training and no understanding of English are able to use the product.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	Neutral		
Source:	User Needs Analysis		

[CRD-137] The product shall be self-explanatory and intuitive			
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Minor
Reporter:	Sarah Suleri	**Assignee:**	Omar Morando
Labels:	NFR		
Requirement Type:	Non-Functional - Usability and Humanity		
Rationale:	This refers to the learnability of the system.		
Fit Criterion:	The product is self-explanatory and intuitive.		
Customer Satisfaction:	Neutral		
Customer Dissatisfaction:	Neutral		
Source:	User Needs Analysis		

[CRD-138] When an update failure is detected all updates performed during the failed session shall be rolled back to restore the data to pre-session condition.			
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Major
Reporter:	Sarah Suleri	**Assignee:**	Angel Soriano
Labels:	NFR		
Requirement Type:	Non-Functional - Operational		
Rationale:	This refers to the survivability of the system.		
Fit Criterion:	When an update failure is detected all updates performed during the failed session rolled back to restore the data to pre-session condition.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	High		
Source:	User Needs Analysis		

[CRD-139] All data recovered in a roll-back condition shall be recorded for use in forward recovery under user control.			
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Major
Reporter:	Sarah Suleri	**Assignee:**	Omar Morando
Labels:	NFR		
Requirement Type:	Non-Functional - Operational		
Rationale:	This refers to the survivability of the system.		
Fit Criterion:	All data recovered in a roll-back condition is recorded for use in forward recovery under user control.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	High		
Source:	User Needs Analysis		

[CRD-140] When operating after a failure the user shall be informed the application is operating in a “safe mode” and all data is available for review without update			
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Major
Reporter:	Sarah Suleri	**Assignee:**	Miguel Cantero
Labels:	NFR		
Requirement Type:	Non-Functional - Operational		
Rationale:	This refers to the survivability of the system.		
Fit Criterion:	When operating after a failure the user is informed, the application is operating in a “safe mode” and all data is available for review without update		
Customer Satisfaction:	High		
Customer Dissatisfaction:	High		
Source:	User Needs Analysis		

[CRD-141] The system shall prevent access to failed functions while providing access to all currently operational functions			
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Major
Reporter:	Sarah Suleri	**Assignee:**	Omar Morando
Labels:	NFR		
Requirement Type:	Non-Functional - Operational		
Rationale:	This refers to the survivability of the system.		
Fit Criterion:	The system prevents access to failed functions while providing access to all currently operational functions.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	High		
Source:	User Needs Analysis		

[CRD-142] Unless the system is non-operational, the system shall present a user with notification informing them that the system is unavailable.			
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Major
Reporter:	Sarah Suleri	**Assignee:**	Angel Soriano
Labels:	NFR		
Requirement Type:	Non-Functional - Performance		
Rationale:	This refers to the availability of the system.		
Fit Criterion:	Unless the system is non-operational, the system presents a user with notification informing them that the system is unavailable.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	High		
Source:	User Needs Analysis		

[CRD-143] Passwords shall never be viewable at the point of entry or at any other time.			
Status:	Quality Check passed		
Project:	CPSwarm - Requirements and Development		
Type:	Volere Requirement	**Priority:**	Medium
Reporter:	Sarah Suleri	**Assignee:**	Omar Morando
Labels:	NFR		
Requirement Type:	Non-Functional - Security		
Rationale:	This refers to the access security of the system.		
Fit Criterion:	Passwords are never viewable at the point of entry or at any other time.		
Customer Satisfaction:	High		
Customer Dissatisfaction:	Neutral		
Source:	User Needs Analysis		

4.2 Requirements Validation

Validation activities done by SLAB are divided throughout the project lifetime into two tasks: T2.4 Validation Framework Specification and T8.4 Use Cases Validation.

The deliverable D2.8 describing the Validation Framework (produced by T2.4) due M18 describes a methodology established by SLAB which was used to validate the requirements created for the CPSwarm Workbench. Section 4.3 of deliverable D2.6 specified the main characteristics and workflow of the Validation Framework designed for the CPSwarm project; we repeat the content in the paragraphs below for accessibility purposes.

Validation and Verification are procedures in quality management checking whether a product, service or system meets its predefined requirements and whether it fulfils its intended purpose. Since these two terms are often used together, sometimes interchangeably, it is worth taking some words to clarify what we mean by validation and verification in the CPSwarm project. Validation aims to answer the question "*Are we building the right system?*" whereas Verification helps us to answer to "*Are we building the system right?*". Validation is used to ensure that a product, service or system is designed to satisfy the needs of its customers, users and other stakeholders while verification ensures that the end product complies with its specification.

Our methodology uses different kind of metrics to validate and verify requirements – namely Key Performance Indicators (KPIs), Test Cases and Maturity Levels.

First, the requirements are translated into measurable metrics: either test cases which, when passed indicate that the requirement has been met, or into KPIs which set a target value in a way that supports the assumption that the requirement has been met. Templates for KPIs and formal/informal test cases are included in the deliverable.

When a requirement meets the KPI assigned to it or passes its test case, it indicates that the project is making progress – but to measure how much, these events are linked to specific maturity levels. Thus, different KPIs are required for different maturity levels. We defined five maturity levels to be used in the Validation Framework:

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

Building on the roadmap described in the project proposal, we set a number of milestones based on the due dates of relevant deliverables with target maturity levels for each component and the workbench as a whole.

The goal of our validation activities is to track and validate changes to the project requirements implementing an iterative approach. When requirements change and/or components mature, these changes are periodically registered and new metrics are tailored to validate them.

At the time of writing, an initial evaluation of the CPSwarm components has been performed using the methodology described above (and specified in detail in D2.8). The results of this evaluation have been published in deliverable D8.7.

The maturity levels of the CPSwarm components have been determined based on KPI values at the time of the evaluation. Overall, maturity levels for most components are ML1 or ML2, somewhat behind the expected overall ML2 maturity level. The evaluation has identified some clear areas of improvement for the less-mature components in the project moving forward.

As more advanced functionality is implemented into the CPSwarm components, we will be updating the requirements in preparation for the second round of evaluations. This second round of requirements will be specified in Deliverable D8.8 – Final Validation results (M36).

5 Lessons Learned

The following sections discuss lessons learned for various components of the CPSwarm workbench. We have used a standard template [18] in order to document lessons learned.

5.1 Modelling Library

Table 1: Lessons learned for Modelling Library

Category	Issue Name	Problem/Success	Impact	Recommendation
Interface	Model element differentiation	The graphical differentiation between some UML or SysML elements is often not clear enough for new modeller.	Users misunderstand examples and encounters difficulties during their learning phases.	Graphical differentiation between UML and SysML elements must be increased by using, e.g., different colours.
Wording	Confusion between Modelling Catalogue and Modelling Wizard	Both Modelling Catalogue and Modelling Wizard concepts have been introduced in CPSwarm. The Modelling Catalogue is composed of a set of model examples, the Modelling Wizard provides guidelines, based on the Model examples, to sketch CPS swarms.	Users often learn by try and test different models but the confusion between the Modelling Catalogue and the Modelling Wizard lost them.	Modelling Catalogue and Modelling Wizard must be more documented to clarify these concepts.

5.2 Modelling

Table 2: Lessons learned for Modelling

Category	Issue Name	Problem/Success	Impact	Recommendation
Modelling	Environment modelling is not user friendly	Modelling a complex 3D environment is not efficient using UML/SysML-like editors.	Modelling a physical environment under an UML/SysML-like editor is a waste of time and energy.	The concept of environment must be still available under the Modelling Tool, but as reference to external artefacts managed by dedicated third-party tools.

Category	Issue Name	Problem/Success	Impact	Recommendation
Modelling	Security/Safety aspect management	Several concepts of the models have impact on the security or safety aspects.	Time is spent during safety and security management in specified aspect already known at the modelling level	Identifying and exploiting the modelling impact on safety and security aspects would be a huge gain of time.
Component Integration	Integrating modelling simulation.	Simulation was originally planned to be performed for Optimization only.	Modeller cannot simulate their designs without optimizing them before.	A connection between the Modelling and the Simulation Tools should be implemented.

5.3 Optimization

Table 3: Lessons learned for Optimization

Category	Issue Name	Problem/Success	Impact	Recommendation
Interface	Optimization Tool Generality	Optimization Tool API should support multiple optimization tools.	XMPP interface must be useable by any optimization tool.	Explore and test integration with other optimization tools.
Interface	Optimization Tool Robustness	Optimization Tool API should be robust against failure.	Failure requires restarting the optimization session.	Extend API to support resuming optimization sessions.
Functionality	Problem Generality	The Optimization Tool must support multiple problem definitions.	The Optimization Tool supports extensive configuration options but remains agnostic to the specific of the actual problem definition.	Test the applicability of different configurations in real-world scenarios.
Functionality	Swarm Algorithm Optimization	The Optimization Tool must be able to optimize swarm algorithms.	The Optimization Tool can optimize swarm algorithms in an identical fashion to other algorithms.	Test optimization problems involving swarm algorithms.

Functionality	Human-in-the-loop Optimization	The Optimization Tool must be able to optimize human-in-the-loop parameters.	Human-in-the-loop parameters may be optimized by optimizing multiple related sub-problems.	Implement specific support for human-in-the-loop parameters. Test optimization problems involving human-in-the-loop parameters.
Component	Optimization Tool Scalability	The Optimization Tool must be able to scale to large problems.	As heavy computation is not conducted by the Optimization Tool itself, simulation nodes may be added to scale the system's performance.	Test scalability using complex problems.
Component	Optimization Tool Performance	The Optimization Tool must be able to efficiently optimize problems.	Better performance requires more sophisticated evolutionary algorithms.	Implement and evaluate sophisticated evolutionary algorithms such as CEA2D.

5.4 Simulation

Table 4: Lessons learned for Simulation

Category	Issue Name	Problem/Success	Impact	Recommendation
Component Integration	Integrating simulation environment	Several physical-close simulators (e.g. Gazebo and Stage) have been integrated to test the XMPP based distributed approach.	The code has been refactored to support this type of simulators.	Extend the simulation environment to other ROS based simulators, to make it as general purpose as possible.
Interface	Simulator API	With the refactored simulation API, the controller is passed from the OT to the simulator.	The number of messages exchanged is lower than before.	Check if this is possible with all the candidate types, not only neural networks.
Interface	Simulator API	With the refactored API, the discovery of the server is done	The discovery process doesn't affect the optimization time.	Implement a solution based on XMPP presences to keep the list of available servers

Category	Issue Name	Problem/Success	Impact	Recommendation
		only in a first phase.		updated also during the optimization.
Component	Compilation Time	Recompile the candidate passed to the simulator requires the recompilation of the ROS project.	The overall optimization time is very long (almost a week with one simulator).	A way needs to be found (incremental compilation, use of text file) to reduce the compilation times.
Component Integration	External Simulator integration	Integrating Gazebo and Stage as distributed simulation environment using ROS.	Stage and Gazebo simulator started using XMPP messages.	Implement standard interfaces to better control the simulation process.
Component	Simulation Environment Scalability	Scaling the simulation environment requires to use a different (virtual) machine for each simulation server.	Scaling the simulation environment requires the access of too many physical machines.	Use technologies like Docker and Cloud Services to easily scale the Simulation Environment.
Component	Communication Protocol	Implement the API using a communication protocol that supports all the communication patterns required.	The implementation of the API in the different components require only to integrate the client for one protocol.	Use protocols that support multiple communication paradigms (publish/subscribe, one-to-one, presences) like XMPP.

5.5 Code Generation

Table 5: Lessons learned for Code Generation

Category	Issue Name	Problem/Success	Impact	Recommendation
State Machine implementation library	System Responsiveness	Transition from a state to another triggered by external events has some latency.	The SMACH state machine is not enough responsive to manage safety or emergency situations.	Emergency or safety operations should not be managed at the State Machine level if there are strict time constraints.
Code generator input	SCXML standard extensibility	The SCXML format in not easily extensible.	Extend the SCXML format in order to fulfil some specific Code Generation requirement is not always straightforward.	The possibility to extend the SCXML parser library or to implement an ad-hoc one to parse CPSwarm's SCXML extensions could be considered.

5.6 Abstraction Layer

Table 6: Lessons learned for Abstraction Layer

Category	Issue Name	Problem/Success	Impact	Recommendation
Component Design	Abstraction Library API design	Developing the Abstraction Library starting from ROS turned out to be a good decision.	Levels of abstraction provided by ROS through message interfaces allowed a good software's reusability and flexibility for future extensions.	In order to support "not-compatible with ROS platforms" the possibility to guarantee the same features should be considered as a starting point.
Component	Abstraction Layer usability	Definition of an Abstraction Layer is not sufficient to support the automatic generation of model-designed swarm behaviour.	Functionalities provided by the Abstraction Layer cannot be used if not correctly modelled in the Modelling Tool.	Provide accurate models for all high-level functionalities in the Abstraction Library.

5.7 Deployment

Table 7: Lessons learned for Deployment

Category	Issue Name	Problem/Success	Impact	Recommendation

Code Compilation	Long compilation time	The compilation takes a very long time on CPS hardware.	The long compilation time extends the duration of deployment process.	Adopt cross-compilation strategies using more powerful hardware. Perform incremental compilation.
Code Compilation	Cross compilation complexity	The high complexity involved in cross compilation toolchain setup is costly for most developers.	Most developers tend to perform native compilations.	The Deployment Tool should provide a once-for-all native compilation utility.
Deployment Monitoring	Log management	Collecting all application logs wastes networking and processing resources.	Logs are essential information for debugging. Collecting all logs from remote devices has significant performance drawbacks.	The Deployment Tool should prioritize logs and offer a way to collect more verbose information on demand.
Deployment	Target selection	It is difficult to select CPSs among a large pool of heterogeneous devices.	It becomes more difficult to select devices as their number increases, making it error-prone, tedious, and often impossible.	The Deployment Tool should provide a high level grouping system to target devices based on their meta information or location.
Deployment Monitoring	Status monitoring	Monitoring the status of many devices is not feasible by looking at logs from individual devices.	Erroneous application runtime behaviour is difficult to catch.	The Deployment Tool shall provide a way of clustering important information depending on their similarity.

5.8 Monitoring

Table 8: Lessons learned for Monitoring Tool

Category	Issue Name	Problem/Success	Impact	Recommendation
Integrateability of process	Adapt for industrial grade Requirement Management Tools	We use Integrity and DOORS. It was complex to integrate the requirements into the system.	This meant, that all requirements needed manual integration.	We will not make use of this method after the project. It was interesting to explore such option, but we do not see a use in

				industrial development.
Tracking requirements/Coverage	Complexity in tracking	When generating/developing the Monitoring Tool we found it difficult to track coverage of the requirements other than by manual access.	In our tool approach this is covered with powerful Tool support.	In our requirements process it is also questionable if it is feasible to take several rounds of requirements generation. In general, we need to make a "Change Request" to the customer in case of modifying agreed requirements. We find it difficult to apply this approach to our internal development process.

5.9 Continuous Integration

Table 9: Lessons learned for Continuous Integration

Category	Issue Name	Problem/Success	Impact	Recommendation
Build Artefacts	Artefact delivery	Most CPSwarm components are released after successful build and test as soon as the developer makes changes to the code.	All interested stakeholders have access to the latest tested artefacts.	All new components should follow the same CI flow to ensure up-to-date and secure artefact delivery.
Build Status	Build and test results	Developers receive e-mail notifications whenever their changes result in build errors and failed tests.	Developers get information during every development iteration and can possible fix issues.	Developers should subscribe to build and test results and promptly react to possible issues.

6 Design Pattern Library

Throughout the requirements elicitation process in the first 16 months of the projects, it turned out that there are more than "just" functional and non-functional requirements and "user needs" that need to be taken into account when designing a cyber-physical system in a specific domain. There is a need of a certain knowledge foundation on which the team can start designing applications. This knowledge foundation includes knowledge, e.g., on:

- Rules and regulations

- Safety rules and law
- Proven design guidelines
- Standards
- De-facto standards
- Commonly known agreements and workarounds
- UI Design

Knowledge from these areas is usually not directly formulated as a requirement but something that needs to be taken in to account when trying to fulfil other requirements. Thus, this knowledge is implicit and only accessible via domain experts. The approach of the CPSwarm project is to gather technologies and knowledge in libraries as described in the preceding sections. For the kind of knowledge described here, the common knowledge ground cannot be kept, explained and conveyed in a technical way.
Therefore, the concept of *Application Design Patterns* is applied in this concept and interconnected pieces of knowledge will be formulated within an evolving design pattern library. The concept of the design pattern is established since long in architecture [19], software systems [20], organizational contexts [21] as well as user-interface [22], website [23] or application design [24].

A design pattern consists of specific parts describing the *context*, in which a certain *problem* occurs and how it can be *solved*, including the *consequences* of the solution. Patterns are usually organized in clusters of design pattern libraries that are presented by domain experts after a long time of engineering. CPSwarm follows the approach of described in [25] who formulates patterns during system engineering and design time. Pattern mature and reach a reliable state during the project work and are supported by the project community instead of one pattern expert. The whole approach is web-based such that the CPSwarm project can present and make accessible the patterns gathered over the project duration and beyond. This helps to present sustainable domain knowledge supporting future work.
In upcoming deliverables from work packages 4 that handle the "Human-in-the-Loop" concept, the implemented approach will be described in more detail together with the online version of the CPSwarm Design Pattern Library.

6.1 Design Pattern Structure

The used pattern structure is inspired by the approaches described by Alexander and others [24, 25] and provides a flexible set of fields that are filled over time during the maturation process. The more mature a pattern becomes, the more fields need to be filled in order to improve its completeness. Based on the fields, i.e., the pattern's formulation quality and validity are determined and influenced by updated formulations. Each pattern should take into account the following principles:

- The patterns are formulated in natural language.
- The patterns must be easily understandable by non-experts.
- The patterns must be relevant for the project's domain.

The recommended reading path of the pattern starts with the *name* that should already give an idea about the pattern's topic, followed by a *problem summary* and *suggested solution*. The contents should be easy and fast to read. Based on these fields, the reader is able to make a quick decision whether the pattern is suited to the current situation in the project work or information finding process. In this case the fields about the context, detailed problem description, solution summary and solution description are the next parts of interest. For the pattern as a whole, the fields are arranged in a different order as shown below since they are arranged in an argumentative way for reading the entire pattern.

The derived pattern structure that is used within the evolving library approach is as follows:

- The pattern's *Name* should be short and instructive, reflecting the solution to the problem being addressed. As in the traditional approaches, the name should be easy to remember and encapsulate the pattern's central statements such that it can serve the project's vocabulary.

- The *Hierarchy Level* is treated as a category. However, instead of simply clustering patterns, they shape the structure of the pattern library and therewith the relations between patterns, i.e. more abstract patterns are formulated in the upper hierarchies, more concrete patterns in the lower ones (cf. Section 5.3). The hierarchy level can be suggested by the pattern author but altered by community suggestions during the process. This way, the pattern can be moved to another hierarchy level depending on its abstractness of formulation.

- The Pattern *Maturity State* is determined by the rules of the pattern maturation process and depends on the pattern's formulation quality, with regard to readability, understandability and appropriateness, as well as its validity as explained in [25].

- Authors can initially mark a contribution as anti-pattern via an *Anti-Pattern Indicator*. In the interest of the amount of project knowledge that needs to be managed, only non-trivial flaws should be documented. Still, the anti-pattern indicator is continuously adjusted during the process that allows the contrary development of a pattern based on supporting and refuting evidence.

- The *Context* section relates patterns to each other. In the context section of a pattern, the patterns that point towards the current one can be described such that only a brief summary of the context needs to be given. Further information can be found in the preceding patterns the context refers to. Often, a preceding pattern is extended by the current one that now tackles more specific aspects of a more general solution. A problem is examined in more detail and more specific solutions are described. With the help of the context, the reader is able to decide whether he/she possesses enough knowledge to understand the current situation the pattern describes or if he/she needs to read more preceding patterns in order to fully understand the current pattern's intention.

- The *Problem Summary* field briefly outlines the central problem the pattern tackles in order to allow the reader to quickly decide whether the pattern matches to the problem situation he/she is currently dealing with.

- The field on *Problem Details and Forces* further describes the problem context and discusses reasons that lead to the problem. Reasons can originate from external influences such as legal or technical restrictions that are further elaborated. These "forces" influence the proposed solution. When applying the pattern, the reasons for the forces as well as their impact on the solution need to be understood by the reader. Since the proposed solution may be well-suited for specific aspects but implies disadvantages on other aspects, different patterns on the same level may propose different alternative solutions. In the scope of exploring the project and development knowledge, the understanding of forces within the detailed problem discussion may lead to alternative solutions that have not yet been analysed.

- The *Solution Summary* provides the central statement of how to solve the problem in the given context. The concise formulation of the solution serves the reader's decision-making process whether the pattern is suited to his/her current situation but also helps to better remember the advice given by the pattern.

- The section on *Solution Details and Consequences* elaborates on the factors and reasons that lead to the solution. Explanations and considerations on the pros and cons with regard to the forces are given. This part should point out benefits but also discuss disadvantages that occur when following the advice. Here, the consultative character of a pattern as solution approach but also as knowledge source

is emphasized. The solution is not to be accepted as a statement out of question but as consideration and elaboration of known possibilities and consequences.

- *Illustrations* enrich the explanation of the solution and further support the usage of the pattern as vocabulary. The pattern's name and central solution statement can be connected to the illustration therefore making it easier to remember and recognize the pattern again during browsing the pattern language. In the presented approach, multiple illustrations may be submitted as drawings, diagrams, pictures or videos. The pattern author is encouraged to primarily provide a key illustration in an image-based format to support the described mental connections.

- The mentioning of a pattern's *Pattern Origin* encourages the reuse of already existing and validated results from other projects or repositories that are relevant for the current project knowledge. The origin field differentiates between newly derived and introduced patterns. It may be necessary to adapt the latter kind of patterns since they may be formulated in a different, more specific or more abstract context but the essence of the pattern is relevant for the project and can be extracted. This circumstance has to be regarded when formulating the pattern. The insertion may need to adapt naming conventions and formulations and transform them into the pattern language's terminology. The approach distinguishes between three different categories:

  - Derived from project: The pattern was derived directly from the work within the project. Continuous formulation and validation need to ensure the pattern's validity.
  - Adapted to project: The pattern originates from external sources but has been adapted to the project's context. Still, the need for validation is given. Preliminary project-external work was already put into the pattern's formulation and is used as evidence supporting the pattern as described below.
  - Project-external: The pattern exists in other related pattern collections. The pattern can directly be used in the current project scope since the project domain and pattern origin are closely related.

6.2 Design Pattern Library Structure

As the current state of the library is still initial a first idea of hierarchy levels is introduced that needs to be further elaborated with pattern experts. Figure 7 shows a screenshot of the current pattern library structure. The web-based version of the pattern library is available online at https://patterns.fit.fraunhofer.de/cpswarm.
The current state of the design pattern library foresees the following pattern categories that are put in the following, preliminary hierarchy:

- Laws and Ethics
- Safety
- Technology and System Design
- Human-Swarm Interaction Design
- Data Management
- Privacy and Security
- Application Domain #1: Search and Rescue
- Application Domain #2: Warehouse Logistics
- Application Domain #3: Automotive

As per process definition, the structure is not fixed yet and will be adapted over time according to the findings during the pattern formulation. The categories represent current findings where knowledge on application design was already formulated by application partners during requirements and demonstration planning sessions. The current discussion is about the reintegration of existing patterns from the existing project

BRIDGE[1] from the emergency response domain and the formulation of current safety patterns that were partially revealed during the conception of the two review demos. About categories, the current discussion involves the splitting of laws and ethics since these are not necessarily always in line with each other.

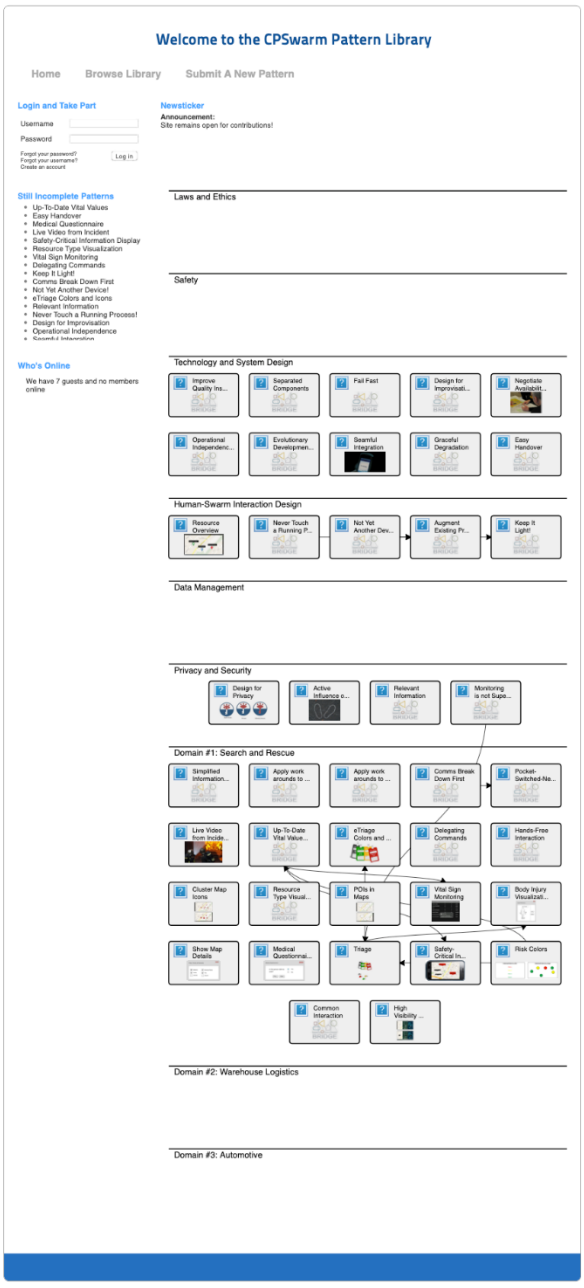

Figure 7: Screenshot of the current pattern library structure

6.3 Plans to proceed

From the necessity of modelling and documenting human-swarm interaction, the so-called "human-in-the-loop" concept (cf. D4.2), the idea arouse to document "soft" findings and "things that need to be known" when designing a swarm application as design patterns. With the introduced pattern library structure, several aspects are covered and trigger the designer to consider them at design time. Since the concept and the gathering of

1 http://www.bridgeproject.eu

contents really started in the third project year, the need for a pattern library and the current structured can be expressed in this deliverable.
First pattern candidates are already collected. However, the detailed formulation of patterns needs to follow in dedicated focus groups and together with application experts.
The current structure is not final and will develop over time. In addition, the project does not aim at providing a complete pattern library at the end but a knowledge structure with content that can be edited and extended beyond the project and within the community. This way, the library is, in this deliverable, formulated as a requirement to cover knowledge and needs from a pattern perspective.

7 Conclusion

The initial phase of the CPSwarm project focused on the specification of use cases, the definition of its stakeholders, as well as the description of the communication flow between them. Beyond, it focuses on the workflow of the workbench and on the vision of the deployment of CPSwarm workbench in practice.

One of the objectives of the present deliverable was to establish a common ground on which the remaining WP2 tasks, and later the remaining technical WPs (WP3 to WP7), will build their foundations towards the demonstration (WP8). The work in WP2 follows a scenario-driven approach, starting with the formulation of vision towards which the project will develop. The visions serve as basis for identifying involved stakeholders, available knowledge, used technologies as well as their interplay and data flow. From the basic set of use cases, further specifications of workflows performed with the help of the CPSwarm workbench will evolve.

The analysis presented in this deliverable started with the description of the Volere requirements scheme that is used throughout this deliverable to specify requirements. The process of requirement engineering in D2.7 proceeds by taking one step forward from the user needs and requirements identified in D2.3 and D2.6. In D2.3 we extracted user roles who interact with workbench and alter the communication flow between them by dividing it into four phases; Design, Implementation, Deployment and Operation phase. From the perspective of each user role, we defined user needs in the form of user stories. The next step was to translate these user needs into abstract workbench components and to define flow of information between them. The responsibilities of these workbench components and the data flow between them were defined in the form of technical requirements. D2.7 contains the reiteration of these user needs and requirements. In addition to the requirements, D2.7 also contains an updated set of lessons learned during the design and implementation of various components of the workbench.

The requirements specified in this version of the deliverable are to be seen as the final iteration in the scope of remaining WP2 tasks and WPs 3 to 8. By defining a common set of user needs and Volere requirements, this deliverable D2.7 laid the foundation that will be used in further implementation in the technical WPs.

Conclusively, this deliverable documented the iterative process of ideation and concept development in order to identify various user needs. In addition to the user needs, the identification and specification of Volere requirements related to the workbench components and their lessons learned are significant results from this task that will be used as input to subsequent activities of the project.

Appendix A

WP2 will address the demand for safety and security requirements to pay attention to in a living document as part of the CPSwarm Libraries in the sense of background knowledge and things application engineers and domain experts need to know when designing an application via the CPSwarm workbench. This way, the library and knowledge gathering can stay alive and be extended.

All partners, especially from the application domains have a strong security background and were asked to provide references to existing

- regulations,
- norms,
- laws,
- rules,
- guidelines

for swarm application design.

Aeronautics sector (predominantly DigiSky)

As regards UAVs and drones, a basic reference Regulations does not exist. The European Commission, in accordance with the European Aviation Safety Agency, is working a on mid-term project to develop a standard regulation to increase safety operations with unmanned vehicles.

- A first step is the EU Reg. 2018/1139.
- Some basic information and guidelines for drones operations are available on the EASA website[2]. Related to the link, there are lots of general safety/operational publications and articles with interesting points of view and opinions on the problem.
- On the Member States of European Union, every national authority has the possibility to follow the national laws approved by governments. For example, in Italy, the aviation authority has developed different regulations like:
    - Remotely Piloted Aerial Vehicles Regulation
    - Air Rules
- It is very important to verify in every country the applicable laws and regulations regarding unmanned vehicles. Some guidance instructions and rules are available also in EN9100:2018 that define quality assurance requirements for aerospace products (control on design and development, production.
- In a perspective of a global Safety analysis of a typical CPSwarm scenario, useful material could be found in the International Civil Aviation Organization (ICAO) "Safety Management Manual" Doc.9859, with the complete overview and process description of a Risk Management Analysis.
- Other norms not directly related: ISO 13849-1:2015 Safety of machinery, ISO 13850:2015 Standard specifies functional requirements and design principles for the emergency stop function on machinery and IEC 60204-1:2016 Safety of machinery - Electrical equipment of machines.

2 https://www.easa.europa.eu/easa-and-you/civil-drones-rpas

Unmanned Vehicles

- ISO/TS 15066:2016 applies to industrial robot systems as described in ISO 10218?1 and ISO 10218?2. It does not apply to non-industrial robots, although the safety principles presented can be useful to other areas of robotics. We are using the general principles for the risk assessment.
- Novelties on service robotics are managed by Technical ISO Committee TC299: https://www.iso.org/committee/5915511/x/catalogue/p/0/u/1/w/0/d/0
    - New developments are carried on ISO/CD 22166-1.2, "Robotics- Modularity for service robots - Part 1: General Requirements" [4], which is not available yet.
    - We are still using an old norm that applies for unmanned vehicles:
    - EN 1525:1997: Safety of industrial trucks - Driverless trucks and their systems

Other norms not directly related

- ISO 13849-1:2015 Safety of machinery. Provides safety requirements and guidance on the principles for the design and integration of safety-related parts of control systems (SRP/CS), including the design of software.
- ISO 13850:2015 Standard specifies functional requirements and design principles for the emergency stop function on machinery, independent of the type of energy used.
- And in general, IEC 60204-1:2016 Safety of machinery - Electrical equipment of machines - Part 1: General requirements

Norms with regard to cyber-security

- Information technology -- Security techniques -- Information security management systems -- Requirements: ISO/IEC 27001
- Common Criteria for Information Technology Security Evaluation: ISO/IEC 15408
- OWASP Software Assurance Maturity Model - prescriptive framework for securing the development process: https://github.com/OWASP/samm/raw/master/Supporting%20Resources/v1.5/Final/SAMM_Core_V1-5_FINAL.pdf
- Build Security In Maturity Model - descriptive framework for identifying security activities used in the real world: https://www.bsimm.com/content/dam/bsimm/reports/bsimm9.pdf
- Build Security In - best practices for secure software design and implementation: https://www.us-cert.gov/bsi
- Software Assurance Forum for Excellence in Code (SAFECode) - fundamental rules for secure software development: https://safecode.org/wp-content/uploads/2018/03/SAFECode_Fundamental_Practices_for_Secure_Software_Development_March_2018.pdf
- Microsoft Security Development Lifecycle (SDL) - prescriptive framework for securing the development process: https://www.microsoft.com/en-us/securityengineering/sdl/

- International Council on Systems Engineering: Systems Security Engineering working group (INCOSE SSE): https://www.incose.org/incose-member-resources/working-groups/analytic/systems-security-engineering
- Team Software Process for Secure Systems Development (SEI-CERT, 2002): https://apps.dtic.mil/dtic/tr/fulltext/u2/a634138.pdf
- SEI CERT C Secure Coding Standard: https://wiki.sei.cmu.edu/confluence/display/c
- SEI CERT C++ Coding Standard: https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682

Automotive sector (predominantly TTTECH)

From the experts in this domain, no further explicit guidelines were mentioned adding the one already stated.

Acronyms

Acronym	Explanation
CPS	Cyber Physical System
CI	Continuous Integration
GUI	Graphical User Interface
HW	Hardware

List of figures

List of tables

References

[1] Chin, G., M.B. Rosson, and J.M. Carroll. Participatory analysis: shared development of requirements from scenarios. In SIGCHI conference on Human factors in computing systems. 1997.

[2] Easterbrook, S., Negotiation and the Role of the Requirements Specification. Appears in P. Quintas (ed.) Social Dimensions of Systems Engineering: People, processes, policies and software development, 1993: p. 144-164.

[3] Glinz, M., Improving the Quality of Requirements with Scenarios, in Proceeding of the Second World Conference on Requirements Engineering. 2000: Schaumburg. p. 254-271.

[4] Holbrook H., I., A scenario-based methodology for conducting requirements elicitation SIGSOFT Softw. Eng. Notes 1990 15 (1): p. 95-104

[5] ISO, ISO 9241-210:2010 Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems. International Organization for Standardization, 2010.

[6] Jarke, M. and K. Pohl, Requirements engineering in 2001: (virtually) managing a changing reality. IEEE Software Engineering, 1994. 9(6).

[7] Penna, G.D., et al., An XML Definition Language to Support Scenario-Based Requirements Engineering. International Journal of Software Engineering and Knowledge Engineering, 2003. 13(3): p. 237-256.

[8] Ramesh, B. and M. Jarke, Toward Reference Models for Requirements Traceability. IEEE Trans. Softw. Eng., 2001. 27(1): p. 58-93.

[9] Robertson, J. and Robertson, S.; Mastering the Requirements Process. 1999: Addison-Wesley.

[10] Robertson, J. and Robertson, S.; Requirements-Led Project Management. 2004: Addison-Wesley.

[11] Robertson, J. and Robertson, S.; Volere Requirements Specification Template. 2010.

[12] Stufflebeam, W., A.I. Antón, and T.A. Alspaugh. SMaRT – Scenario Management and Requirements Tool. In 11th IEEE International Requirements Engineering Conference. 2003.

[13] Sutcliffe, A.G., W.-C. Chang, and R. Neville, Evolutionary Requirements Analysis, in 11th IEEE Requirements Engineering Conference. 2003.

[14] Sutcliffe, A.G. Scenario-based Requirements Engineering. In 11th IEEE International Requirements Engineering Conference (RE'03). 2003.

[15] Zimmerman, J., Stolterman, E., and Forlizzi, J. An Analysis and Critique of Research through Design: Towards a Formalization of a Research Approach. In Proceedings of the 8th ACM Conference on Designing Interactive Systems, 2010: ACM Press, p. 310–319.

[16] Design, Venture, The Inter-disciplinarian, and About Me. "Your Best Agile User Story". Alex Cowan. N.p., 2017. Web. 18 June 2017.

[17] Gomaa, H. and D.B.H. Scott. Prototyping as a tool in the specification of user requirements. In Proceedings of the 5th international conference on Software engineering. 1981. San Diego, California, United States.

[18] M. Piscopo, "Lessons Learned," Lessons Learned Template. [Online]. Available: http://www.projectmanagementdocs.com/project-closing-templates/lessons-learned.html#axzz55ll5w9dT. [Accessed: 04-Jan-2018]

[19] Alexander, C;. A Pattern Language: Towns, Buildings, Construction. Oxford University Press, New York, NY, USA, 1977.

[20] Gamma, E. and Helm, R. and Johnson, R. and Vlissides, J.; Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley Longman, Amsterdam, 1994.

[21] Manns, M., and Rising L.; Fearless Change: Patterns for Introducing New Ideas: Introducing Patterns into Organizations. Addison-Wesley, Boston, MA, USA, 2005.

[22] Tidwell J.; Designing Interfaces. O'Reilly Media, Sebastopol, CA, USA, 2nd edition, 2011.

[23] Van Duyne, D., and Landay J., and Hong J.; The Design of Sites: Patterns for Creating Winning Websites. Prentice Hall, 2nd edition, 2007.

[24] Borchers, J.; A Pattern Approach to Interaction Design. John Wiley & Sons, West Sussex, England, 1st edition, 2001.
[25] Reiners, R.; An Evolving Pattern Library for Collaborative Project Documentation. Shaker, 2014.