Addressing Psychosocial and Lifestyle Risk Factors to Promote Primary Cancer Prevention: an integrated platform to promote behavioural change (IBeCHANGE)

Project Number: 101136840

D1.3 – Technical Assessment

Related Work Package	**WP1 – Project Management and coordination**
Related Task	**Task 1.3 - Technical and innovation management**
Lead Beneficiary	EUT
Contributing Beneficiaries	IEO, iBeChange consortium
Document version	v1.0
Deliverable type	R
Dissemination level	PU
Due date	31/07/2025
Delivery date	31/07/2025
Authors	Silvia Orte, Marta Picazo, Laura Sistach
Contributors	David Suñol, Giorgia Miale (iBeChange PM, IEO)
Reviewers	iBeChange Consortium

Funded by the European Union

This project has received funding from the European Union's Horizon Europe research and innovation programme under the Grant Agreement Number 101136840.

Disclaimer

Table of Contents

List of Abbreviations

Abbreviation	Explanation
DAG	Directed Acyclic Graph
EUT	Fundació Eurecat
ICO	Institut Català d’Oncologia
IEO	Istituto Europeo di Oncologia
PoC	Point of Care
POLIMI	Politecnico di Milano
RL	Reinforcement Learning
SIMAVI	Software Imagination & Vision
TU/e	Technical University of Eindhoven
UX	User Experience
VU	Virtual User
WP	Work Package

List of Tables

List of Figures

Executive Summary

This deliverable (D1.3) presents the first Technical Assessment of the iBeChange project, developed under Work Package 1: Project Management. It outlines how technical coordination has been carried out during the initial phase to ensure that progress stays on track, milestones are reached, and work is aligned with project objectives and quality standards.

Led by EUT under Task 1.3: Technical and Innovation Management, this assessment describes the governance structure, coordination mechanisms, and monitoring tools that support the execution of technical activities. Bi-weekly meetings, milestone reviews, and continuous tracking through Jira software helped ensure consistent progress and resolve blockers across tasks.

The project utilizes shared development tools (GitLab, Docker, MongoDB, Airflow) and collaboration platforms (Microsoft Teams, OneNote, Google Drive) to ensure transparency and alignment between partners. Testing and code review procedures are in place to support quality assurance and modular development.

Overall, the technical activities have progressed as planned, with key milestones on track and risks proactively identified. This assessment will continue to serve as a reference to support coordination and ensure the quality and coherence of future developments in iBeChange.

1. Introduction

This deliverable presents the first Technical Assessment of the project, developed under Work Package 1 (WP1) – Project Management. WP1 aims to ensure effective coordination, monitoring, and support across all project activities, ensuring that the work progresses according to plan, milestones are met, and results meet high standards of quality.

This report was developed in accordance with Task 1.3: Technical and Innovation Management, that supports the coordination of all technical tasks within iBeChange to ensure:

- adherence to the work plan,
- technical and innovation quality of methodologies and results,
- achievement of technical milestones,
- effective coordination among interlinked technical tasks, and
- promotion of collaboration with technical stakeholders while maintaining alignment with end-user needs and the state-of-the-art.

This deliverable also supports the activities of Task 1.1 (Project coordination) by providing a structured overview of the technical progress, risk identification, and mitigation strategies, and will contribute to WP7 (Ethical, Privacy, and Data Protection) by ensuring that technical developments are aligned with ethical and data protection requirements.

This report also includes an overview of the internal mechanisms adopted to manage and coordinate technical innovation across partners; outlines lessons learned and provides an outlook on upcoming activities and challenges.

This document provides a detailed overview of the technical and innovation management processes implemented during the first project period and up to date (M1-M19), aligning with WP1’s overarching goal of ensuring the smooth, high-quality, and ethically aligned execution of the project’s scientific and technological roadmap.

2. Technical management strategy

Technical management within the iBeChange project is designed to ensure alignment across technical work packages, adherence to the project work plan and the delivery of high-quality outputs. This strategy is implemented under the leadership of Eurecat, as the task leader for Task 1.3, in close collaboration with the Project Coordinator (IEO) and the Clinical Manager (ICO) of the iBeChange Consortium.

2.1. Technical Governance Structure

Role of the Innovation/Technical Manager

Laura Sistach (EUT), the Innovation/Technical Manager plays a key role in coordinating day-to-day technical oversight. The Technical Manager is responsible for implementing and maintaining monitoring procedures, ensuring the smooth progression of technical tasks, and supporting inter-partner coordination. This includes the facilitation of regular technical meetings, tasks tracking, milestones reviews, and documentation of technical progress. Carolina Migliorelli Falcone was originally indicated as Innovation/Technical Manager; however, she is no longer in this role. These responsibilities have since been taken over by Laura Sistach since October 2024.

Technical partners

The technical development of the project is carried out collaboratively by a group of expert partners, each contributing specialised knowledge and responsibilities aligned with their domain of expertise. Below is a summary of the key roles and contributions of each technical partner:

- EUT: Responsible for the development and maintenance of both the app backend and the PoC backend and frontend. It also guides the frontend development of the mobile app. EUT also leads the technical integration of all partner components, manages the integration with the Oura API (including scheduled data retrieval) and oversees database design and management. Additionally, EUT is in charge of building the VU model, which integrates and processes multi-source data to support personalized intervention.
- POLIMI: Different research groups at POLIMI oversee:
    - Reinforcement Learning (RL): Development of the reinforcement learning model that selects optimal behavioural recommendations tailored to individual users. These recommendations are based on user characteristics and historical interaction data.
    - Wearables Data: Identification and selection of wearable devices capable of passive, non-intrusive monitoring of behavioural determinants. Tasks include device testing, data acquisition, and sensor data processing.
    - Voice Recordings and Emotional Monitoring: Development of methods for automatic voice analysis to infer emotional valence and arousal. This involves on-the-fly extraction of acoustic markers from voice samples to monitor stress levels and emotional states.
- SIMAVI: Responsible for the implementation of the mobile application. They translate the user journey and functional specifications into an interactive digital product. Activities included: Exploration and validation of different interface designs and frontend development of the mobile app based on technical inputs and user-centred requirements.
- TU/e: Leads the development of an RL-based framework to determine the best timing for the delivery of recommendations, tailored to user traits and contexts. Users' characteristics, contextual features provided by the wearables (for the sub-population enrolled on the wearable study) and the users' interaction with the platform are taken into account to learn opportune

moments for notification delivery for the users. Furthermore, they worked on designing methods to monitor and enhance user engagement over short-, mid-, and long-term timeframes.

2.2. Monitoring Procedures

A structured set of monitoring procedures has been established to ensure effective technical management and transparency across all partners. These procedures are coordinated by the Technical Coordinator.

Bi-weekly Technical Meetings

Bi-weekly technical meetings serve as the core mechanism for coordination and tracking of ongoing work. During each meeting, all technical partners provide updates on:

- What has been done
- What is currently being done
- What will be done next
- Blocking issues and doubts

These meetings also offer a dedicated space to discuss technical topics, raise blockers, clarify implementation doubts, and align priorities across work packages. All meetings are minuted, and action points are recorded and assigned to relevant partners to ensure follow-up and accountability.

Milestone Reviews

At key stages of the project, milestone reviews will be conducted to evaluate progress in relation to the project timeline. These reviews include:

- Cross-checking milestones and deliverables with the Description of Action (DoA)
- Ensuring consistency with technical and scientific objectives
- Aligning task outputs with milestone expectations defined in each WP. Table 1 shows the different technical milestones.

Table 1. List of technical milestones and monitoring procedures.

Milestone Number	Milestone Name	Related WPs	Due Date	Means of Verification	Monitoring Procedures
M3	Description of the iBeChange integrated platform and PoC	2	M21	Report	Reviewed in bi-weekly clinical and technical meetings; tracked via Jira; progress reported to the Consortium
M4	Selection of wearable sensors	3	M6 (concluded)	Report	Bi-weekly technical meetings; alignment verification

M5	Deployment of interfaces for data collection, visualisation, and user engagement	3	M24	Software release	Full-flow testing; demo session; status tracked in Jira; reviewed during sprint reviews and bi-weekly technical meetings
M6	Deployment of the iBeChange platform	4	M24	Software release	GitLab status; platform demo; technical milestone review with partners during bi-weekly technical meetings; issues monitored via Jira

Interaction Between Technical Tasks

To facilitate coordination and avoid silos, clear communication channels and reporting structures have been established between technical work packages. Task leaders meet regularly to align efforts, clarify interfaces, and manage dependencies between software development, data integration, behaviour change logic, and user experience design.

A simplified diagram of input-output flows between key technical tasks is included in Figure 1.

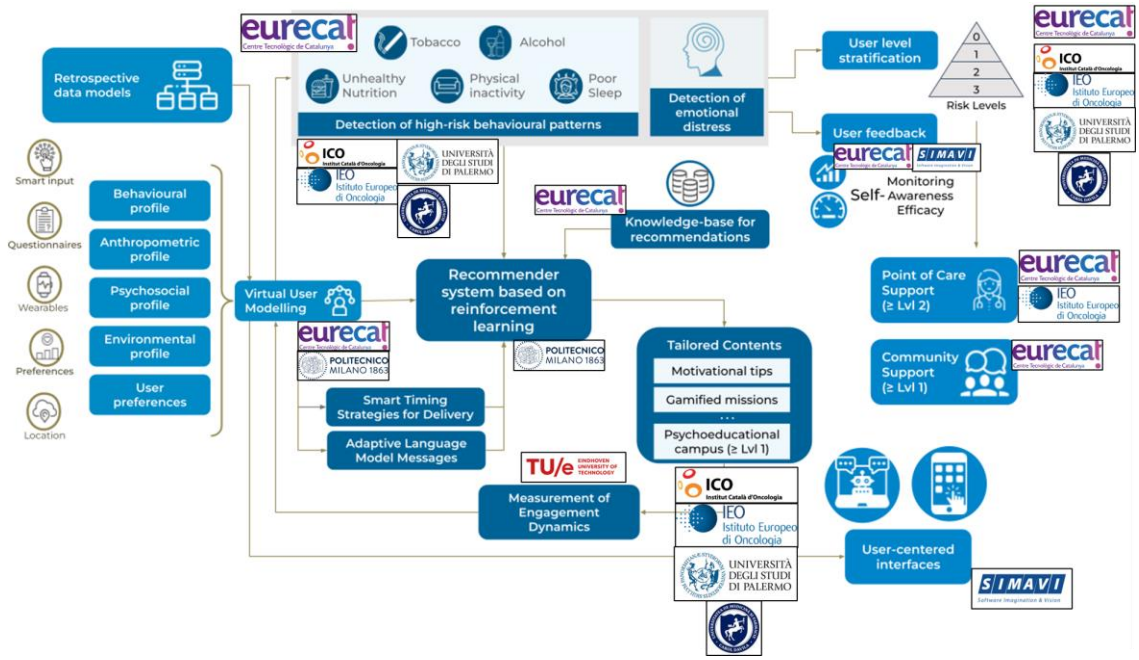

Figure 1. Diagram of the flow between technical tasks.

Examples of effective coordination between partners (highlighted cases)

Throughout the first phase of the project, several instances of successful technical coordination have taken place. For example, the collaboration between the data science (e.g. EUT, POLIMI) and clinical teams (e.g. ICO, IEO) enabled the early definition of input data requirements for the virtual user model, also ensuring compatibility with ethical and regulatory constraints provided by WP7

partners (i~HD). Similarly, joint sessions between the UX responsible partners (SIMAVI) and recommender system teams (EUT, POLIMI) helped translate user feedback into technical design improvements, strengthening user-centred innovation across the platform.

3. Evaluation Procedures

To ensure that the technical work carried out in the project is reliable and consistent, the Technical Coordinator applies a structured evaluation process. This includes regular peer reviews of the code, systematic testing at different levels, and continuous monitoring of progress and functionality. These procedures are part of EUT's day-to-day development work and help us detect problems early, improve collaboration, and make sure that every part of the system works as expected.

3.1. Code Reviews

All source code developed by the Technical Coordinator is hosted in EUT’s GitLab [1] repository, where the development process is structured through branches, commits, and merge requests. This environment supports version control and collaborative development across team members.

Each code contribution is submitted via a merge request and must go through a peer review before it can be merged into the main branch. Reviews are carried out by at least one other developer, who checks the functionality, structure, and overall quality of the code. The aim is to ensure consistency with the project’s architecture, detect potential bugs early, and confirm that the implementation meets the requirements.

Code reviews also include verification of test coverage and documentation. Reviewers assess whether the new code is properly tested and whether any updates to technical documentation (such as README files or API specifications) are needed. If any changes are required, the author must address them before the merge request is approved.

3.2. Unit, Integration and System Testing

Testing is a fundamental part of our development workflow and is structured into different layers to ensure full coverage of the system’s logic. All tests are written in Python [2] using *pytest*.

Fixtures and test data are managed explicitly within the test structure. The *database_fixtures.py* module allows us to preload mock data and simulate realistic database states, making tests repeatable and consistent.

At the **unit** level, individual modules and functions are tested in isolation. These tests focus on verifying the internal logic of components such as processing routines, update functions, and database interactions. For example, we test the logic behind Health Pillars processing pipelines, as well as the prescription engine and update logic. These tests ensure that each building block of the system performs correctly under a variety of input conditions.

The **functional** test layer verifies that complete features work as expected. These tests interact with the system’s APIs to validate expected behaviour across modules. Functional tests are organized by features, such as actions, missions, or objectives, helping ensure that changes in one module do not unintentionally affect another.

At the **integration** level, we simulate real workflows of the users on the App and interactions between modules using full-flow tests. For instance, the file *test_full_flow.py* reproduces end-to-end use cases, such as the processing of user data, recommendation generation, and delivery through the backend. This level of testing ensures that all components integrate smoothly and that the system operates as a whole.

This layered testing approach ensures that errors are caught early, logic remains consistent across updates, and deployments are safe.

4. Documentation

4.1. Postman for API Documentation

Postman [3] was extensively used to document and test the backend APIs developed for both the mobile app (mApp) and PoC. It served as a collaborative tool for frontend and backend developers to validate request/response flows and ensure consistent API behaviour across services.

The workspace for the mobile app, named **ibechange-app**, contains a structured set of collections corresponding to the main functional modules of the system. Each collection represents a domain-specific group of endpoints used by the apps to interact with the backend, including:

- auth: Endpoints related to authentication and authorization (e.g. login, token management).
- user: User account creation and profile management.
- content-manager: APIs for managing dynamic content shown in the mApp, including multimedia and text blocks.
- chat: Chat-related endpoints for communication features within the mApp.
- process-manager: Workflow management endpoints, likely handling task or interaction flows within the mApp (for example, sending questionnaire answers or sending feedback of a recommendation).
- push: Endpoints for registering and un-registering push tokens.
- Pillars, Missions, Actions, Objectives: These collections reflect the core structure of the behavioural change model implemented in the mApp, enabling personalized guidance and monitoring.
- Recommendations: Endpoints serving personalized recommendations to users.
- Health status: APIs for tracking and updating the user's health-related scores, based on their questionnaire answers.
- Questionnaire image: Endpoints that support the images of questionnaires in the mApp.

Furthermore, there is a dedicated workspace for the Web app backend (PoC), named **ibechange-backend**. This workspace documents the backend infrastructure for the PoC and admin interface. Collections are versioned (v1.0) and logically grouped:

- auth: Authentication services for web platform access.
- user: Admin user management and permissions.
- content-manager: Admin tools to download contents.
- process-manager: Backend process orchestration, e.g., retrieving participant tasks, retrieving task logs or retrieving tasks.
- questionnaire: Management of digital questionnaires.
- alert: Includes endpoint to retrieve user alerts.
- version: Versioning control and compatibility endpoints.

Apart from the mApp and PoC backends, there is also a dedicated workspace for the Python module, which regulates the logics of the mApp and PoC. It includes collections for managing health pillars, objectives, missions, actions, etc., and point of care visualizations.

Subsequently, a new collection was created specifically to support and coordinate the integration efforts of all technical partners involved in the project. More concretely, to manage the integration between the RL, the engagement dynamics and the wearables modules.

Reinforcement Learning (RL) Module: This module requires specific inputs related to the detection of "high-risk behaviour patterns," lifestyle factors associated with cancer risk, a concept internally referred to as "health pillars." To function effectively, the RL module must be able to

retrieve user-specific information, access the user's defined health pillars, and both select and transmit tailored recommendations back to the user. These interactions rely on dedicated endpoints that enable the retrieval of relevant data and the delivery of personalized outputs.

Wearables Module: The wearables module is designed to interact with the VU application to obtain raw data from wearable devices, as well as voice-related data, linked to a specific user. This data retrieval is on-demand and occurs whenever the system identifies the need for additional contextual or physiological information. The raw wearable data can be accessed through a designated endpoint, which ensures secure and timely delivery of the necessary data streams for further processing and analysis.

Engagement Dynamics Module: This module focuses on capturing and interpreting user engagement patterns and feedback. It has the capability to request detailed information regarding a user's interaction dynamics with the system. This is facilitated through a dedicated endpoint that provides access to metrics and qualitative data reflecting how users respond to and engage with the digital intervention over time.

In every collection, each endpoint is accompanied by:

- Request examples with required headers, parameters, and body payloads
- Expected response formats (success and error cases)
- Status codes and basic validation scenarios

The Postman collections are version-controlled and aligned with the API schemas defined in Swagger documentation (section 4.2). They have been shared with development partners and testers to support rapid debugging, testing, and validation across environments.

4.2. Swagger for API Schemas and Data Models

Swagger (OpenAPI Specification) [4] has been used to document and define the backend APIs for both the mobile app (mApp) and the web-based PoC. It provides a structured, static description of all available endpoints, including their methods (GET, POST, etc.), required parameters, headers, and response formats.

Unlike Postman, Swagger focuses on descriptive and declarative API documentation, providing a comprehensive overview without including runtime values or sample executions. Here, we could clearly define input and output parameters, document requests and response structures, and ensure consistency across frontend and backend implementations.

Furthermore, with Swagger we could define data models (schemas). These models describe the structure of the objects exchanged via the APIs (such as User, Questionnaire, Action, or Recommendation), and include field types, required properties, and nested relationships.

All endpoints are grouped by module (e.g., auth, user, questionnaire, etc.) and follow versioning best practices (e.g., v1.0) to ensure traceability and backward compatibility as the system evolves.

Figure 2. Example of Swagger API Schema: User Model.

4.3. GitLab for documentation about deploying and running the project

Apart from using GitLab as a repository for code, it has been actively used to document all processes related to deployment, execution, and maintenance of the system. Thus, in the project, GitLab serves as the central repository and DevOps platform for both the mobile and web components of the project.

More concretely, the GitLab repositories of the project include:

- Deployment Instructions: Step-by-step guides to set up and run the backend services and databases. This includes environment variables, service configuration files, and dependencies required for local development and production deployment.
- CI/CD Pipelines: Configuration files (e.g., example.yml) define automated build, test, and deployment workflows.
- README and Wiki Pages: Detailed technical documentation is available via repository README files and GitLab’s integrated wiki. These pages cover aspects such as initial project setup, running backend/frontend locally, API base URLs and authentication mechanisms, and system requirements and dependencies
- Issue Tracking & Merge Requests: GitLab’s issue and merge request system provides traceability of development tasks, bug fixes, and feature integration, including links to related documentation or deployment changes.

4.4. Microsoft Word for functional and technical documentation

Microsoft Word [5] was used to create and maintain both functional and technical documentation throughout the project. This includes:

- Functional documentation: High-level descriptions of app features, expected behaviour, and user flows.
- Technical documentation: Detailed explanations of application screens, including the backend API endpoints associated with each screen and data interactions.
- Project deliverables: Several official project deliverables and internal reports were prepared using Word, ensuring consistency in formatting and collaborative editing across consortium members.

4.5. Miro for functional documentation

Miro [6] was employed as a collaborative whiteboarding tool, primarily during the co-creation sessions with clinicians and stakeholders. It played a key role in:

- Designing the User Journey: Mapping the end-to-end experience for different user types, especially individuals with intellectual disabilities and their clinical teams.
- Early functional planning: Visualizing features, interaction flows, and feedback loops before wireframes and development.
- Collaborative ideation: Facilitating asynchronous and real-time discussions across partners during the design and validation phases.

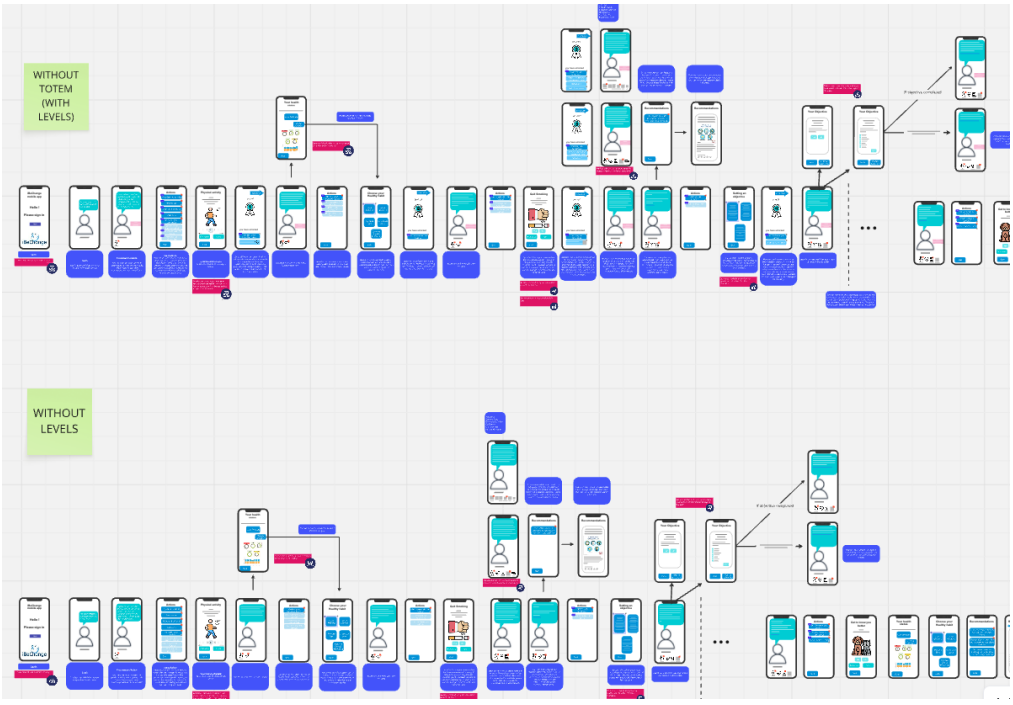

Figure 3. Section of Co-Creation Miro.

4.5. OneNote

To document meetings and follow-up actions across the project, Microsoft OneNote [7] was used as a digital notebook. This tool allowed us to centralize notes from different types of meetings.

The notebook is organized by sections and pages. Each meeting entry includes the date, title (e.g. “8th Technical Meeting”), and detailed notes regarding updates, open points, partner feedback, and agreed next steps.

Sections are used to distinguish between internal meetings, consortium-level discussions, co-creation sessions, and advisory board activities.

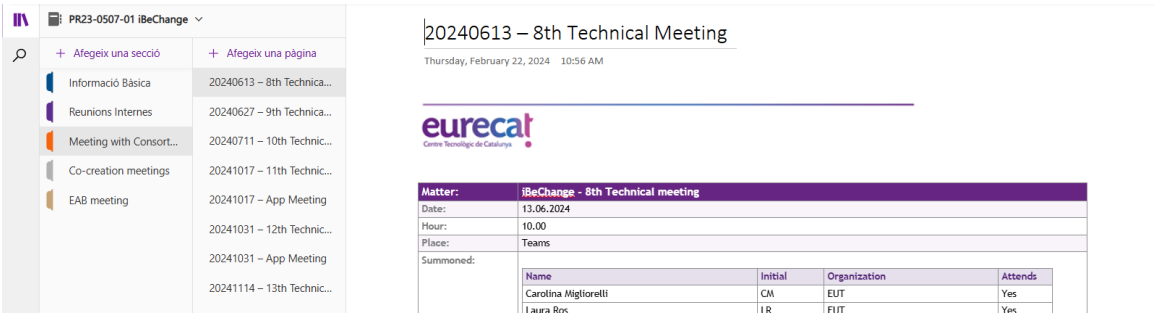

Figure 4. Microsoft OneNote structure.

Using OneNote ensures that technical partners can access the meeting documentation, track discussions, and follow up on action items transparently. It complements other coordination tools such as Microsoft Teams and Google Drive and helps maintain a clear memory of the project.

5. Tools and Platforms

To support collaboration, development, testing, and deployment throughout the project, the technical partners have used a combination of tools and platforms. These tools help the team stay organized, maintain code quality, and ensure repeatable deployments.

5.1. Google Drive

Google Drive [8] is used to store and collaborate on project documents, including reports, deliverables, planning sheets, and meeting notes. IEO maintain a shared Google Drive repository titled "iBeChange CONSORTIUM SPACE". This centralized workspace is structured into clearly named folders corresponding to the main areas of project coordination and documentation, including:

- Work packages: for technical documents and implementation material for each work package.
- Deliverables, Milestones, and Interim Report 2025: for tracking and submitting formal outputs.
- Minutes and continuous reporting: for monitoring project progress and meetings.
- Dissemination and communication, Cluster activities and iBeChange training and co-creation: for knowledge transfer and stakeholder engagement.
- External advisory board: for governance and external input.
- Templates, documents, and contacts: for internal coordination and resource sharing.

All consortium partners have access to this space, which is kept up to date with structured naming and access rights.

5.2. Microsoft Teams

Microsoft Teams [9] is EUT's main platform for daily communication, coordination, and internal project management across the technical team. It plays a central role in keeping partners connected, informed, and aligned throughout the project. We use Teams mainly to host video meetings such as weekly team check-ins, bi-weekly technical reviews, and coordination meetings.

Teams acts as a central hub for communication, helping reduce email overload and keeping discussions structured by topic. Thanks to built-in integration with Microsoft Outlook [10], calendars, and file storage, it also simplifies scheduling and collaboration on shared documents.

5.3. Slack

Slack [11] is used as a quick communication tool, especially during active development or testing phases. We use separate channels for specific components or technical issues.

5.4. JIRA

The technical team uses Jira as the main tool for planning, tracking, and reviewing all development tasks. It helps the team stay aligned, manage workloads, and deliver on time across all technical phases. We organize our technical work into 2-week sprints, which allows us to deliver incremental updates. Each sprint includes development, testing, documentation, and review tasks. All project team members use Jira on a daily basis to update progress, provide comments, and keep track of what’s completed, ongoing, or blocked.

Jira helps us break down the backlog into defined tasks, such as "Test mission prescription" or "Implement notification system", allowing for clear visibility of what needs to be done. Tasks are

grouped by sprint, priority, and component, making it easy to organize the workload and align it with the project's timeline. Responsibilities are assigned to team members, and the estimated effort is indicated using story points, facilitating capacity planning.

Progress is monitored through sprint boards, which help the team stay up to date and identify any delays early.

The screenshot below shows an example of Sprint including tasks such as push notification implementation, APK feedback, and deliverable reviews. Each task is labeled, assigned, and has a status that’s updated in real time.

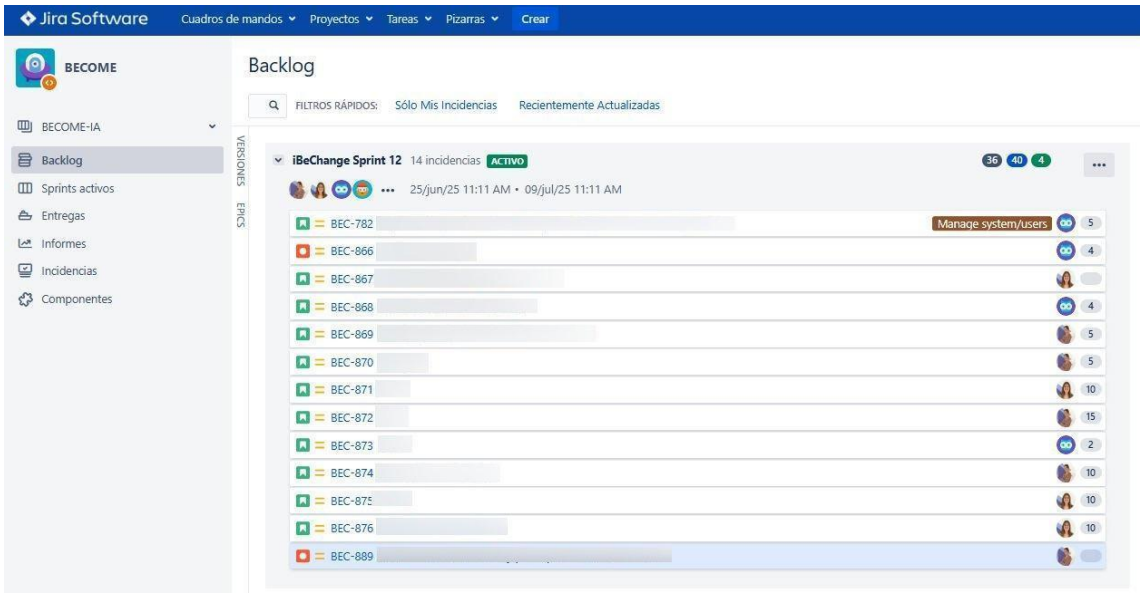

Figure 5. Example of iBeChange Sprint 12 in Jira.

Jira is also integrated with GitLab, so any updates or changes are instantly shared across platforms, keeping the entire team informed.

5.5. GitLab

GitLab is EUT's main platform used to manage source code and coordinate development of the backend that powers the iBeChange mobile application. The VU backend is developed in Python and organized into a modular architecture, with clear separation between APIs, business logic, database models, and data processing components.

The project repository includes essential development tools such as a Dockerfile for containerization, Makefile for automation, testing and deployment scripts, Postman collections, and well-structured documentation (README, requirements). The codebase is version-controlled using Git, and all contributions are integrated through merge requests that include peer reviews, comments, and test execution.

The main directory structure separates core functionalities into subfolders like APIs, logic, database, and processing, making the codebase easy to navigate and maintain. Each module is regularly updated, and commits are traceable, following good development practices, while supporting collaboration between developers and partners working on different parts of the system.

5.6. Docker

Docker [12] allows us to ensure that all applications are developed, tested, and deployed in consistent environments. All project services are containerized using Docker, meaning each one includes all required dependencies and configurations. This guarantees that the behaviour of each component is identical across local development setups, testing environments, and production.

During development, we rely on Docker Compose to quickly launch full environments with multiple services, such as APIs and databases, which simplifies integration across teams and reduces setup time.

Docker is integrated with both GitLab and Airflow. When code is merged into the main branch of GitLab and a new Docker image is built, a new version of the Docker image is uploaded to the internal container registry. This process is then picked up by Airflow, which orchestrates the deployment of the updated image to the corresponding environment. This combined system enables us to maintain up-to-date services in a reliable and fully automated way, reducing manual errors and shortening the deployment cycle.

5.7. Airflow

Apache Airflow [13] is used to automate and coordinate key processes across different technical modules in the project. It allows us to manage the execution of independent but interrelated modules, such as the backend of the mobile application, the Reinforcement Learning system that selects and schedules personalized recommendations, and retrieving data from wearables and audio managed by other partners.

Each process is defined as a DAG (Directed Acyclic Graph) using Python and can be scheduled or triggered automatically. Airflow enables us to schedule daily jobs (e.g. data retrieval), trigger real-time updates, or link pipeline executions across components. This keeps everything connected and running without manual steps.

Airflow works together with GitLab and Docker. When the code is updated and a new Docker image is created, Airflow can detect the change and run the necessary tasks, like redeploying services or updating models. This way, deployments are smooth and fully automated.

The Airflow interface provides information on the status of each DAG, with indicators of success or failure, execution logs, and scheduled runs.

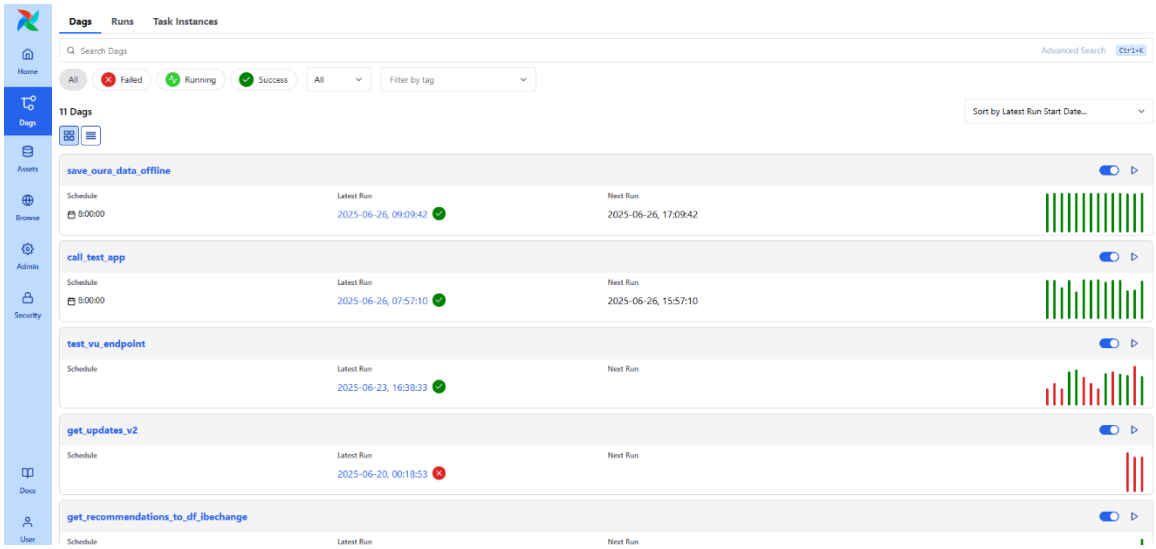

Figure 7. Example of active DAGs managed through Airflow for the iBeChange system.

5.8. MongoDB

MongoDB [14] is used as the main database to store all user-related information necessary for the operation of the iBeChange system. The structure is fully anonymized, so no personal identifiers are stored, and the data is organized in collections that reflect the app’s core components, such as behaviour change, recommendations, intervention levels, and health scores.

The database (iBeChangeDB) is structured into collections. Each collection stores documents in a flexible format, which allows us to easily adapt to evolving data needs without requiring strict schemas.

This database acts as a central hub that connects the different modules of the system. The web backend uses it to fetch and update user progress and deliver appropriate content to the Point of Care, while external systems, such as the RL system or the wearables module, receive relevant information through secured API endpoints or periodic synchronization processes.

MongoDB’s flexible document model is key to maintaining a scalable system, where each technical partner can work independently but still access consistent and up-to-date data needed for their logic or analytics.

5.9. Keybase

Keybase [15] is used to securely share sensitive files such as API keys, configuration files, or credentials. It uses end-to-end encryption and identity verification to ensure that only authorized team members can access this content.

6. Conclusions

This Technical Assessment gives an overview of how the technical activities in iBechange have been coordinated and monitored during the initial phase, in line with the goals of Task 1.3 Technical and Innovation Management. This task, led by Eurecat, focuses on ensuring that all technical work follows the project’s work plan, meets high quality standards, achieves its milestones, and stays aligned with both end-user needs and the state of the art.

We have put in place a solid structure to ensure technical quality, including regular code reviews, tests, and continuous integration. Shared tools like GitLab, Jira, Docker, MongoDB, and Airflow have allowed teams to work efficiently, track progress, and stay aligned. Communication tools such as Microsoft Teams, OneNote and Google Drive have supported internal coordination and documentation.

Thanks to these procedures, the technical work has progressed according to plan, and key milestones have been achieved. The system is being developed in a modular way, allowing different teams to contribute while keeping a shared direction. We have also started identifying potential risks and applying mitigation strategies.

As the project continues, this plan will remain a reference to guide coordination and quality control. It will be regularly reviewed and adapted based on experience and future needs. The goal is to keep improving collaboration and ensure that all developments contribute effectively to the success of iBeChange.

This deliverable also contributes to WP7 by ensuring that the technical development is consistent with ethical and data protection standards.

7. References

[1] *GitLab*. [Online]. Available: https://about.gitlab.com/
[2] *Python*. Python. [Online]. Available: https://www.python.org/
[3] *Postman*. [Online]. Available: https://www.postman.com/
[4] *Swagger*. [Online]. Available: https://swagger.io/
[5] *Microsoft Word*. [Online]. Available: https://www.microsoft.com/es-es/microsoft-365/word
[6] *Miro*. [Online]. Available: https://miro.com/es/
[7] *Microsoft OneNote*. [Online]. Available: https://www.microsoft.com/es-es/microsoft-365/onenote/digital-note-taking-app
[8] *Google Drive*. [Online]. Available: https://workspace.google.com/intl/es/products/drive/s
[9] *Microsoft Teams*. [Online]. Available: https://www.microsoft.com/es-es/microsoft-teams/group-chat-software
[10] *Microsoft Outlook*. [Online]. Available: https://www.microsoft.com/es-es/microsoft-365/outlook/email-and-calendar-software-microsoft-outlook
[11] *Slack*. [Online]. Available: https://slack.com/intl/es-la
[12] *Docker*. [Online]. Available: https://www.docker.com/
[13] *Apache Airflow*. [Online]. Available: https://airflow.apache.org/
[14] *MongoDB*. [Online]. Available: https://www.mongodb.com/
[15] *Keybase*. [Online]. Available: https://keybase.io/

Version history

Version	Description	Date completed
v1.0	First draft	30\6\2025
v1.1	IEO revision	7\7\2025
v1.2	CONSORTIUM revision	25\7\2025