

# Searchable multi-dimensional Data Lakes supporting Cognitive Film Production & Distribution for the Promotion of the European Cultural Heritage

Grant Agreement No 101095303

**DELIVERABLE D5.1:**

**SCENE testing & system integration.R1**

Work Package: 5

**LEAD BENEFICIARY:**

**HYPERTECH**

Delivery Date: 28.02.2025



# Document Sheet

<b>Project acronym</b>	<b>SCENE</b>
<b>Project full title</b>	Searchable multi-dimensional Data Lakes supporting Cognitive Film Production & Distribution for the Promotion of the European Cultural Heritage
<b>Programme</b>	Horizon Europe
<b>Topic</b>	HORIZON-CL2-2022-HERITAGE-01-06
<b>Type of Action</b>	HORIZON-Research and Innovation Actions
<b>Grant Agreement</b>	101095303
<b>Start day</b>	1 February 2023
<b>Duration</b>	36 months

## LEGAL NOTICE

This project has received funding from the European Union Horizon Research and Innovation programme under grant agreement No 101095303. Neither the European Commission nor any person acting on behalf of the Commission is responsible for the use, which might be made, of the following information. The views expressed in this report are those of the authors and do not necessarily reflect those of the European Commission.

## ©SCENE Consortium, 2025

Reproduction is authorised provided the source is acknowledged.

## Document Information

<b>Deliverable number</b>	D5.1
<b>Deliverable name</b>	SCENE testing & system integration.R1
<b>Lead beneficiary</b>	HYP
<b>WP</b>	5
<b>Related task(s)</b>	T5.1 Platform Integration
<b>Type</b>	R
<b>Reviewers (Organisation)</b>	WR, LINKS
<b>Delivery date</b>	28.02.2025
<b>Main author(s)</b>	HYP
<b>Contributor(s)</b>	CERTH, DTT, LINKS, MOG, FOKUS, UPV, AUTH, CETMA

## Dissemination level

<b>PU</b>	Public	
<b>SEN</b>	Sensitive, limited under the conditions of the Grant Agreement	<b>X</b>
<b>Classified R-UE/EU-R</b>	EU RESTRICTED under the Commission Decision No2015/444	
<b>Classified C-UE/EU-C</b>	EU CONFIDENTIAL under the Commission Decision No2015/444	
<b>Classified S-UE/EU-S</b>	EU SECRET under the Commission Decision No2015/444	

## Document history

Version	Date	Changes	Reviewer/Contributor
v0.1	10/01/2025	Preparation of table of contents & introduction	HYP
V0.2	18/01/2025	Preparation of the first draft of the deliverable	CERTH, HYP
V0.3	26/02/2025	Feedback from reviewers	LINKS, WR
V0.4	28/02/2025	Preparation of the second version based on reviewers' comments	HYP
final	05/03/2025	Final version ready for submission	HYP, CERTH



# Table of Contents

Executive Summary.....	8
<b>1 Introduction .....</b>	<b>9</b>
1.1 Scope of this deliverable.....	9
1.2 Relation with other deliverables .....	9
1.3 Structure of the deliverable.....	10
<b>2 Integration Methodology .....</b>	<b>10</b>
2.1 Integration Plan.....	10
2.2 Integration Timeline.....	12
2.2.1 Ramp-up phase (standalone tools).....	12
2.2.2 Core Integration phase .....	13
2.2.3 Pilot phase.....	13
<b>3 SCENE Platform Integration.....</b>	<b>14</b>
3.1 Component interconnections .....	15
3.1.1 Data Lake .....	16
3.1.2 Ontology formulation .....	16
3.1.3 Conversational agent.....	16
3.1.4 Media Asset Manager.....	17
3.1.5 3D Reconstruction.....	17
3.1.6 Blockchain technologies.....	17
3.1.7 Location Scouting tool.....	17
3.1.8 AI-based Audience Preferences Scouting tool.....	17
3.1.9 Audience Building Tool.....	18
3.1.10 Lighting Simulation module .....	18
3.1.11 Audio Simulation module.....	18
3.1.12 Post-production Effects & Quality metrics.....	18
3.1.13 UWB-based Tracking System .....	18
3.1.14 Distribution Engine.....	18
3.1.15 Recommendation System .....	19
3.2 SCENE Platform interface.....	19
<b>4 Testing Process Management.....</b>	<b>20</b>
4.1 Software Quality Assurance .....	20
4.1.1 Software Quality Models .....	20
4.1.2 The ISO/IEC 25010 Model .....	22
4.2 Testing Methodology .....	22



4.2.1	<i>SCENE’s Agile Testing Levels</i> .....	23
4.2.2	<i>SCENE’s Integration Testing</i> .....	24
4.2.3	<i>Integration testing approach</i> .....	24
4.2.4	<i>Manual Testing</i> .....	25
5	<b>Integration Process Management Tools</b> .....	26
5.1	<i>Software Configuration Management</i> .....	26
5.1.1	<i>Code Management with Git</i> .....	27
5.2	<i>Code Analysis &amp; Quality Measurement</i> .....	28
5.3	<i>Integration Strategies</i> .....	29
5.4	<i>Integration Process of the SCENE Platform</i> .....	30
5.4.1	<i>Code Synchronization</i> .....	30
5.4.2	<i>Front-end &amp; Back-end Development</i> .....	30
5.4.3	<i>Code Publishing</i> .....	30
5.4.4	<i>Work-flow Adaptation</i> .....	30
6	<b>Conclusion &amp; future work</b> .....	31
	<b>References</b> .....	32



## List of Figures

Figure 1: Relations of D5.1 with other deliverables. ....	9
Figure 2: Integration Steps. ....	10
Figure 3: Integration methodology. ....	11
Figure 4: Integration timeline. ....	12
Figure 5: Ramp-up timeline. ....	12
Figure 6: Core phase timeline. ....	13
Figure 7: Pilot phase timeline. ....	14
Figure 8: SCENE conceptual architecture ....	15
Figure 9: SCENE platform mock-up. ....	19
Figure 10: Several perspectives on software quality. ....	20
Figure 11: Quality in use characteristics. ....	22
Figure 12: Agile Testing Levels. ....	23
Figure 13: Software quality dimensions supported by SonarQube ....	28
Figure 14: Integration Strategies. ....	29
Figure 15: SCENE’s Integration Process. ....	30

## List of Tables

Table 1: Ramp-up activities. ....	13
Table 2: Core integration activities. ....	13
Table 3: Pilot phase activities. ....	14
Table 4: SCENE component interconnections. ....	16
Table 5: Software Quality Models. ....	21
Table 6: Software Integration Test Plan. ....	24





## Abbreviations

Abbreviations	Full name
AI	Artificial Intelligence
D	Deliverable
T	Task
WP	Work Package
M	Month
UI	User Interface
SSO	Single Sign-on
SAML	Security Assertion Markup Language
OAuth	Open Authorization
TRL	Technology Readiness Level
API	Application Program Interface
LLM	Large Language Model
NFT	Non-Fungible Token
SCI	Software Configuration Item
SCM	Software Configuration Management
ISO	International Standardization Organization
KPI	Key Performance Indicator
DAT	Developers Acceptance Testing
PAT	Pilots Acceptance Testing



## Executive Summary

This deliverable, “D5.1 SCENE Testing & System Integration.R1”, presents a comprehensive report on the undergoing and planned integration and testing activities for the implementation of the SCENE platform prototype. The document outlines a structured, multi-phase integration methodology designed to ensure that all disparate tools and components work together seamlessly. Key integration phases include:

- Ramp-Up Phase: Validation of individual, standalone tools and their interfaces.
- Core Integration Phase: Consolidation of component interconnections across the platform, ensuring robust data flows and unified user experience.
- Pilot Phase: Execution of end-to-end system tests in simulated real-world environments to gather user feedback and drive final refinements.

At the heart of the platform is a robust, layered architecture composed by the Data, Application, and Presentation layers. The Data Lake serves as a central repository for diverse media and metadata, while the Application layer houses the core processing engines, from Location Scouting and Media Asset Management to 3D reconstruction, Blockchain technologies, and AI-driven tools. A unified front-end interface enables users to access all functionalities through a single sign-on, streamlining the overall user experience.

To guarantee high-quality performance, the integration strategy is supported by agile testing methodologies that encompass unit, integration, and system testing levels. Continuous integration pipelines, coupled with automated testing and code quality assessment tools are employed to maintain consistency and rapidly address issues.

Overall, this deliverable validates that the SCENE platform meets its objectives of being robust, scalable and secure, providing a solid foundation for cognitive film production and distribution in a highly collaborative, distributed environment.



# 1 Introduction

## 1.1 Scope of this deliverable

This document (D5.1) provides a report on the integration efforts being conducted for the common platform, called “SCENE platform”, and the validation of its seamless and flawless functionality. The platform is an integrated product with various interoperable components that support different phases of the documentary/film creation process: pre-production, production, post-production and distribution. Users can access all tools through a single login via the platform’s front-end interface. Each tool has a back-end engine for core functionalities and a user-friendly front-end interface. The architecture includes distributed tools, server-based tools, and a central shared Data Lake accessible by all platform tools. Additionally, the deliverable outlines the overall architecture of the SCENE platform, including its high-level and skeleton architecture.

The present deliverable outlines the tools, methodologies, and models used for testing during the SCENE platform integration. It emphasizes the importance of software quality assurance for assessing internal, external, and user-perceived quality. More precisely, this deliverable describes the general methodology of software integration and implementation. The distinct steps that are part of this methodology are being presented and analyzed in detail. The document also details the iterative and incremental development approach used in SCENE. Additionally, an agile testing methodology is adopted. Lastly, the integration process management tools for the SCENE platform prototype are being described, detailing software configuration management through tools for version control. A hybrid integration approach accommodates various components on different servers, ensuring seamless interaction through RESTful services.

## 1.2 Relation with other deliverables

This section presents the relationship between the D5.1 deliverable with other deliverables of the SCENE project. The deliverable D5.1 is centrally positioned (Figure 1) and is influenced by several preceding deliverables.

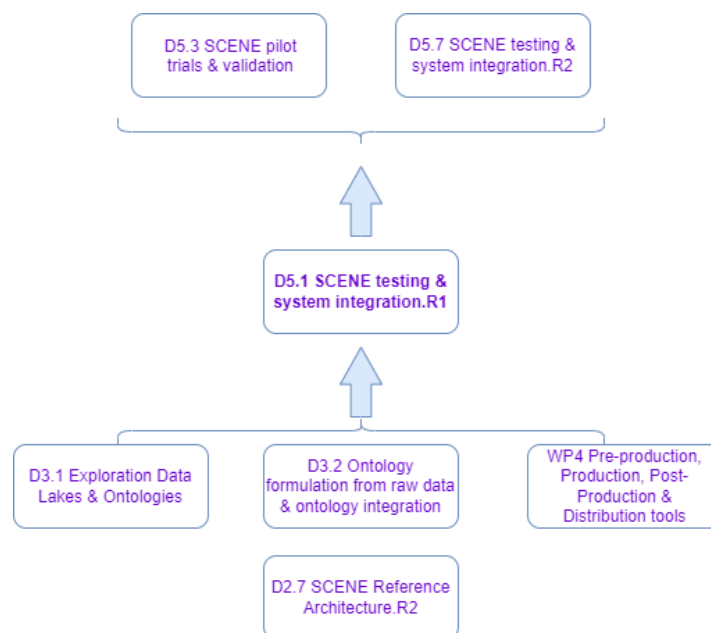


Figure 1: Relations of D5.1 with other deliverables.

Initially, the deliverables “D2.6 - SCENE Reference Architecture.R1” and “D2.7 - SCENE Reference Architecture.R2” regarding the system’s architecture are used as a blueprint about the implementation of this document. Additionally, it receives input from deliverable “D3.1 - Exploration Data Lakes & Ontologies” and deliverable “D3.2 - Ontology formulation from raw data & ontology integration” which is about the technologies of the ontology formulation and the ontology enrichment.

D5.1 will provide input to deliverable “D5.3 - SCENE pilot trials & validation” that is about the pilot trials of the SCENE project while the deliverable “D5.7 - SCENE testing & system integration.R2” which reports on the second round of the testing and SCENE systems integration as well as the finalization of the platform. All in all, this document acts as a bridge between the individual development of each tool, the system architecture and the project pilots.

### 1.3 Structure of the deliverable

This deliverable is structured and organized as follows. Section 1 provides the scope of this deliverable in detail, its relation with other deliverables and a brief description of the deliverable’s structure. Section 2 presents and describes the methodology of the integration and the software implementation. It also presents the integration timeline of the SCENE platform and details the distinct phases of this process. Section 3 details the holistic integration of the SCENE platform and analyses the interconnections of each component. Section 4 describes in detail the testing process management, containing both the quality assurance and the testing methodology. Section 5 describes the tools and strategies used for the integration process of the SCENE platform. Lastly, Section 6 acts as the conclusion of the deliverable and showcases possible future work.

## 2 Integration Methodology

### 2.1 Integration Plan

In software engineering, the development process is broken down into distinct tasks to streamline project management and maintain the quality of the end product. A similar structured approach can be applied to defining integration methodologies. Figure 2 illustrates the distinct tasks involved in standard integration planning and showcases the tasks that are iterated.

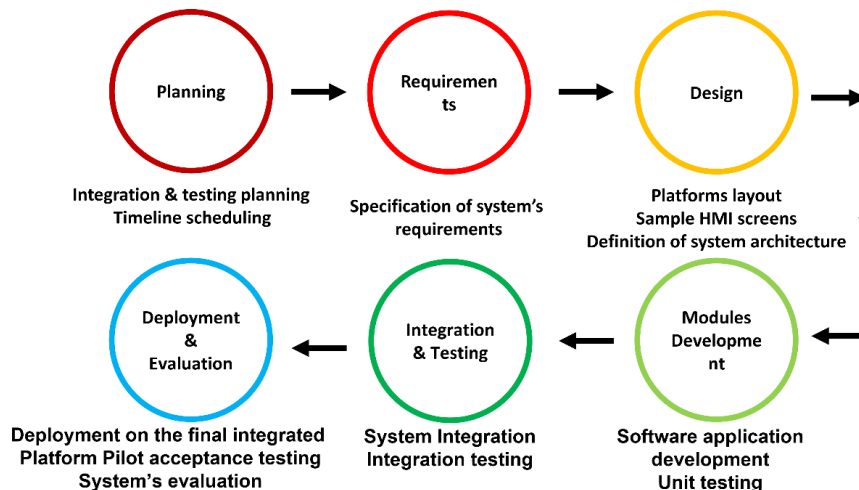


Figure 2: Integration Steps.

A short description of the afore-depicted tasks follows below:

- **Planning:** Initially, there is a need to define the integration planning, as well as the testing planning. Therefore, the timelines are drawn and defined and the scheduling for the whole of the activities and

efforts is defined until the end of the project, as agreed by the consortium partners. It is important to note at this point that the efforts for producing the present deliverable are part of the activities of the “Planning” step.

- **Requirements:** In this task, the system’s requirements are identified and defined. This process happens through effort from the partners that are responsible for the development of the individual tools and the final integrated platform. The requirements definition is part of the deliverables D2.2 and D2.3.
- **Design:** This step is responsible for the design and definition of the platform’s layout. These designs act as the structured base of the final integrated platform. This step contains the architecture of the Final Integrated Platform, which is further described in detail in section 3.2.
- **Component Development:** The component development task refers to the development and the finalization of each discrete component or tool. WP3 and WP4 are related to the effort of this task.
- **Integration & Testing:** This task is responsible for the combination of each discrete tool. These distinct tools are combined within the same platform establishing intra tool communications and facilitating certain workflows and use cases. As detailed below, this task also includes a parallel testing phase that is continuous and iterative. It will cover not only single, discrete tools but also the holistic integration tests, including functional and technical issues. Therefore, the functional and technical aspects will be tested at both the module and system levels. After each iteration, refinements will be made, based on test results and performance evaluations. These two sub-steps are described in later sections.
- **Deployment & Evaluation:** At this step the final integrated platform is being deployed under a pilot demonstration. The pilot demo acts as simulated environment where important feedback produced by end users can be acquired. The result of this is used for the application of final corrective actions. Last but not least, this step additionally includes the end-user evaluation process of the whole, final integrated platform, the analysis of which will also lead to corrective technical actions if necessary.

In order to achieve the integration of all modules an iterative strategy is followed as shown in Figure 3.

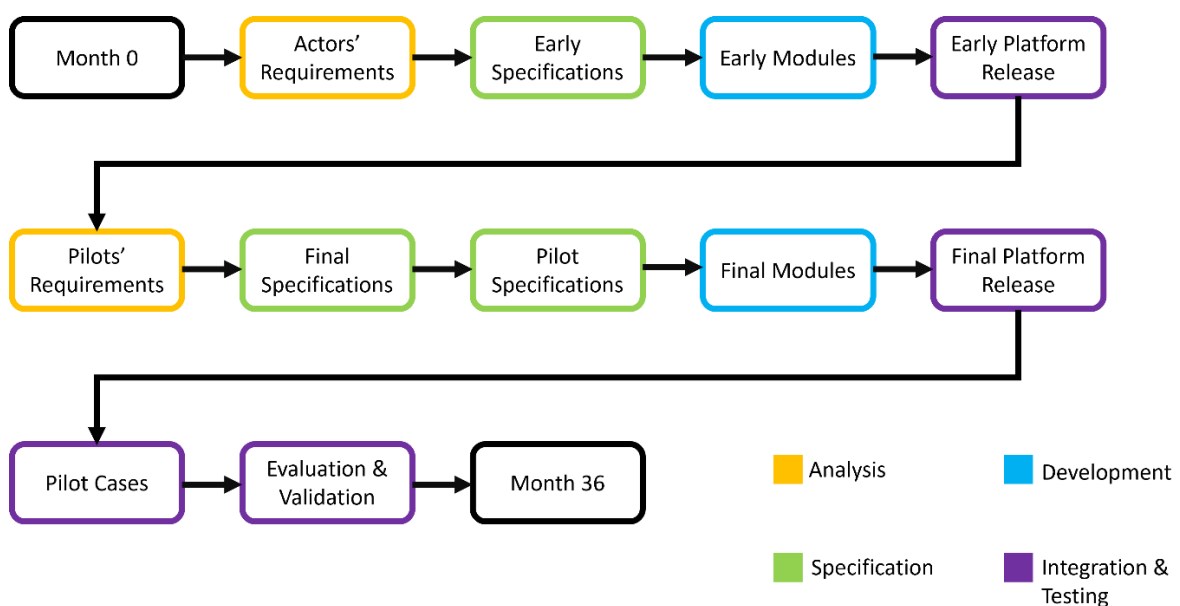


Figure 3: Integration methodology.

This methodology separates the whole procedure in four (4) sessions (differentiated by colour in the above image) that are followed repeatedly. During the first cycle, the requirements of the actors are detected in the Analysis session. Following that the initial specifications are being defined in the Specification Session. Using this information an initial, early version of all modules is being developed. Along with these modules, a first release of the platform is being released with the integration and testing of the early modules. In the second iterative process of the procedure, the preparation of the demos/pilots starts. Initially, there is the definition of the demo and pilot requirements. Following that, the final specifications including the pilot specifications are being created. At the end of the second cycle of the development section, all modules and tools are in their finalized version and after the last integration the final platform is ready.

## 2.2 Integration Timeline

The general time-scheduling for the integration process of the SCENE platform is detailed and defined in this sub chapter. The process starts at M16 and concludes at M33 as shown in Figure 4 below. This period also coincides with the timespan that WP5 is active.

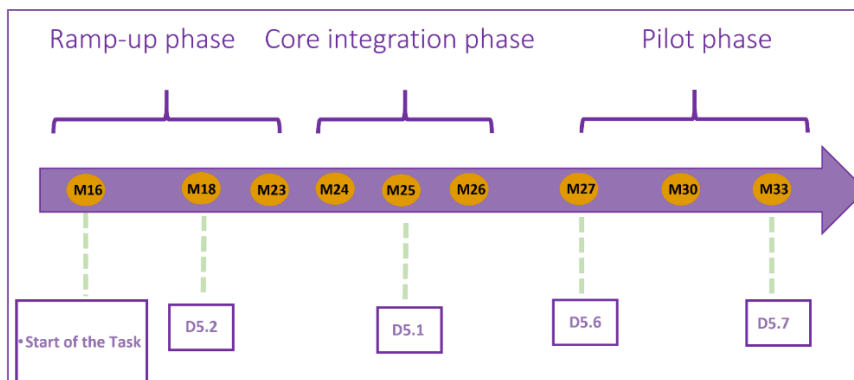


Figure 4: Integration timeline.

The aforementioned period (M16-M33) has been split in three distinct sub-periods. The aim of this is to facilitate the integration process and structure the management of the integration activities in a more structured, definite and easier manner.

### 2.2.1 Ramp-up phase (standalone tools)

The ramp-up phase is the preparatory phase of the integration period for the SCENE platform. It spans from M16 through M23 as displayed in Figure 5.

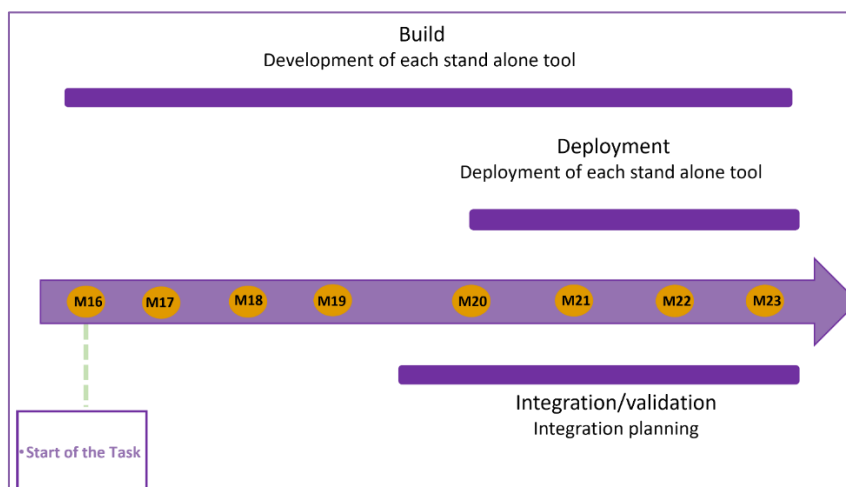


Figure 5: Ramp-up timeline.

The activities that have been carried out during the aforementioned period are presented in Table 1.

Table 1: Ramp-up activities.

Step	Activities
Build	<ul style="list-style-type: none"> <li>• Development of each standalone tool</li> <li>• Front/Back-end development of each tool</li> </ul>
Integration/Validation	<ul style="list-style-type: none"> <li>• Integration methodology</li> <li>• Identification of specifications &amp; requirements for integration</li> <li>• Definition and implementation of the tools' interfaces</li> <li>• Implementation of the single sign on mechanism</li> </ul>
Deployment	<ul style="list-style-type: none"> <li>• Deployment of standalone tools</li> </ul>

### 2.2.2 Core Integration phase

The second stage focuses on the core integration process, during which independently developed tools, spanning a variety of applications across pre-production, production, post-production and distribution phases, will be incorporated into the SCENE platform prototype. This integration will align with the platform's defined specifications and architectural framework. Scheduled to occur between months M24 and M26 this phase is outlined in Figure 6.

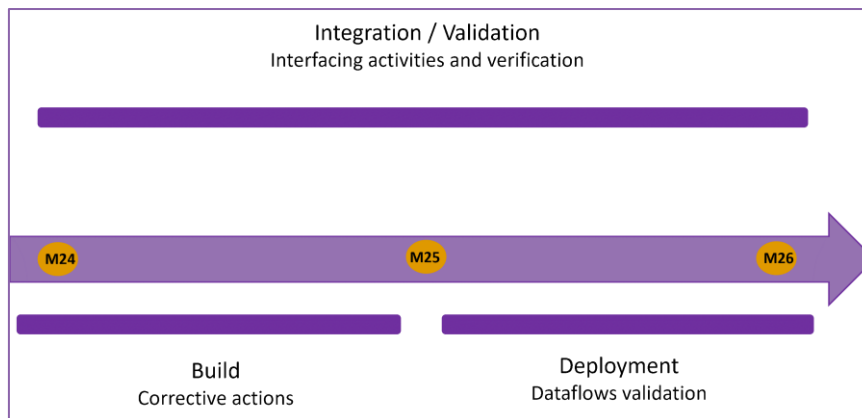


Figure 6: Core phase timeline.

The activities carried out during the core integration phase are presented in Table 2.

Table 2: Core integration activities.

Step	Activities
Build	<ul style="list-style-type: none"> <li>• Corrective action per tool and the complete platform</li> </ul>
Integration/Validation	<ul style="list-style-type: none"> <li>• Finalization/Development of the common front end</li> <li>• Testing of the implemented interfaces</li> </ul>
Deployment	<ul style="list-style-type: none"> <li>• Validation of the dataflows in the final integrated platform</li> </ul>

### 2.2.3 Pilot phase

Lastly, the final phase concerns the feedback and bug collection in a simulated end- user environment, through the pilot tests. After the pilot demonstrations the collected feedback is being analysed and any final corrective actions are applied on the final integrated SCENE platform. This phase spans the months between M27 and M33, as shown in Figure 7.

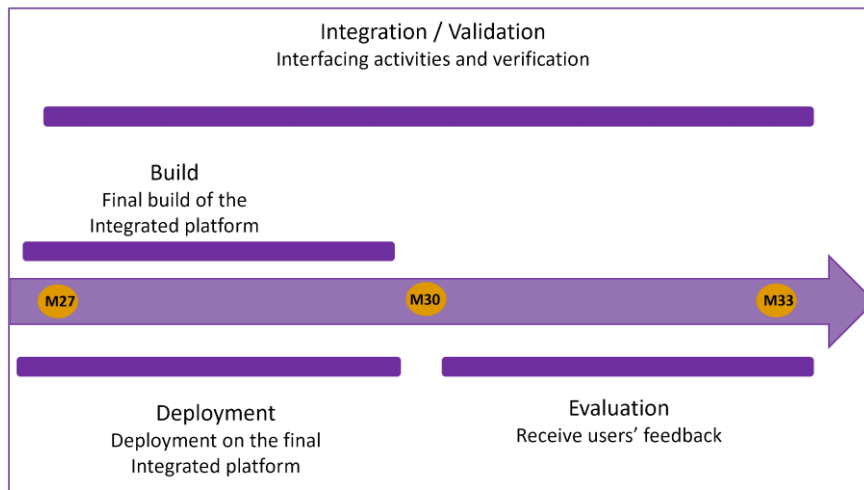


Figure 7: Pilot phase timeline.

The activities carried out during the pilot phase are presented in Table 3.

Table 3: Pilot phase activities.

Step	Activities
Build	<ul style="list-style-type: none"> <li>Final build of the platform</li> </ul>
Integration/Validation	<ul style="list-style-type: none"> <li>Corrective actions based on feedback</li> </ul>
Deployment	<ul style="list-style-type: none"> <li>Deployment of the final integrated platform</li> </ul>
Evaluation	<ul style="list-style-type: none"> <li>Evaluation of the final platform based on end users' feedback</li> </ul>

### 3 SCENE Platform Integration

The system integration activities are highly dependent on the definition of a robust and consistent architecture. This process has been completed and documented in the two related deliverables “D2.6 - SCENE Reference Architecture.R1” and “D2.7 - SCENE Reference Architecture.R2”, which include detailed diagrams and descriptions of how each component is implemented and connected with each other, as well as comprehensive descriptions on how the entire system is structured and operates. The conceptual architecture, presented in Figure 8, provides a high-level viewpoint of the entire system, offering an outline of the system’s components, their interactions and the overall design principles of their integration. As seen in the figure, the system consists of three distinct layers, containing different perspectives of its design, all of them described below.

Starting from the bottom, the **Data layer** is responsible for all the storage and retrieval of information in the system. The Data Lake is the core component in this layer, which enables the storage of various different types of media (images, videos, audio, metadata, etc.) and the exchange of them between the rest of the components, while also providing multiple tools for data analysis and processing. The database of the Blockchain technologies is also contained in the Data layer, which enables content authentication and tokenization of media.

The **Application layer** contains the majority of the algorithmic components of the system, providing the features that SCENE offers to the users. Components like the Location Scouting tool, Media Asset Manager, Audience Building tool, Audio and Lighting Simulation modules, post-production effects, Distribution and Recommendation systems are all included in the Application layer. The components in this layer can be in

turn grouped, based on the filmmaking phase they are intended to be applied to, namely the pre-production, production, post-production and distribution phases. An extra “horizontal” phase contains tools that are applied across multiple phases and are used by different components.

Finally, the **Presentation layer** is the main interface of the SCENE platform, from which the users can access the available components, depending on their role. The front-end interfaces of the individual components are grouped and accessed via a unified graphical interface, consisting the SCENE platform interface.

The system follows a distributed deployment plan, based on which the individual components are deployed in multiple different hosts and communicate by their dedicated interfaces. This helps the agile implementation and testing of the individual components, isolating their hardware and software requirements. Overall, the SCENE platform is designed to support multiple different interconnecting components and tools, offering diverse application scenarios, through a robust and flexible architecture.

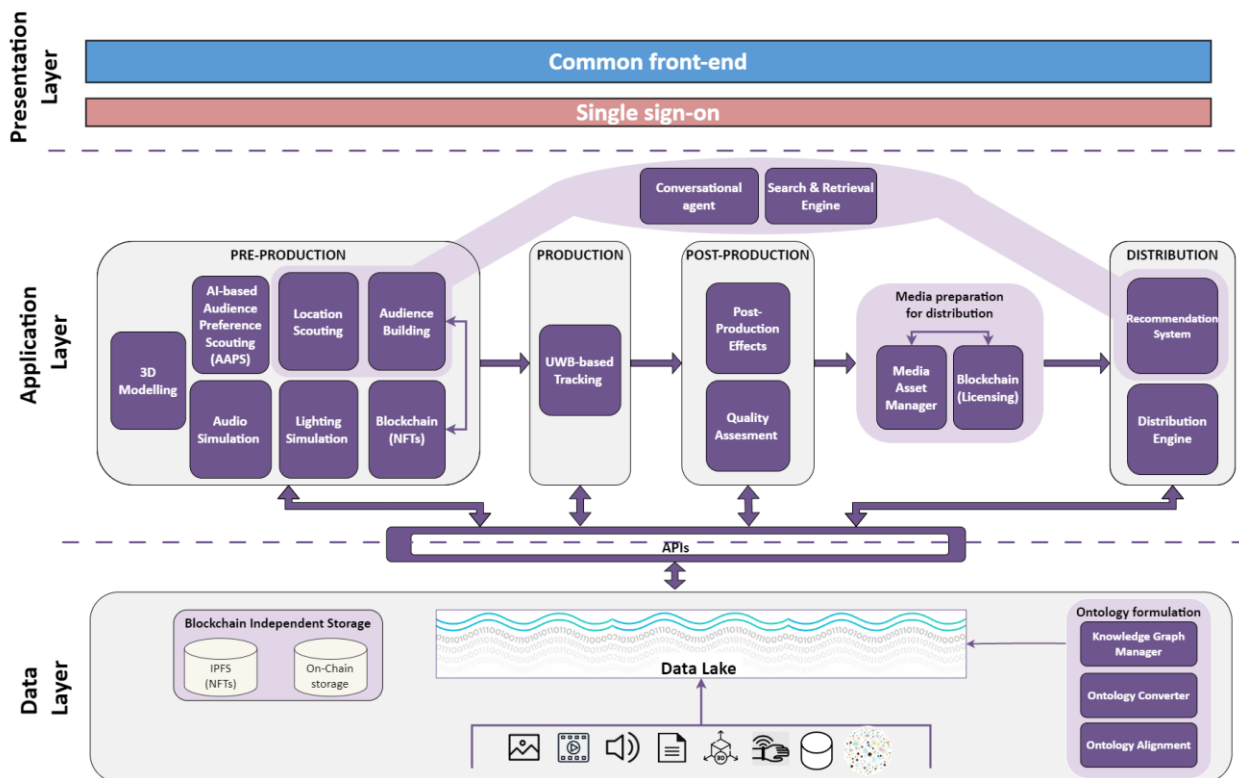


Figure 8: SCENE conceptual architecture

### 3.1 Component interconnections

Table 4 illustrates the interconnections between components in the SCENE system. The non-empty cells of the table describe components with direct communication where the directions of the data flows are indicated by the arrows.

↑ Indicates that the direction of the information’s flow is from the tool of the corresponding row to the tool of the corresponding column.

↓ Indicates that the direction of the information’s flow is from the tool of the corresponding column to the tool of the corresponding row.

Table 4: SCENE component interconnections.

	DL	OF	CA	MAM	3D	BC	LST	AAPS	ABT	LSM	ASM	PEQM	UWB	DE	RS
Data Lake (DL)		↓	↑	↓↑	↓↑		↓↑	↓↑	↓↑	↓↑	↓↑	↓↑	↓	↓↑	↓↑
Ontology formulation (OF)	↑		↓	↓			↓		↓	↓	↓				↓
Conversational agent (CA)	↓	↑					↓		↓						↓
Media Asset Manager (MAM)	↓↑	↑				↓↑		↑	↑		↓	↓		↓↑	
3D Reconstruction (3D)	↓↑						↑			↑	↑		↑		
Blockchain technologies (BC)				↓↑					↓↑					↓↑	
Location Scouting tool (LST)	↓↑	↑	↑		↓					↑					
AI-based Audience Preferences Scouting tool (AAPS)	↓↑			↓					↓					↓	
Audience Building tool (ABT)	↓↑	↑	↑	↓		↓↑		↑							
Lighting Simulation module (LSM)	↓↑	↑			↓		↓								
Audio Simulation module (ASM)	↓↑	↑		↑	↓										
Post-production effects & Quality metrics (PEQM)	↓↑			↑											
UWB-tracking system (UWB)	↑				↓										
Distribution Engine (DE)	↓↑			↓↑	↓↑			↑							↓↑
Recommendation System (RS)	↓↑	↑	↑											↓↑	

### 3.1.1 Data Lake

The Data Lake represents the main data layer of the SCENE platform and a core component in the SCENE’s system. It is responsible for the storage and management of all the data that is used and exchanged by the algorithmic components. For this reason, the Data Lake is connected with most of the components, either for providing a storage space for their generated data or for retrieving and utilizing data, stored by other components or users.

### 3.1.2 Ontology formulation

The Ontology formulation component provides a set of functionalities for the utilization of the SCENE-O ontology. It mainly interacts with the Data Lake, by storing the semantically annotated information of the ontology. The main functionality lies on the use of the search and retrieval engine, which enables the semantic exploration of stored information about movies, as well as data generated by other components, such as the Conversational agent, Location Scouting and Audience Building tools, the Audio and Lighting Simulation modules, the Recommendation System and the Media Asset Manager, and are related with the filmmaking process.

#### 1.1.1 Conversational agent

The Conversational agent component is a versatile tool that is intertwined with various other components of the system. It is closely related to the Ontology formulation component, since it utilizes the semantic layer



for the retrieval of information from the Data Lake. Moreover, it collaborates with the Location Scouting tool, the Audience Building tool and the Recommendation System, which leverage the AI capabilities the Conversational agent offers.

### 3.1.3 Media Asset Manager

The Media Asset Manager is responsible for the management of video assets, handling metadata, categorization and other properties. Components working with video media, like the AI-based Audience Preference Scouting tool, utilize the MAM to process the videos, which are associated with the metadata defined in the SCENE-O ontology, formed by the Ontology formulation component and are finally stored in the Data Lake. The generated metadata of MAM can be utilized by AI-based Audience Preferences Scouting tool and Audience Building tool. Additional access to the Blockchain network of the system ensures data integrity and secure records for the media content.

### 3.1.4 3D Reconstruction

The 3D Reconstruction component utilizes the latest machine learning methods to generate a 3D model of a location, by using 2D image views of it. The inputs and outputs of the components are retrieved and stored to the Data Lake, so that they can be explored by other related components. The rendered views of the location can be directly passed to the Lighting Simulation module to modify their lighting conditions, while the Location Scouting tool uses the generated 3D models of the locations to enhance its capabilities. Additionally, the Audio Simulation module uses the 3D reconstruction to generate a blueprint of the given location, based on which the audio experiments will be made. Finally, this component can collaborate with the UWB-based Tracking System, in order to provide the map of the area, where the actor positions will be tracked.

### 3.1.5 Blockchain technologies

The Blockchain network implemented under the scope of this project covers two main functionalities. The first one is connected directly to the Media Asset Manager and aims for the licensing preservation and generation of contracts between producers and consumers, about the media being transferred. The second one is responsible for the creation, ownership and transfer of non-fungible tokens (NFTs) of content for production crowdfunding purposes and it interacts with the content produced by Audience Building tool.

### 3.1.6 Location Scouting tool

The Location Scouting tool component encompasses a collection of tools that help the exploration for the most suitable filming location by filmmakers, depending on various factors, such as its visual appeal, accessibility and availability of services. The component supports the registration of new locations, which are adapted to the SCENE-O ontology, through the Ontology formulation features, and stored in turn into the Data Lake. The Conversational agent can also be directly accessed by this component, to deliver useful information to the users, answering their queries. The Location Scouting tool can additionally interact with the 3D Reconstruction module, by enhancing the locations with 3D models of them, and the Lighting Simulation module, in order to simulate how the location would look like under different times of the day or with added light effects.

### 3.1.7 AI-based Audience Preferences Scouting tool

The AI-based Audience Preferences Scouting (AAPS) tool mainly consists of two major tools: the Preference Forecaster and the Sentiment Analysis tools. The former tool trains and utilizes an AI model to predict trends of viewing preferences, based on textual metadata that are stored in the Data Lake. The latter tool enables the measurement of the reaction of audience members, when presented with a video content. The video is retrieved from the Data Lake, after categorized and tagged by the Media Asset Manager, and in combination



with audience metadata produced by the Audience Building tool and Distribution Engine provide useful feedback about audience preferences to the filmmakers.

### 3.1.8 Audience Building Tool

The Audience Building Tool assists filmmakers to understand audience interests and engage viewers, by sharing campaigns and contests about a film on social media. The information collected, transformed as metadata, are used by the AI-based Audience Preferences Scouting tool to analyze the reactions of viewers and classify them into separate categories. The data is stored into the Data Lake, after semantically annotated by the Ontology formulation. The integration to the Conversational agent enables the support of AI-powered functionalities, making it possible for the users to chat with an LLM model. The Audience Building Tool also offers crowdfunding capabilities, by connecting to the system's Blockchain network, to author NFTs that can be bought by interested users to support the film's production.

### 3.1.9 Lighting Simulation module

The Lighting Simulation module is a versatile component, which provides the filmmakers the ability to meddle with the lighting conditions of an image. Different times of the day or artificial light effects can be simulated on a given location, through the use of generative AI. The component retrieves its input and stores the generated output in the Data Lake, after being annotated by the Ontology formulation. Rendered views of 3D reconstructed locations can be also used as input to the component. Moreover, the Location Scouting tool can interact with it, in order to give users the ability to directly experiment with a location they are exploring.

### 3.1.10 Audio Simulation module

The Audio Simulation module assists the filmmakers with the exploration of the acoustic conditions of a location, in order to be better prepared during shooting. The audio model and location can be retrieved from the Data Lake and used through the UI. This component additionally supports the evaluation of the audio of video media, by calculating various quality metrics, which are in turn use by the Media Asset Manager to enhance the video's metadata.

### 3.1.11 Post-production Effects & Quality metrics

This component contains a set of tools to enhance the post-production phase of filmmaking. It supports the retrieval of media from the Data Lake and the application of various post-production effects on them. Multiple metrics that measure the visual quality of the processed media are then calculated, in order to compare between them and the original source. The resulted outputs can then in turn stored back to the Data Lake.

### 3.1.12 UWB-based Tracking System

This component provides a novel system to monitor the position and movement of an actor in an area during production. The system uses the 3D models generated by the 3D Reconstruction component to retrieve the area's map and track the actors' positions in space. The actor positions estimated at each timestamp are then stored into the Data Lake, along with additional useful information.

### 3.1.13 Distribution Engine

The Distribution Engine is responsible for managing the way production-related content is distributed and presented to the end-users. It is closely related to the Media Asset Manager, utilizing it to retrieve assets associated with specific users or audiences and the aforementioned Blockchain network, storing and managing licensing information and permissions. The generated output of this component is utilized by AI-

based Audience Preferences Scouting tool, in order to analyse the audience engagement, and the Recommendation System, to provide accurate recommendations about movies.

### 3.1.14 Recommendation System

The Recommendation System supports the generation of accurate recommendations about movies and documentaries to the end-users, based on their preferences and film attributes. The communication with the Distribution Engine provides the information about the audience preferences, while the Data Lake stores the relevant information about available movies, after being semantically annotated by the Ontology formulation. It also utilizes the functionalities of the Conversational agent of the system, in order to provide AI-enhanced recommendations in natural language.

## 3.2 SCENE Platform interface

The SCENE platform interface (Figure 9) is designed to serve as a centralized gateway that integrates the diverse front-end UIs of multiple system components—ranging from media asset management and 3D reconstruction to audience building and distribution—into a single, cohesive environment. Central to this integration is a single sign-on (SSO) mechanism that leverages industry-standard protocols such as SAML 2.0, OAuth 2.0, and OpenID Connect. Upon authentication, a secure token is issued by a centralized identity provider (e.g., Keycloak or a similar enterprise solution), which is then propagated through secure API gateways to all microservices. This token-based approach not only ensures encrypted, seamless session management but also assigns each user a specific filmmaking role (director, producer, director of photography, distributor, etc.), thereby tailoring the user experience by dynamically controlling access to role-specific functionalities.

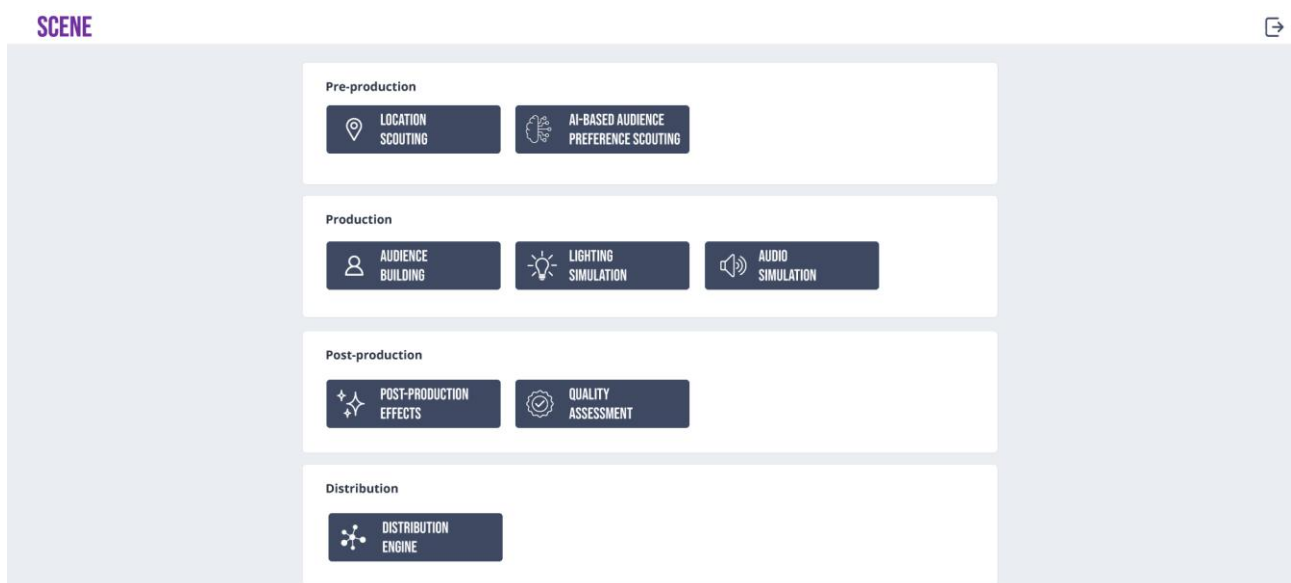


Figure 9: SCENE platform mock-up.

Behind the unified interface, the platform’s microservices architecture underpins secure, real-time communication and data exchange among components via standardized APIs. This modular design allows each tool to be deployed and scaled independently, while still maintaining robust role-based access control across the system. Detailed UI mockups provide clear visual representations of the interface layout, navigation flows, and interactive elements. These mockups illustrate how users will interact with the platform and serve as key references for stakeholders to validate the design and ensure that the integration of various components meets both technical and operational requirements.

## 4 Testing Process Management

An apparent need emerges from the basic integration steps that were described in the beginning of Section 2. The “Integration and Testing” step requires the use of additional and specific tools and methodologies that are geared towards the Testing “sub step”. Considering this, the present section details some widely known tools and methodologies that are available and can be used by each partner in order to facilitate the Testing Process during the SCENE’s platform integration.

### 4.1 Software Quality Assurance

The requirements of a software quality model are a required necessity for the procedure of the tests. Figure 10 displays several distinct perspectives on software quality [1]:

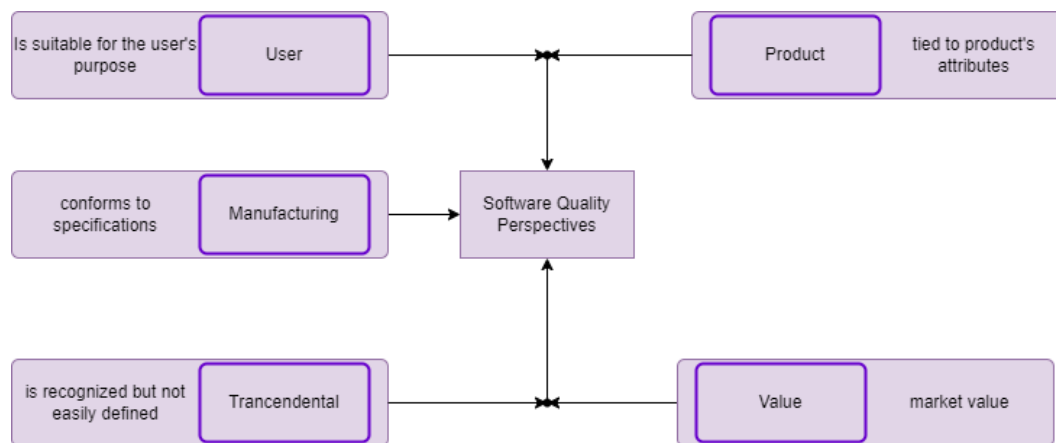


Figure 10: Several perspectives on software quality.

Both the developer’s partners’ perspective and the end users’ perspective is included in the technical quality concept that is adopted by the SCENE project. By considering both the technical insights from the developer partners and the practical experience of end users, the SCENE project creates a balanced framework that bridges abstract quality standards with real-world usability. This integrative approach improves the system’s robustness and promotes continuous feedback, ensuring that the final SCENE platform aligns with evolving stakeholder expectations.

#### 4.1.1 Software Quality Models

SCENE developers need a comprehensive model for the early evaluation of software quality. There are many software product quality models that define several attributes concerning software quality.

Three commonly used models are summarized below:

- **McCall's model** [2] includes eleven criteria encompassing product operation, revision & transition.
- **Boehm's model** [3] is based on a wider range of characteristics and includes nineteen dependent criteria that interact with each other and often cause conflict. Conflicts happen especially when these criteria are incorporated into the software development process.
- **FURPS model** [4] stands for functionality, usability, reliability, performance and supportability. These characteristics can be used as both product requirements and in product quality assessment.
- **Dromey's model** [5] consists of three principal elements, namely 4 product properties, 13 quality attributes and means of linking them.

- **ISO/IEC 25010 model** [6] includes eight quality characteristics, each having a large number of attributes.

The criteria and goals defined in each of these models are listed in Table 5. Note that the ISO Model includes a number of criteria under its characteristics of maintainability.

*Table 5: Software Quality Models.*

Criteria/Characteristics	McCall	Boehm	FURPS	Dromey	ISO/IEC 25010
Correctness	X	X		X	maintainability
Reliability	X	X	X	X	X
Integrity	X	X			
Usability	X	X	X	X	X
Efficiency	X	X	X	X	X
Maintainability	X	X	X	X	X
Testability	X		X		maintainability
Interoperability	X				
Flexibility	X	X			
Reusability	X	X		X	
Portability	X	X		X	X
Clarity		X			
Modifiability		X			maintainability
Documentation		X			
Resilience					
Understandability		X			
Validity		X			maintainability
Functionality			X	X	X
Generality		X			
Economy		X			

The criteria in McCall's and Boehm's models are interconnected and often conflict with one another, particularly when software providers attempt to integrate them into the development process. Among these models, the ISO 25010 model is the most up-to-date and comprehensive, making it the preferred choice for the quality assurance and validation of the SCENE platform prototype. To implement this, we strive to incorporate certain quality model characteristics into our quality analysis methodology, as will be described in section 5.2 below.

### 4.1.2 The ISO/IEC 25010 Model

The software product quality encompasses three gradually dependent dimensions in the ISO/IEC 25010 model:

- **“Internal quality”** refers to the developer perspective on static aspects of the system, for example the architecture & implementation of the system.
- **“External quality”** refers to the externally observed behaviour and functioning of the system either when it is executed or it is hosted in a simulated/controlled environment.
- **“Quality in use”** refers to user's perception of the system quality, while achieving goals in a particular environment/context of use.

The *quality in use* conveys user's perspective on the system quality. Figure 11 summarizes the characteristics of the *quality in use* that refers to system's effective usage and perception by the end user. *Quality in use* can be used complementary to considering the quality of the system to refer to the overall capability of the system to enable specified users to achieve goals concerning its quality characteristics, such as effectiveness, productivity, safety, satisfaction & usability. The *quality in use* is measured by the results of its concrete usage in context, rather than by properties of the software itself. Its evaluation will involve participation of developers working with the SCENE platform.

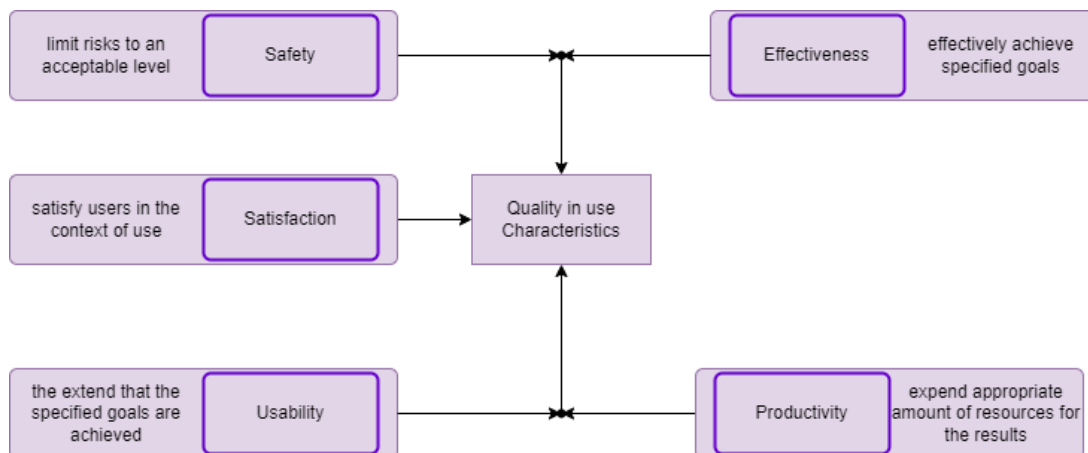


Figure 11: Quality in use characteristics.

## 4.2 Testing Methodology

The testing approach that is followed during the various software development phases depends on a wide range of issues such as [7]:

- the time point within the integration procedure and the regularity that the testing will be performed,
- the responsible people for the testing (developers, independent testers),
- the kind of pieces that will be tested (samples, everything, nothing),
- the level of knowledge about the component under testing (black-box or white-box testing),
- the extent to which the testing will be performed (exhaustive testing, no testing)

In addition to these considerations, the chosen testing approach is also influenced by the development methodology. SCENE's development follows a continuous, iterative, and incremental process, with different partners regularly integrating their components through multiple cycles of analysis, development, and

validation. Given this approach, automated or manual testing is essential whenever feasible, with its scope adjusted based on the platform’s evolving needs. For instance, a demonstrator application is primarily intended to showcase partial functionality, making full-system testing unnecessary at that stage. Overall, a suitable testing methodology that aligns with SCENE’s development approach is based on agile or lean principles [8].

#### 4.2.1 SCENE’s Agile Testing Levels

There are four (4) different levels of testing in the agile approach [9]:

1. **Unit testing:** Verifying each part of the software by isolating it and then testing each individual component in terms of fulfilling requirements & the desired functionality.
2. **Integration testing:** Testing of the communication between different parts of the system in combination to assess if they work correctly together.
3. **System testing or Developers Acceptance Testing (DAT):** Testing of the entire system as a whole.
4. **Validation:** Evaluation at the final phases of development and integration to ensure requirements fulfilment. In the SCENE context, the validation will be done through Pilots Acceptance Testing (PAT) during the execution of the SCENE demonstrators.

In this respect, Figure 12 presents the testing levels of the agile methodology. Here the importance of the sequence among the testing levels should be noted.

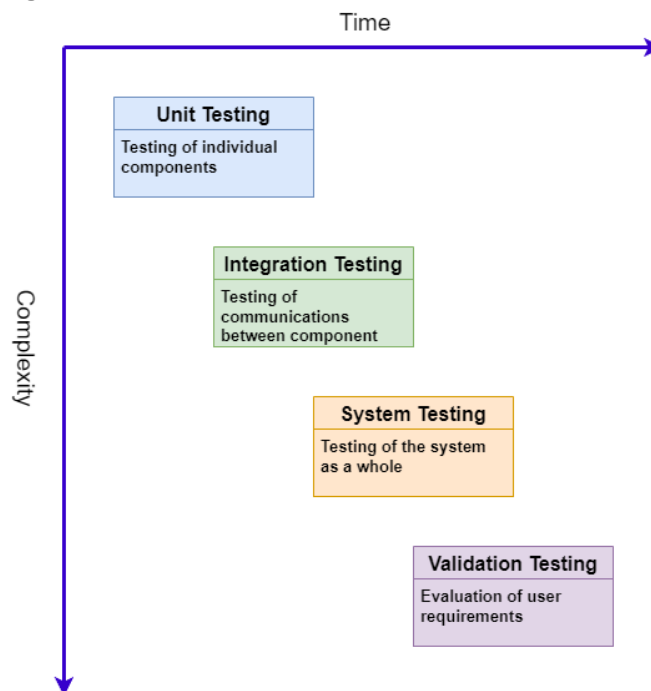


Figure 12: Agile Testing Levels.

Software testing should follow a structured approach, beginning with unit testing, followed by integration testing, and then system testing (DAT) to evaluate the fully integrated platform from the developers' perspective. Finally, the system will undergo testing and evaluation in a simulated user environment during demos and pilots (PAT/Demo), as outlined in D5.3 and D5.2. Since SCENE follows an iterative development model with two prototyping cycles, some testing activities may overlap or require revisiting previous stages. Unit testing will be conducted for each standalone tool within the SCENE platform prototype during the project’s ramp-up phase, which falls under WP3 and will not be further detailed in this document. Similarly,

system testing and validation of integrated business scenarios and key performance indicators (KPIs) will be carried out as part of the demo/pilot testing, which will be covered in D5.3 and is not further discussed here. The next section outlines the integration testing approach for SCENE.

### 4.2.2 SCENE’s Integration Testing

The plan of the SCENE integration testing is displayed below. The testing activities, the frequency of the testing and the relevant responsibilities concerning the testing process are summarized.

Table 6: Software Integration Test Plan.

Level of testing	Activities	Test environment	Frequency of testing	Responsible		
				Writing test cases	Providing test data	Running tests
Integration	<ul style="list-style-type: none"> <li>Select test cases.</li> <li>Manage unit dependencies.</li> <li>Write automated test cases.</li> <li>Prepare non-automated test cases.</li> </ul>	Continuous Integration Infrastructure	<ul style="list-style-type: none"> <li>Automated tests run continuously, when component are built on the Continuous Integration.</li> <li>Manual tests, at least each iteration.</li> </ul>	Component Provider, whose component is calling another component.	Integration Team	Integration Team

### 4.2.3 Integration testing approach

In the context of the SCENE project, there is the need to test over several work packages through an integration test. A complete workflow has to be tested and the results of the test need to be checked. The implementation and the workflow structure are partially known and considered for the integration test. The test can be either manual or automated, depending on developer’s preferences.

The main plan that is proposed for continuous integration and testing:

- 1. Define integration tests for individual components:** Prepare integration tests according to plan for each component.
- 2. Create a pipeline for each test:** In a pipeline, one can define different steps. When a step succeeds, the pipeline moves onto the next step. When a step fails to execute correctly, the entire pipeline fails. Steps contain the actions to be executed at each of the continuous integration stages (Build-Test-Deploy).
- 3. Define an execution environment:** Using Docker images and containers the tests can be run. This allows the pipeline to define the environment and tools required without having to configure various system tools and dependencies manually.
- 4. Record test results:** When there are test failures, it is often useful to grab files generated during the execution of the pipeline to analyse and investigate.
- 5. Deployment:** Docker can be used in the final deployment through Dockerfiles.



#### 4.2.4 Manual Testing

Although automated testing plays a critical role in the continuous integration pipeline, manual testing remains crucial, especially for aspects that require human judgment. Manual testing in the SCENE platform integration encompasses several key areas:

- **Exploratory testing:** Manual testers conduct exploratory sessions to uncover issues that automated tests might overlook. This involves navigating through the platform's front-end interfaces, interacting with components in ways that simulate real-world usage and identifying usability issues or unexpected behaviors. Exploratory testing is particularly valuable during early pilot phases, when new features are first integrated.
- **Usability and User Interface evaluation:** Testers assess the user experience, ensuring that the front-end interfaces are intuitive, accessible, and consistent with the design specifications. This includes verifying that user interactions (e.g., navigation flows, form validations, error messages) meet both functional requirements and quality standards. Feedback gathered during these sessions is essential for refining UI elements and overall platform ergonomics.
- **Regression testing:** With every new integration or bug fix, manual regression tests are performed to validate that existing functionalities remain unaffected. Testers use detailed checklists and pre-defined test cases, often maintained in a test management tool, to systematically verify that previously resolved issues do not recur.
- **Scenario-based and End-to-End validation:** Manual testing sessions are organized around complete user workflows, simulating typical use cases that span multiple components. For instance, testers might validate a full scenario from logging into the SCENE platform, uploading a media asset via the Media Asset Manager, to verifying its correct processing in the Data Lake and subsequent retrieval by the Distribution Engine. These sessions ensure that the entire integration chain functions seamlessly.
- **Cross-platform verification:** Given the distributed nature of the SCENE platform, manual testers also verify functionality across different environments (e.g., various browsers, devices, or operating systems) to ensure consistent performance.

By complementing automated tests, manual testing ensures that the platform not only meets technical specifications but also delivers a high-quality, user-friendly experience. Its iterative application during each integration phase is critical for capturing subtle issues, validating usability, and providing a final quality assurance check before pilot deployment.

Adopting a robust test management tool, as mentioned above, can enhance the organization and traceability of manual test cases. Open source tools like Kiwi TCMS offer a modern, web-based platform that supports comprehensive test case creation, execution tracking, and detailed reporting. Kiwi TCMS's intuitive interface and REST API integrations facilitate collaboration among testers while ensuring that manual test activities are seamlessly integrated into continuous integration pipelines.

Similarly, TestLink is a mature and widely used open source test management tool that excels in managing manual test cases and plans. Its web-based architecture allows teams to create, organize, and execute test cases, as well as generate actionable reports and traceability matrices. By leveraging TestLink, the SCENE project can maintain a clear audit trail of manual test execution and efficiently capture feedback from end-users, ensuring that all aspects of the integrated system meet the specified quality requirements.



We plan to explore the use of Kiwi TCMS and TestLink as potential test management solutions to support our manual testing process. As we have not yet implemented a specific tool, our objective is to evaluate both options to determine which aligns better with our testing needs, team workflow, and integration requirements. This investigation will include assessing their capabilities in test case management, execution tracking, reporting, and integration with our development pipeline. In case we decide to adopt either of these tools (or other similar), we will document our selection and implementation details in D5.7 (M33).

## 5 Integration Process Management Tools

In the current chapter, we describe the plan for the integration of the components of the SCENE platform prototype. Initially, under an administrative perspective the overall integration and testing time line is presented in Section 2. The current chapter refers to the realization of this. Section 5.1 describes the software source and executable code management as a crucial part of the continuous and iterative integration cycle. Section 5.2 details the required inspection of code quality, and finally, section 5.3 analyses the overall and holistic integration strategy for SCENE.

### 5.1 Software Configuration Management

Software Configuration Management (SCM) plays a crucial role in modern software development. Although it is often perceived as a recent concept focused primarily on versioning source files, its scope extends beyond that. At the core of SCM is a version control system, which manages the evolution of source code and other project-related documents over time. In this context, source code is viewed as a system that exists across both time and space. Files and directories represent the spatial dimension, while their progression throughout development represents the temporal aspect. A version control system facilitates navigation through this evolving structure, ensuring efficient management of changes and revisions [10].

Configuration management contains version control, and extends it by providing additional methods of project management [4]. Beyond tracking changes to source code, SCM encompasses build management, change control, release management, and configuration auditing to maintain the integrity and stability of a project. It ensures that different configurations of software, such as development, testing, and production environments, are properly maintained and reproducible. Lastly, SCM supports the continuous integration (CI) and continuous deployment (CD) with the integration of automated testing and deployment pipelines.

Software configuration consists of software configuration items (SCI). The following activities constitute configuration management [11]:

- identification of SCIs and their versions through unique identifiers,
- control of changes through developers, management or a control instance,
- accounting of the state of individual components,
- auditing of selected versions (which are scheduled for a release) by a quality control team.

The Institute of Electrical and Electronics Engineers (IEEE) sees configuration management as a discipline that uses observation and control on a technical as well as on an administrative level. Software configuration management deals with the governing of complex software systems [5].

In SCENE we base the software configuration management on Git that addresses many of the problems listed in [11] and [12]. Given the complexity of SCENE as a diverse, multi-partner project, Git provides a scalable and collaborative approach to managing source code across different teams and development environments.



By leveraging Git's branching model, we can maintain separate development, testing, and production branches, ensuring a structured workflow that supports parallel development and feature integration. Additionally, Git's ability to track changes at a granular level enhances traceability, accountability, and rollback capabilities in case of defects or regressions. Furthermore, Git's collaborative features, such as pull requests, code reviews, and issue tracking, enable better coordination among partners.

### 5.1.1 Code Management with Git

Code management of a big and collaborative project of the scale of SCENE is a hard task. The SCENE project contains thousands of lines of code, developed by different people working for different partners. Considering this, there is a requirement of a robust and efficient project management. Additionally, it was decided that a hybrid integration approach will be followed, respecting individual component specifications as well as integrated platform requirements. According to this approach:

- Particular components reside in the same server.
- Some components will reside on distinct servers (e.g. in order to address tight performance requirements that require special hardware such as servers with GPUs).
- All components will communicate with the integrated system's front end through RESTful services.

This hybrid approach has many advantages and excels at certain workflows. For instance, both workflows in which each tool behaves as a standalone and discrete entity and workflows where there is intra tool interaction will be supported in order to facilitate the film/ documentary production and wide promotion.

For components that reside in the same server, a centralized management system exists with the aim to keep track of every part of the code from all the relevant partners. This happens in order to integrate all the discrete tools into the final prototype. An important requirement is bug tracking and version control of the code, in order to be able to identify any problems that arise during development or testing. After the identification of such problems, an update can happen to the code when needed. It is common practice to immediately commit every new change to a version control system, no matter how small or unimportant it is. The main reason for this is that the other developers should always work with the latest version of the Code base. A Git repository exists for code management purposes, where all the relevant data and code files of the development of each discrete tool are stored and organised.

Git is a version control system used for software development and various version management tasks. As a distributed revision control system, it prioritizes speed, data integrity, and support for decentralized, non-linear workflows, making it an ideal solution for SCENE's code management needs. Unlike traditional client-server models, every Git directory on any computer functions as a complete repository with full history and version-tracking capabilities, operating independently of network access or a central server. Git is also open-source software, licensed under the GNU General Public License.

Git has many desirable features such as:

- Distributed development, with each developer working on a local copy.
- Non-linear development, with branching and merging.
- Cryptographic authentication of history, change or deletion of a commit is marked.
- Pluggable merge strategies, merge auto-completion model is well defined and in case of a conflict the developer can manually merge the files.

The development tracking tool has integrated project management capabilities, allowing code updates to be monitored when necessary. SCENE's partners will receive server access rights to submit code, libraries, or executable files, facilitating the integration of all modules into a centralized repository.

It is crucial to ensure that a revision repository includes all essential files required to build the software system but excludes any files that can be automatically generated from others. Storing redundant files unnecessarily clutters the repository and is considered poor practice [13].

## 5.2 Code Analysis & Quality Measurement

There are two main kinds of analytic software quality assurance:

- Static analysis
- Dynamic analysis

**Static software analysis** examines development-time characteristics that do not require the software to be executed. Common static analysis techniques include software metrics, informal methods such as code inspections and reviews, and compliance checks with coding standards. Software metrics assess various aspects of source code quality, serving as indicators for potential quality concerns (e.g., Source Lines of Code - SLOC). Code inspections often apply predefined rules to evaluate adherence to coding conventions and best practices for a given programming language. This approach can help identify potential issues such as infinite recursion, dependency loops, low test coverage, or deviations from coding standards.

**Dynamic analysis**, on the other hand, involves testing the software during execution. This method requires compiling the source code into an executable form and running it to verify that the program behaves as expected.

Since the SCENE project is an Innovation Project, most of its development starts with a high Technology Readiness Level (TRL), meaning it builds on an existing infrastructure and prior software development. As a result, traditional code inspection methods may be unnecessary. However, if required, SonarQube (formerly "Sonar") will be used in SCENE to monitor code quality. SonarQube is a web-based, open-source quality management platform that supports multiple programming languages and can be extended through plugins.

SonarQube evaluates software quality across seven key dimensions, considering both practical and academic perspectives, as illustrated in Figure 13.

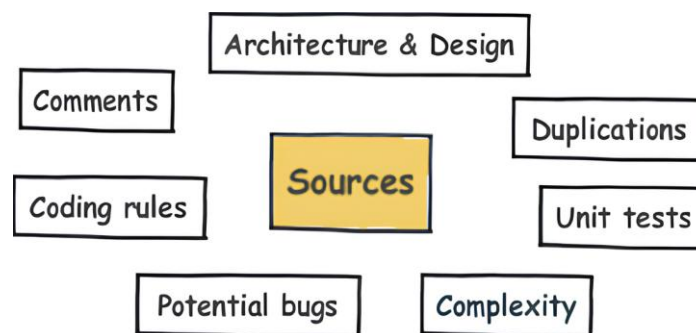


Figure 13: Software quality dimensions supported by SonarQube

Additionally, there are powerful plugins available to analyse advanced metrics such as quality index, technical debt, and overall project quality. The *Quality Index* metric provides a comprehensive assessment by evaluating coding rules, style, complexity, and unit test coverage. *Technical debt* estimates the maintenance effort required for a project, breaking it down by factors such as code duplication, documentation, test

coverage, and complexity. The *Total Quality* metric offers a holistic evaluation of the project’s quality by integrating measurements related to code quality, design, architecture, and unit testing.

### 5.3 Integration Strategies

The SCENE platform prototype integrates all its tools through a continuous process that strengthens their interconnection at every stage (with the final component integration illustrated in Figure 4). This integration follows a recurring six-step cycle, outlined in detail in Section 5.4. The initial phase of integration requires each component to be fully developed and tested independently before being linked with other components to form the complete, fully functional SCENE platform prototype. Until all components are ready, the platform will utilize temporary virtual components (placeholders) to stand in for actual ones. These placeholders either produce constant values or mimic the real component’s behavior as accurately as possible. Depending on the testing strategy, these virtual components may take the form of test drivers or stubs.

The use of placeholders is particularly beneficial for system testing before the actual components are interconnected. Various integration strategies for this process are presented in Figure 14.

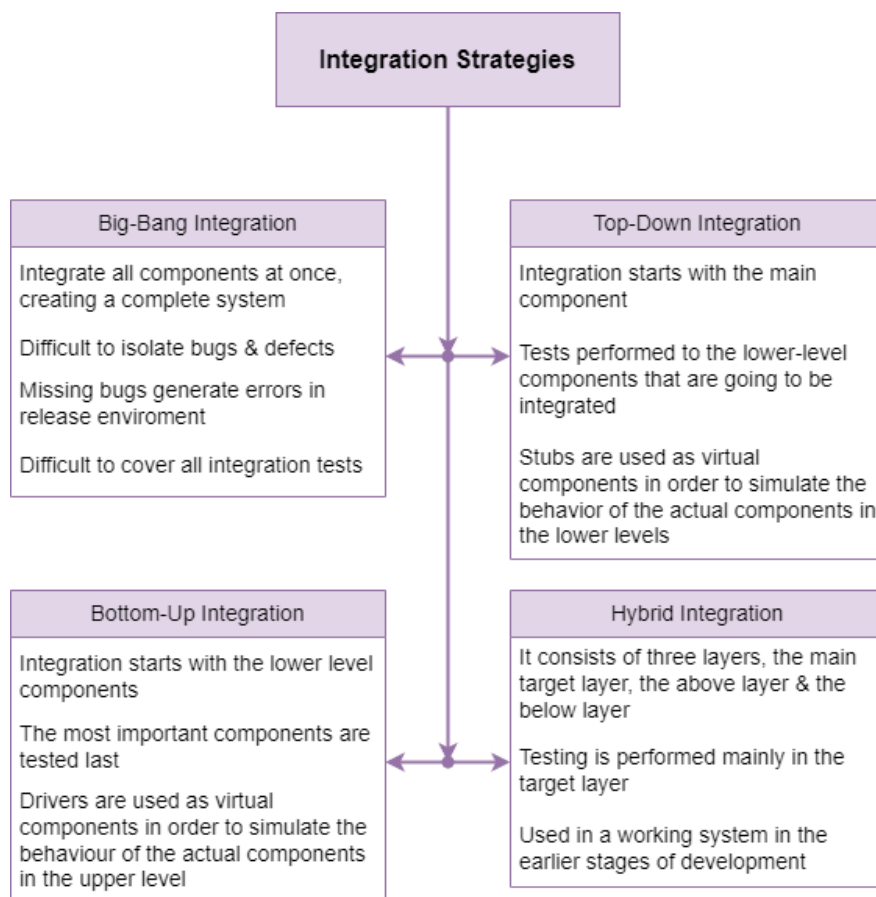


Figure 14: Integration Strategies.

The SCENE platform follows a systematic approach that leverages temporary virtual components when necessary, allowing for early-stage testing and validation before full integration. By adopting a combination of Big-Bang, Top-Down, Bottom-Up, and Hybrid Integration strategies, the platform can balance rapid development with thorough testing, ensuring that issues are identified and resolved early in the process. Ultimately, this integration methodology enhances the platform’s scalability, maintainability, and reliability, supporting a smooth transition from individual component testing to a fully functional system

## 5.4 Integration Process of the SCENE Platform

The system will adopt a continuous integration approach, where components are incrementally incorporated into the platform. This methodology demands meticulous and strategic design, typically involving practices such as automated builds, unit testing, deployment, notifications, and reporting. Crucially, the integration process must be structured to enable timely resolution of issues from the moment they arise, prioritizing efficiency. The SCENE integration framework follows a cyclical sequence of steps, as illustrated in the accompanying figure.

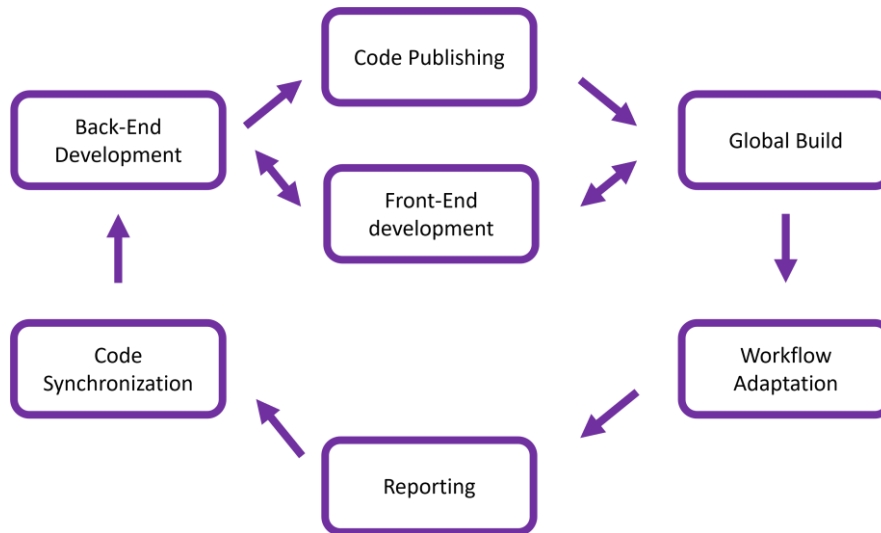


Figure 15: SCENE's Integration Process.

### 5.4.1 Code Synchronization

The first step in the continuous integration process involves utilizing a revision control system, allowing developers to maintain working copies of the source code throughout integration. Developers must regularly synchronize their local code files with the central repository. They retrieve a working copy from the main branch (or master) of the revision control system, ensuring they have the latest integrated version of the source code on their local machine. This local copy is then branched, enabling modifications necessary to implement the required features for the overall software system.

### 5.4.2 Front-end & Back-end Development

The development process focuses on building the front-end and back-end of various SCENE tools. Front-end and back-end developers alternate between implementing new tests and features and refactoring code to enhance its structure and clarity. Refactoring involves modifying the code without altering its external behavior. Local source code changes must be synchronized with the repository, and if updates exist in the repository, they should be merged accordingly. Developers are responsible for ensuring that any new features they introduce do not cause bugs. To verify this, an automated build should be executed on their development machine to compile and link the code into an executable program. Additionally, automated or manual tests should be performed to assess functionality, security, load handling, and performance.

### 5.4.3 Code Publishing

If the local build completes successfully, the developer commits their changes to the software revision system, publishing the updated source code. If new changes have already been committed to the repository, the source code must be synchronized again before submission. This ensures optimal code publishing.

### 5.4.4 Work-flow Adaptation

In addition to automated testing, the common software platform is evaluated using specific workflows that cover a subset of SCENE's functionalities, as outlined in the project's demo and pilot use cases. Any errors or



malfunctions that deviate from the expected behaviour in these use cases are documented and reported to the development team. The integration system must be capable of adapting to, displaying, and processing data according to the workflows defined by the use cases.

## 6 Conclusion & future work

In conclusion, the SCENE testing and system integration process detailed in this document demonstrates a comprehensive and methodical approach to validating the SCENE platform's architecture. Through a carefully structured integration methodology and the combined use of automated and manual testing strategies, the SCENE team has successfully ensured that all components communicate seamlessly and perform reliably. Rigorous planning, execution, and continuous feedback mechanisms have enabled early detection and resolution of defects, resulting in a stable, secure and scalable environment that meets both functional and non-functional requirements.

Looking ahead, the integration framework laid out in this document not only establishes the current pilot deployment, but also provides a robust foundation for future enhancements. The iterative and adaptive nature of the testing process will support ongoing refinements based on real-world performance and evolving user needs. As the platform transitions from integration to full-scale operation, the documented strategies and lessons learned will continue to guide improvements, ensuring that the SCENE system remains at the forefront of innovation in cognitive film production and distribution. The second version of this deliverable, namely "D5.7 - SCENE testing & system integration.R2", will present the updates undergone during the next phase of the integration and testing process.



## References

- [1] B. Kitchenham and S. L. Pfleeger, "Software quality: the elusive target," *IEEE Software*, pp. 12-21, 1996.
- [2] J. McCall, P. Richards and F. Walters, "Factors in Software Quality concept and definitions of software quality," *RADC TR-77-369*, 1977.
- [3] B. Boehm, "Software Risk Management," in *IEEE Computer Society Press, CA*, 1989.
- [4] G. Versteegen and G. Weischedel, *Konfigurationsmanagement*, Springer, 2003.
- [5] B. Westfechtel and R. Conradi, "Software Architecture and Software Configuration Management," in *Software Configuration Management ICSE Workshops SCM 2001 and SCM 2003*, Portland, 2003.
- [6] Technical Committee : ISO/IEC JTC 1/SC 7, "Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models," 2011.
- [7] J. D. McGregor and D. A. Sykes, *A Practical Guide to Testing Object-Oriented Software*, Addison-Wesley Professional, 2001.
- [8] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2004.
- [9] B. Beizer, *Software Testing Techniques*, Itp - Media, 1990.
- [10] R. Venables, D. Spinellis and J. Fuller, *Code Reading: The Open Source Perspective*, Addison-Wesley Professional, 2003.
- [11] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering: Conquering Complex and Changing Systems*, Pearson College Div, 1999.
- [12] R. v. Ommering, "Configuration Management in Component Based Product Populations," in *Software Configuration Management, ICSE Workshops SCM 2001 and SCM 2003*, 2001.
- [13] J. Richardson and W. A. Gwaltney, *Ship it! A Practical Guide to Successful Software Projects*, Pragmatic Bookshelf, 2005.

