

REPORT



# Integration strategy and planning

- project deliverable 6.1

Authors: Francesco D'Andria, Carlos Alberto Martin and Jose Miguel Garrido, Ville Ollikainen and Jarkko Kuusijärvi, Tommi Meskanen

Confidentiality: *Public*



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825585.

<b>HELIOS Integration strategy and planning</b>	
<b>Project name</b> HELIOS	<b>Grant agreement #</b> 825585
<b>Authors</b> Francesco D'Andria, Carlos Alberto Martin, Jose Miguel Garrido (ATOS); Ville Ollikainen and Jarkko Kuusijärvi (VTT); Tommi Meskanen (UH).	<b>Pages</b> 2+28
<b>Reviewers</b> Kristina Kapanova (TCD), Robin Ribback (STXT), Chryssanthi Iakovidou (CERTH)	
<b>Keywords</b> Integration strategy and planning; DevOps, Continuous Integration, Software Release	<b>Deliverable identification</b> D6.1
<p><b>Summary</b></p> <p>This deliverable presents the first and preliminary version of the HELIOS project integration plan.</p> <p>The HELIOS consortium will provide new and updated versions of the HELIOS integration activities in the further documents belonging to the WP6.</p> <p>The document describes the first and preliminary integration approach that will be followed during the project as well as the code version control repository, the staging infrastructure and the tools for supporting the continuous development, integration and operation of the HELIOS Social Network. In the first project phase the consortium will rely on a private GitLab repository managed by ATOS: <a href="https://gitlab.atosresearch.eu/ari/HELIOS">https://gitlab.atosresearch.eu/ari/HELIOS</a>. At M18 the consortium plans to move the HELIOS code to a public GitHub code repository.</p> <p>Finally, a short description of the main components belonging to the HELIOS architecture together with the preliminary calendar for their integration is also described in the document.</p>	
<b>Confidentiality</b>	Public
29.6.2019 Written by  Francesco D'Andria	
<b>Project Coordinator's contact address</b> Ville Ollikainen, <a href="mailto:ville.ollikainen@vtt.fi">ville.ollikainen@vtt.fi</a> , +358 400 841116	
<b>Distribution</b> HELIOS project partners, subcontractors, the Project Officer and HELIOS web site	



## Contents

---

Contents .....	1
List of Contributors.....	2
List of Figures .....	2
List of Tables .....	2
List of Acronyms .....	3
1 Introduction.....	4
2 Development and integration approach .....	6
2.1 Agile methodology.....	7
2.2 Best practices.....	8
3 HELIOS architecture components .....	11
4 HELIOS support tools.....	13
4.1 Project management tools.....	13
4.2 Distributed version control system.....	13
4.3 Real-time collaborative tools .....	14
5 Integration process.....	17
5.1 Testing process: Integration testing & Validation testing.....	17
5.2 Support tools for Automated Testing .....	20
5.3 Testing Infrastructure approach.....	21
5.4 Operation Infrastructure.....	22
6 First HELIOS software integration and delivery plan.....	23
7 Conclusion .....	26
8 Reference.....	27



## List of Contributors

---

Partner short name	Main contributions
<b>ATOS</b>	Section 1 “Introduction”, Section 5 “Integration process”, Section 6 “First HELIOS software integration and delivery plan” and Section 7 “Conclusion” Sub-section 4.2 “Distributed version control system” and Sub-section 4.3 “Real-time collaborative tools”
<b>VTT</b>	Section 3 “HELIOS architecture components” Sub-Section 4.1 “Project management tools”
<b>UH</b>	Section 2 “Development and integration approach”

## List of Figures

---

Figure 1. HELIOS logical architecture components.....	11
Figure 2.HELIOS Real-time collaborative room in Slack.....	16
Figure 3. V model for software testing (Image taken from [16])[2].....	17
Figure 4. First HELIOS Integration roadmap.....	25

## List of Tables

---

Table 1. Real-time collaborative tools investigated in the context of HELIOS .....	15
--	----



## List of Acronyms

---

Acronym	Description
AR	Augmented Reality
CD	Continuous Delivery
CITDM	Continuous Integration, Testing and Deployment Methodology
CI	Continuous Integration
CVS	Concurrent Versions System
DevOps	DEvelopment OPerationS
MR	Mixed Reality
P2P	Peer-to-Peer
QoS	Quality of Service
SVN	Subversion
T2.1	HELIOS activity Task 2.1 “Design Fiction”
VR	Virtual Reality
XR	Extended Reality
WP2	HELIOS activity Work-Package 2 “Concept Design”
WP3	HELIOS activity Work-Package 3 “System development”
WP4	HELIOS activity Work-Package 4 “Social Networks Layers and Analysis”
WP5	HELIOS activity Work-Package 5 “Services and user interfaces”
WP6	HELIOS activity Work-Package 6 “System integration and operation”



## 1 Introduction

---

The H2020 HELIOS project main goal is to research, validate and introduce to the market a new way for online interaction that mimic real-life organic networking. HELIOS social network will be based on ad-hoc or pre-defined networking, cognitive interactions and peer-to-peer (P2P) collaboration between people, smart objects and their Virtual Reality (VR), Augmented Reality (AR), Mixed Reality (MR) cyber representations [1].

HELIOS aims to design, implement and deliver a decentralized social media platform that will address the dynamic nature of human-to-human and human-to-machine (so involving smart objects) communications in three dimensions: contextual, spatial and temporal.

This platform further aims to provide an extension for the mobile operating systems (focussing on Android for the duration of the project), providing easy-to-apply P2P social media functionality for 3<sup>rd</sup> party developers.

In HELIOS architectural components will be built in a modular and extensible manner, by enabling all features present in social media platforms today and building beyond the current state of the art. For human networking, HELIOS will introduce novel concepts for social graph creation and management, which are grounded on trust and transparency.

These concepts will be validated in variety of business cases together with novel social media features, such as innovative feedback technology and shared spaces.

Furthermore, HELIOS will be modular, extensible and built upon open source, ensuring that developers and/or start-ups can easily create novel social media apps on top of HELIOS in the future, beyond the end of the project.

The purpose of this public HELIOS project deliverable (named D6.1 “Integration strategy and planning”) is to guarantee that the Continuous Development, Integration and Operation (DevOps) of the HELIOS Social Network platform happens in a coordinated way.

The HELIOS DevOps approach is based on some key principles, listed below:

- Coordination of the software development, realized through the technical work-packages (WP3, WP4 and WP5) to ensure a gradual, unified, continuous and rigorous software integration. The main goal is to avoid the typical integration pitfalls [36] and to ensure that approaches inconsistencies and software incompatibilities are discovered and fixed in a timely manner.
- The coordination described above will be facilitated through tools such as:
  - a) the plan defined in this document and,



b) the adoption of a continuous integration platform and tools as described in sections 4 and 5 of this deliverable.

- The goal is the development of part of the system as an open source project since the very beginning. The purpose is to ensure that the project gradually acquires visibility and support by the community.

To ensure the consistency of the approach, the deliverable herein, defines the methodology and the related support tools which will be utilized throughout the development and integration of the HELIOS Social Platform, as well as the deadlines and the preliminary roadmap.

Deliverable D6.1 is structured as follows:

- Section 1 provides an introduction to the deliverable and the required acronyms.
- Section 2 describes the approach followed for the development of the HELIOS Social Network software platform.
- Section 3 reports on the main components of the HELIOS architecture described in the WP3 Deliverable D3.1 “Draft system architecture and basic API specification”.
- Section 4 introduces the HELIOS (software management) support tools.
- Section 5 presents the integration and the testing processes, as well as the infrastructure.
- Section 6 introduces the preliminary integration roadmap with the milestones and a calendar for the integration of the different components and release schedule of the three main versions of the software.
- Section 7 concludes the deliverable.



## 2 Development and integration approach

---

In the context of the WP6, the HELIOS consortium aims to:

- Establish an efficient and reliable platform development, integration and delivery strategy to meet the required quality indicators for the HELIOS ecosystem,
- Integrate all the technical components required for successfully develop, integrate and operate the HELIOS platform in a relevant business context (baseline plus components implemented by the technical WPs: WP3, WP4 and WP5),
- Verify and assess the reliability, performance, vertical and horizontal scalability and security compliance of the HELIOS platform. This makes the platform ready for game and stories development,
- Deploy and Operate the HELIOS platform under a required Quality of Service (QoS) strategy.
- Provide feedback to the project management (WP1) as well as to technical WPs based on the technical assessment of the integrated technologies.

The HELIOS system integration strategy will be aligned with the major milestones of the project, it will consider the architectural design of the system together with the feedback received from the technical tasks, and other target operating environments of the system. A reliable infrastructure will be provided to support the HELIOS system integration and a qualification infrastructure in order to test HELIOS releases.

Continuous integration and testing will be carried out to provide quality assurance and the evolution of the systems. The roadmap is aligned with a three-stage approach envisaged for the delivery of the solution:

- The Alpha release for internal testing should be available on M12 (early release with just few functionalities)
- Public Beta versions of the software will be delivered on M18.
- The final version (Candidate Release) of the platform will be delivered on M28.

The final version consists of a fully tested and qualified system as a consequence of the integration activities where all the selected baseline technologies as well as the modules developed in WP3, WP4 and WP5 have been integrated.

Integration tests will be conducted to verify and validate the robustness, performance, scalability and security compliance of the HELIOS distributed platform.

Where necessary, automated deployment scripts will be implemented for efficient deployment of platform components. The project will create special considerations to enable security and privacy through the adoption of state-of-the-art solution.





A full range of collaborative instruments will be made available for integrating and configuring off-the-shelf solutions: for developers in term of code management, issue tracking and continuous integration testing.

## 2.1 Agile methodology

The HELIOS project will use a software engineering approach called **Continuous Delivery (CD)**, related to Agile methodology. Continuous delivery was first described in 2010 by Jez Humble and David Farley in their book “Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation” [31].

Compared to the traditional waterfall model where one first plans the product and then implements and maintains it, CD relies on a cyclic approach. where the teams produce software in short cycles. In each cycle small improvements are made to the software, the latter then is tested followed by a release of a new version. The aim is to make this process both fast and frequent.

The benefits of CD include low risk releases, faster time to market, higher quality, lower costs, better products and happier teams [32].

CD includes a delivery pipeline described in refcard [33].

According to Humble and Farley, there are three main purposes for the delivery pipeline:

- **Visibility** – in order to encourage cooperation, each aspect of the delivery system (i.e. building, deployment, testing and release) are completely visible to each team member of the project.
- **Feedback** – problems are broadcasted to all team members as soon as they occur, in order to be able to be fixed as soon as possible.
- **Continually deploy** – the utilization of a fully automated process, allowing the team to deploy and publish a software version to any possible environment.

The delivery pipeline in CD incorporates several automation tools and frameworks in at least the following categories: source-code management, build, continuous integration (CI) server, configuration management, deployment and provisioning, as well as testing frameworks [34].

In the architectural style, known as microservices, each application is divided into a group of separate functionalities. This style is often used alongside CD. The benefits of using microservices include increased deployability (deployment independency, shorter deployment tile, simpler deployment procedures and zero downtime) and increased modifiability (shorter cycle time, incremental quality



attributes changes, easier technology selection changes and easier language and library upgrades) [35].

Although HELIOS will follow the Agile methodology, the integration strategy will be aligned with the major milestones of the project lifecycle and dependencies described through architecture and according to the piloting/validation plan described in WP7. Therefore, each incremental working version of the HELIOS platform on delivery will offer increasing levels of functionalities.

In general, the project will have 4-month Sprints, aligned with the releases.

The integrated releases will be delivered on the project month 12 (build upon components with alpha status), project month 18 (build upon components with minimum a beta status) and month 28 (build for piloting). The basic HELIOS distributed operations reports will be delivered on the Project Month 34.

## 2.2 Best practices

The principles of the Agile methodology are listed in [3] and include:

- To satisfy the needs of the customer by utilizing early and continuous publishing of valuable software is the highest priority.
- Evolving requirements, even later in the development process are welcomed since as such the agile delivery can provide advantage for the competitive advantage of the customer.
- The period for the delivery of working software should be frequent, by which one understands – couple of weeks to couple of months, although the former time scale is preferable.
- Daily collaborative work between business people and developers throughout the project.
- The team of people working on the project, should be of motivated individuals. To be successful they require not only the correct environment and support, but also trust level that the job can be done.
- From the various possible ways to transfer/transmit information to the team members and between them, the most effective and efficient strategy remains the face-to-face conversation.
- One measures progress by the working software produced.



- To realize sustainable development, one requires the utilization of agile. Various stakeholders (i.e. sponsors, developers as well as users) need to work at a constant pace for an undefined period of time.
- To increase agility, one requires a constant awareness to technical excellence and the presence of good design.
- “Simplicity is the art of maximizing the amount of work not done--is essential.”
- Self-organizing teams are capable to develop the most suitable architectures, requirements, and designs.
- The ability of the team to evaluate itself at regular intervals, in order to improve effectiveness and then to adjust its behaviour correspondingly.

In [4], Slethold et al. identify 35 agile practices, which are listed in the table below:

List of agile practices. Table reproduced from Slethold et. al [4]	
Practice number	Agile practices
1.	Priorities (product backlog) maintained by a dedicated role (product owner)
2.	Development process and practices facilitated by a dedicated role (Scrum master)
3.	Sprint planning meeting to create sprint backlog
4.	Planning poker to estimate tasks during sprint planning
5.	Time-boxed sprints producing potentially shippable output
6.	Mutual commitment to sprint backlog between product owner and team
7.	Short daily meeting to resolve current issues
8.	Team members volunteer for tasks (self-organizing team)
9.	Burn down chart to monitor sprint progress
10.	Sprint review meeting to present completed work
11.	Sprint retrospective to learn from previous sprint
12.	Release planning to release product increments



13.	User stories are written
14.	Give the team a dedicated open work space
15.	Set a sustainable pace
16.	The project velocity is measured
17.	Move people around
18.	The customer is always available
19.	Code written to agreed standards
20.	Code the unit test first
21.	All production code is pair programmed
22.	Only one pair integrates code at a time
23.	Integrate often
24.	Set up a dedicated integration computer
25.	Use collective ownership
26.	Simplicity in design
27.	Choose a system metaphor
28.	Use class-responsibility-collaboration (CRC) cards for design sessions
29.	Create spike solutions to reduce risk
30.	No functionality is added early
31.	Refactor whenever and wherever possible
32.	All code must have unit tests
33.	All code must pass all unit tests before it can be released
34.	When a bug is found tests are created
35.	Acceptance tests are run often and the score is published



### 3 HELIOS architecture components

The HELIOS draft system architecture is defined in WP3, in the “Confidential<sup>1</sup>” deliverable D3.1 “Draft system architecture and basic API specification”, which is released at project month 6. The HELIOS consortium will provide new and updated versions of this architecture in the future public document D3.2 “Final system architecture and API specification” that will be released at project month 18. Figure 1 depicts the draft logical architecture of the HELIOS platform with the foreseen modules/components to be integrated during the system integration and operation work.

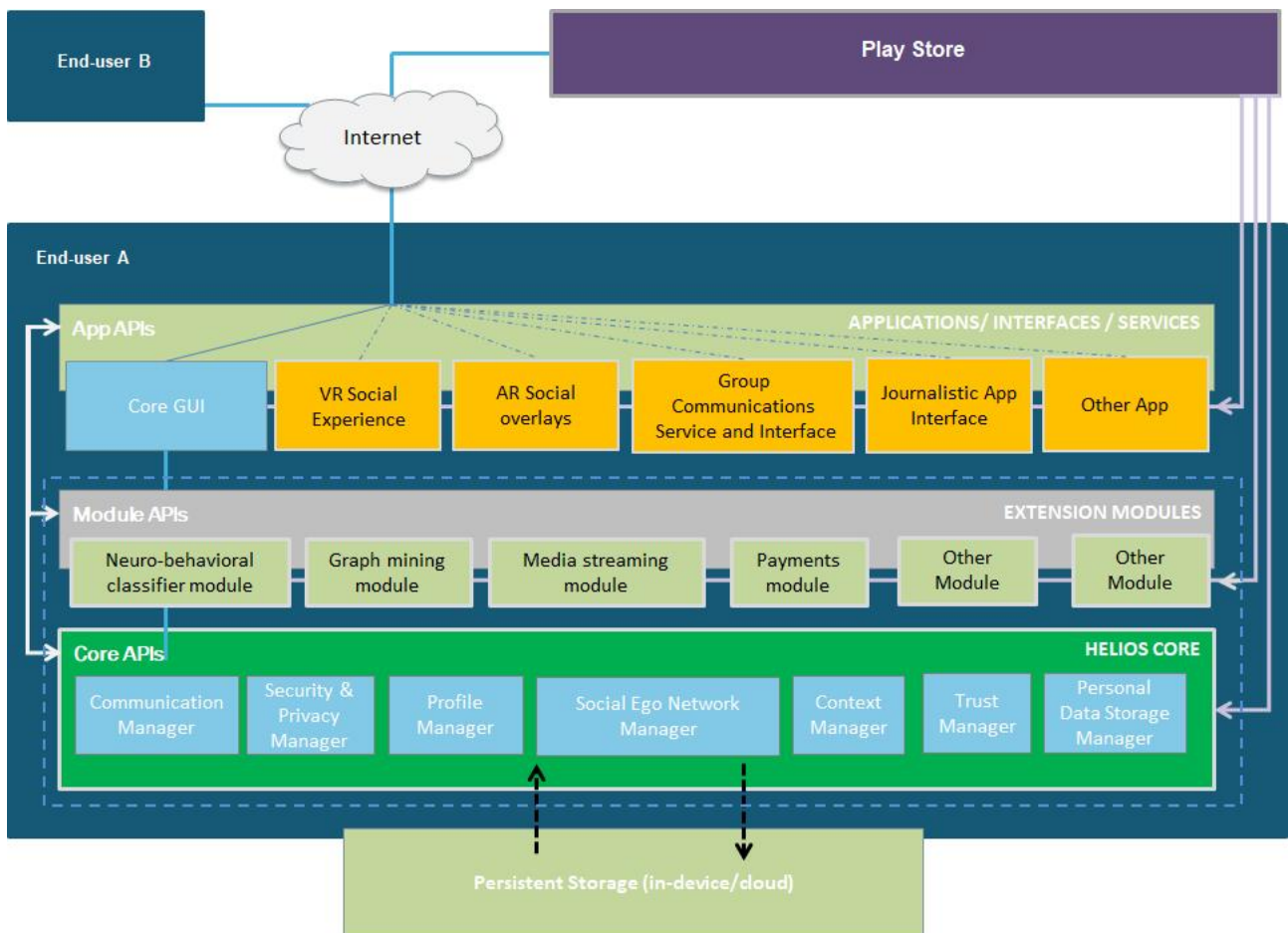


Figure 1. HELIOS logical architecture components

<sup>1</sup> [https://ec.europa.eu/research/participants/data/ref/h2020/other/hi/secur/h2020-hi-guide-classif\\_en.pdf](https://ec.europa.eu/research/participants/data/ref/h2020/other/hi/secur/h2020-hi-guide-classif_en.pdf)



The draft logical architecture of the HELIOS platform is organized into three logical layers and it adopts a highly modular approach in which platform functionality can be extended by adding new components/modules to the platform.

1. **HELIOS core**, the *lean* heart of the system, takes care of basic connectivity, security, and social networking functions, developed by the project.
2. **HELIOS extension modules** aim at providing additional features on top of the HELIOS core components that can be used by the applications. Some extension modules are planned to be developed by the project, and later more extension modules can be developed by the community. In order to minimize HELIOS footprint, Extension modules are only installed when they are needed for a specific functionality. The basic HELIOS functionalities offered by the HELIOS core can be used without any extension modules.
3. **HELIOS applications** utilize the functionality and structured information provided by the HELIOS core and installed extension modules, in order to communicate with other HELIOS applications. The applications can also be developed by third-party developers, such as open-source communities, who are looking to provide novel user-centric services for the HELIOS platform. Applications might require a certain extension module to be present in the platform in order to provide a certain functionality. For instance, Virtual Reality/Augmented Reality applications are likely to require additional modules to be installed on top of the core.

It should be noted that architecture is designed to be general enough to be implemented for multiple platforms. However, in the scope of this project, Android is the target platform. Therefore, the dashed blue area seen in the Figure 1 combining the *HELIOS core and HELIOS extension modules* may be implemented as separate module layers or they may be unified into one.

As such, the testing and integration of the platform components will be divided into three separate levels of *interactions* (going from low-level to more high-level interactions), as follows:

1. *Component level interactions* inside the HELIOS core. This will include integration of the separate functionalities used by the core components from other core components.
2. *Interactions between the foreseen layers*, i.e., HELIOS applications, HELIOS extension modules, and HELIOS core. These interactions will possibly include inter-process communications, such as using Android system functionality for IPC<sup>2</sup> (e.g., Intents, Binder, or Messenger). This might also include interactions with an outside personal data storage (persistent storage).
3. *Interactions between different users*, i.e., HELIOS applications exchanging messages with other HELIOS applications, utilizing the functionality offered by the HELIOS core and extension modules. This will include testing the whole communication chain with related security and privacy functionality enabled, as enforced by the platform based on the user's settings.

---

<sup>2</sup> <https://developer.android.com/training/articles/security-tips.html#IPC>



## 4 HELIOS support tools

---

In order to ensure that all the development and integration activities are properly coordinated, the HELIOS consortium plans on setting up a series of support tools. Section 4 introduces these HELIOS (software management) support tools.

### 4.1 Project management tools

The following tools are used to support technical project management and integration:

- 1) Jira by Atlassian [5]. HELIOS will use Jira for agile software development (Kanban/Scrum) boards. The project will acquire Jira licences for the technical Work Package leaders and individual developers as needed.
- 2) Confluence by Atlassian [6]. Integrated with Jira, HELIOS will use Confluence as a technical development team workspace with the same licensing policy as Jira.
- 3) GitLab and GitHub issue tracker: an important element in the development process is the issue tracker. The issue tracker allows to create an issue report, assign it and track the progress and keep a state (assigned, progressing, resolved). GitLab and GitHub integrates an issue tracker.

### 4.2 Distributed version control system

One of the most important tools for a software development team is the code repository, which usually incorporates some form of a revision control system. Traditionally, software projects have been using a Centralized Version Control system (- CVS), and later Subversion (usually abbreviated SVN). SVN is a complete re-implementation of CVS and a good tool that resolves some of the issues with CVS. Unfortunately, the philosophy of the tool remains the same as CVS. While the tool is useful for small projects, with a rapid growth of the project, the limitations of the method of work that these tools enforce are evident [7].

SVN implicitly adapts a linear approach to the software development cycle. The development is a series of partial versions of the application, each one superseding the old one, until the creation of a last, definitive version. In practice, development does not work in that way. Sometimes work proceeds in parallel for different features, sometimes a bug from previous versions requires fixing before finishing a whole new version. Moreover, one should consider the many other circumstances that might arise and present a strong problem for the linear model.

A viable alternative is to give up the linear model and instead adopt a model of development using branches from a code base. A branch is an independent replica of the code. A branch, therefore, can change without affecting the work in other branches or the code base. Each new feature or correction is developed in a separate branch that is then, in turn, merged with the main branch when



required [8]. Although we can employ branched through SVN, in practice it is a very cumbersome process.

Another important limitation is the centralized architecture of SVN. There is a central node and some local nodes where the actual developments are made. This represents variety of limitations and problems, including for instance the failures if the central repository is broken or somehow lost.

In order to circumvent those problems, the version control of the HELIOS repository system is going to rely on a more advanced tool, such as the open source tools Git or Mercurial. Both tools allow a non-linear, distributed development process. Git [9] was created in 2005 as a tool for Linux kernel development, but in the last few years it has become the mainstream solution for code development in various fields.

One of the reasons of the success of Git has been the popularity of development portals like GitHub[10], or GitLab [11]. These portals offer an integrated development environment with other features including issue tracking, wiki, and some other tools allowing a continuous integration process.

GitLab is an open source application by itself and it can be installed for private infrastructure development. Currently, for the first phase, there is a private repository, available for HELIOS project partners only at <https://gitlab.atosresearch.eu/ari/helios>. As the project evolves, public versions will be offered to the open source community through GitHub <https://github.com/orgs/helios-h2020>.

### 4.3 Real-time collaborative tools

The HELIOS project consists of a large European consortium including artists, business analysts, legal partners, industrial and research partners from several institutions across Europe.

To ensure and support a smooth software integration activity and maximize the efficiency and productivity of the developers, a real-time chat platform/collaborative software will be provided [18].

The tool needs to fulfill the following five principal requirements:

- **Private and temporary channels** to allow developers to discuss about specific topics in separated virtual rooms;
- **Markdown Code Formatting** to easily carry out discussions about code, data structure, objects quality and their purpose;
- **File or Images/Screenshot upload** to allow developers to share screen shots or pieces of the code;
- **Optional requirement: Solid permissions.** This is necessitated due to the levels of different access permission required by the developers;
- **GitLab/GitHub integrations** since the consortium is adopting GitLab/GitHub as a software code repository.









Before selecting the best real-time collaborative tools for HELIOS, the consortium investigated some tools already available on the market.

The following table summarizes the advantages and disadvantages of the tools that have been investigated and include some general comments:

Table 1. Real-time collaborative tools investigated in the context of HELIOS

 slack [18]	 GITTER [20]	 Mattermost [21]	 DISCORD [22]
<p>Pros (+) / Cons (-)</p> <ul style="list-style-type: none"> <li>+ Threads in chat</li> <li>+ Easy to setup and to manage the channels</li> <li>± One to one voice and chat</li> <li>- Limited data archival on free plans</li> <li>- Harder to access, as you need to setup a proxy invite page</li> </ul>	<p>Pros (+) / Cons (-)</p> <ul style="list-style-type: none"> <li>+ Easy to access (no need for custom sign ups etc)</li> <li>+ Archival of chat</li> <li>- Lack of separate channels. <u>This is a key requirement the software does not fulfill.</u></li> <li>- Lack of threads in channels. <u>This is an expected requirement the software does not fulfill.</u></li> <li>- Limited features outside of text chat.</li> </ul>	<p>Pros (+) / Cons (-)</p> <ul style="list-style-type: none"> <li>+ Self-hosted</li> <li>+ Unlimited history, users, channels, etc</li> <li>+ LaTeX and full Markdown</li> <li>- Limited visualization—several rendering issues</li> <li>- The mobile app is not working properly</li> <li>- No video or audio conferencing.</li> </ul>	<p>Pros (+) / Cons (-)</p> <ul style="list-style-type: none"> <li>+ As Slack, it is easy to setup and to manage channels</li> <li>+ Easier to access - just click a link to login</li> <li>+ Group voice channels</li> <li>+ Screen sharing available in groups</li> <li>+ Larger data archival size for searching</li> <li>± Roles, allowing you to clearly see who is experienced/maintainer etc, if needed</li> <li>- No ability to create threads in chat. <u>This is an expected requirement that the software does not fulfill.</u></li> </ul>



<p><b>General Comments:</b> Slack is a robust tool. Its threaded chat is quite good for discussing specific points in detail without derailing chat</p>	<p><b>General Comments:</b> Gitter is quite easy to use and is basically just an evolution of IRC client, so while its fine as an entry point there are few limitations that make other tools better suited for the consortium needs.</p>	<p><b>General Comments:</b> Mattermost is indeed an interesting solution even more so, if the consortium wants to run the software within its own security parameters on a private cloud. The latter provides a high level of security and control.</p>	<p><b>General Comments:</b> Discord is quite a good tool given it is free and with many features available. However, the inability to create threads within the chat is an expected requirement that the software does not fulfill. Discord is probably the best client in terms of experience and accessibility.</p>
---	---	---	---

Taking into account the result of the investigation the consortium selected Slack as the project’s real-time collaborative tool.

In addition, by the end of the project, HELIOS consortium will use HELIOS platform for its internal communications.

Figure 2 presents a screenshot of the HELIOS h2020 chatting room hosted in Slack.

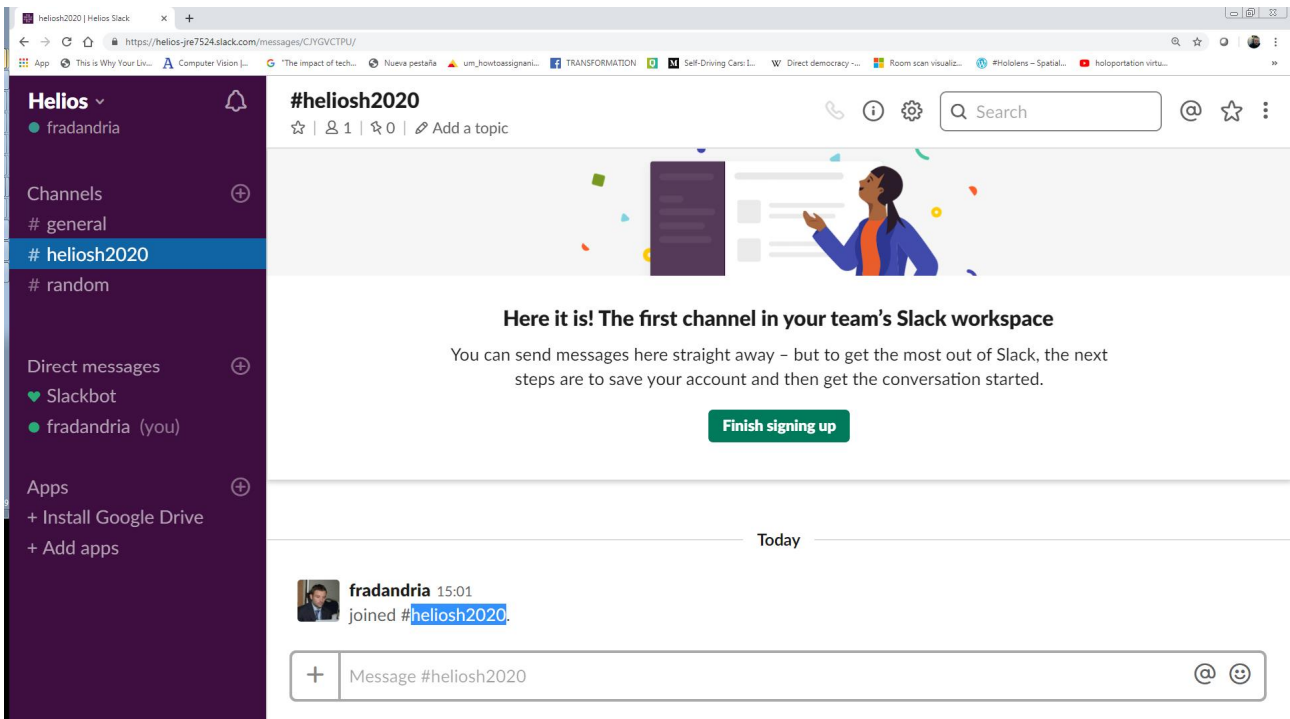


Figure 2. HELIOS Real-time collaborative room in Slack



## 5 Integration process

---

### 5.1 Testing process: Integration testing & Validation testing

This section deals with the tests required for the integration procedures. Testing is key in any integration activity. Within the context of HELIOS integration, testing is even more important since the complete system is a composition of individual modules, that are developed by different HELIOS consortium partners.

This section describes the testing methodology and the stages established in HELIOS for integration. The process consists of three stages:

- Unity tests
- Integration tests
- Validation tests

This methodology resembles the V model for testing, as described in [16]. Nevertheless, HELIOS approach is more sophisticated, since it includes automatic tests and continuous integration. In HELIOS, new functionalities are supported by different extension modules, and thus the project must ensure not only the interoperability between them but also between the applications and the core. The HELIOS applications, on the other hand, have to be tested with the different extension modules and the core. See section 3, for a description of the main components. For the remained of this section, module is used to refer to any component (core, extension modules, applications).

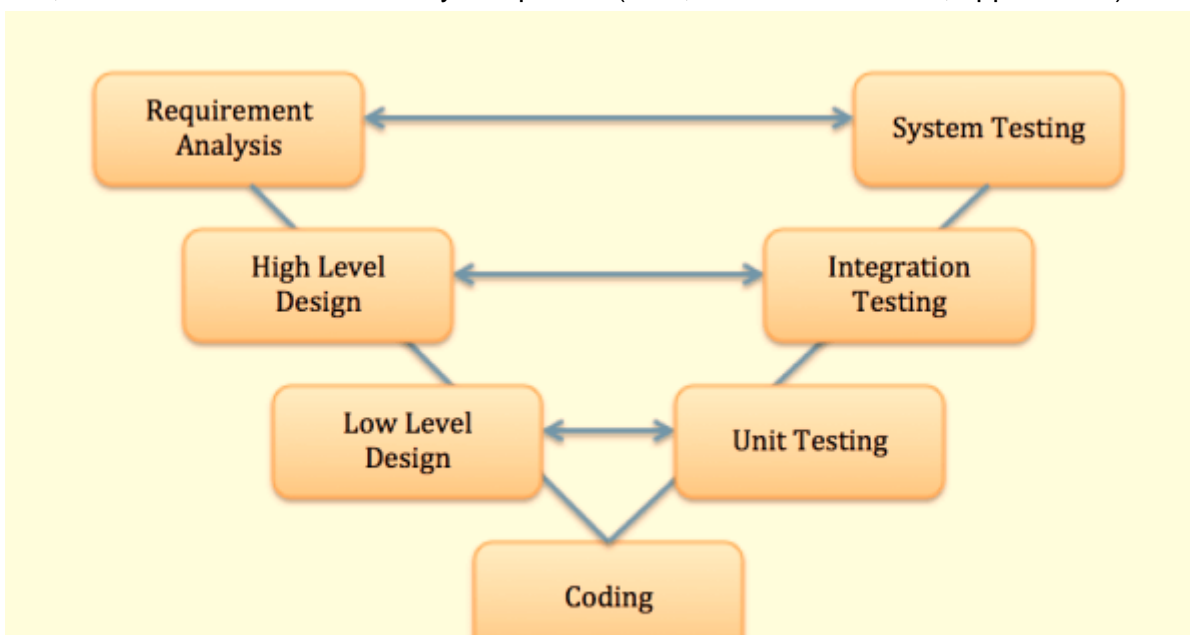


Figure 3. V model for software testing (Image taken from [16])[2]



The classical or standard V model is presented in Figure 3 (taken from [16]), where one can see the characteristic shape of the model. The right-hand side shows three kinds of tests, which are equivalent to the three kinds of tests listed before in this section.

The description showcases how these tests satisfy different levels of abstraction: the low level (individual component), the high level (diagram view, integrating different components) and the requirement analysis level (system view, providing meaningful functionalities).

In HELIOS, these tests present different characteristics and are carried out in successive stages, as detailed in the following paragraphs.

### **Unit tests**

The purpose of the unit tests is to check the functionality and the performance of a certain module, as a primer test in the integration procedure and before performing joint tests with other HELIOS modules. It must be taken into account that unit tests are not only done at the integration stage.

These tests must also be performed during the development cycle of each module to ensure the functionality before starting the integration process. The partner in charge of developing a certain HELIOS module must provide the following information:

- The component to be integrated with release version numbering, either as a code release (branch) in the repository or application package (APK in case of Android).
- Documentation about the component, its functionalities and its interfaces (e.g., REST APIs, other API descriptions, that provide methods to activate module capabilities).
- Configuration files, if needed.
- Benchmarking tests carried out and passed by the component, including information about the testing environment.
- A set of benchmarking tests (e.g., scripts) to check the functionality of the component once it is deployed on the integration environment. This information must also include the expected results of the tests. When possible, the test will automatically compare the obtained result and the expected one in order to detect a difference in the functionality. As a best practice, this set of tests will be included in a specific folder of the integration repository under the release to be tested. Section 0 introduces automatic tools for testing. Automatic tests ensure that the test conditions are under control.

These tests validate the right functionality of a module when it is received in the integration process. Thus, in the case of malfunctions in latter testing steps, it is possible to step back to a stable, initial situation for each individual module.



### **Integration tests**

The execution of an integration test requires a diagram design, the specification of the interfaces (including the way the different components interact), the specification of the data input and the expected output or behaviour. As stated in [17], integration tests pay special attention to the interfaces between modules. Sometimes a certain module can be unavailable or is missing parts of its intended functionality. In this case, the missing module(s) can be emulated, as long as emulated modules interface and the behaviour are the same as in the real module.

The continuous integration approach in HELIOS may impose additional requirements to the integration tests, such as the update of a certain module in the chain without stopping the complete diagram.

### **Validation tests**

A validation test aims to check the fulfilment of the needs of a complete use case, involving several HELIOS components. “Use case” must not be interpreted in the sense of the HELIOS scenarios created in T2.1 in the HELIOS work plan, but they are use cases that are specified ad-hoc for the testing process, describing a complete functionality (e.g., in the context of HELIOS one of this use cases would be the exchange of messages between final devices using the P2P approach of the project).

The preparation of a validation test requires the description of the use case, the identification of the involved modules and the description of the expected result.

Beyond these successive stages of system complexity, there are several kinds of tests (which can be carried out during each stage):

- **Functional test:** the objective is to check that the expected functionality is achieved, regardless resources that are used.
- **Performance test:** it measures the resources required for the test execution since the focus is not only on achieving the functionality but also on the evaluation of the resources.
- **Stress or load test:** it determines if the functionality remains balanced even in cases of large loads.
- **Exhaustive test:** it tries to include all the possible uses or situations of the system under test, beyond the normal functionality of the component.

The execution of some of these described tests above can be automated, e.g., to measure the response of the system under test to face different load levels). Sub-Section 5.2 details the tools specified in HELIOS to perform automated tests.

A summary list of **the best practices** to consider in the process of HELIOS integration testing are as follows:



- Modules must have been tested individually before starting the integration process.
- Developers must provide a set of tests to verify the functionality of the modules at the beginning of the integration process (unit tests).
- Documentation must be generated along the entire process, including the expected results, the obtained results, possible problems, additional comments and environment conditions. These test result documents are used as feedback to the other work packages as feedback for the agile development process.
- In a project like HELIOS, where modules are provided by different partners, the accurate specification of each module interface is key.

## 5.2 Support tools for Automated Testing

As stated in the previous section, each developer has to develop the unit test for their modules. Although these tests will be executed automatically, their writing is not automatic. There are several libraries and tools available that can assist the developers in the task of creating the automated test cases. These libraries facilitate both the creation of the test code, as well as the automating the execution of the tests. The following provides a non-exhaustive list of possible tools to use.

**JUnit [12]:** This is the most famous library for Java applications. There are ports for nearly all main programming languages, as NUnit, for .NET, CUnit for C, or JUnit for JavaScript.

**pytest [13]:** Several libraries are available for Python. There is a version of JUnit for python, (called pyUnit), but usually developers use unittest (included in the standard library) and compatible improved versions as nose [38] and pytest. There is a complete ecosystem around these libraries in python.

**Postman [14]:** Testing APIs offered by web servers as a specialized task, different from traditional testing code. The testing APIs is often is mandatory since nowadays usually the way of communication between modules in the application and with the external world is carried out through APIs. Some tools such as Postman are used to write the test for webservice testing.

**Selenium [15]:** Traditionally, testing a web application required human intervention. Now there are utilities as Selenium that allow to make the tests within the web browser automatically. Selenium has several operation modes, with one of them able to capture the sequence of interactions from a human and repeat it in the automatic test. In addition to testing web applications, it can be used to automate testing for Android applications as well (using Selenium for Android).

**Robotium [37]:** Robotium is a test automation framework for Android that supports testing of native Android apps as well as hybrid apps. Robotium Recorder supports recording test cases for Android applications easily, which helps the developers to create tests covering also user interface interactions. Robotium can be integrated as part of the Continuous Integration process.



### 5.3 Testing Infrastructure approach

As a prominent alternative, HELIOS platform is likely to be deployed following a microservices architecture approach. Microservices architecture is a framework that has been used in the recent past to tackle challenges associated with the conventional monolithic approach [23].

Microservices is a popular term nowadays, despite the unavailability of a common exact definition. In general, microservice architecture pattern is the practice of splitting the whole application into several small modules that implement one or more functionalities, and are as such loosely coupled. In the past years, it has been a good practice to divide the application in different modules. But the difference to this approach is that, these modules usually were bundled together in one executable. The modules in a microservices architecture are instead different executables [24].

In some cases, the “microservices” concept is associated with the terms “serverless” and used as synonymous. This is not the case in the context of HELIOS. Ideally, the HELIOS consortium wants to shape an overall methodology where each microservice delivers just one functionality or has a specific mission and communicates with other microservices using a network interface (each microservice can, e.g., expose a REST API or other API and consume some REST APIs from the others). In such an approach it is ideal to use a containerization engine such as Docker. Docker is a tool designed to make it easier to create, deploy, and run applications by using containers [25]).

The development and testing infrastructure will be composed of several logical elements:

- Android development tools, especially Integrated Development Environments.
- The Docker runtime software running on a server
- A private Docker image register, to allow HELIOS developers to publish development images (container that deliver the specific software functionality) in a specific HELIOS Docker rep instead than in the public available generic Docker hub.
- A Software for automating the image creation, testing and deployment, as Jenkins [27] or CircleCI [28].
- Container orchestration, using Docker Swarm or Kubernetes [26] is an open-source system for automating deployment, scaling, and management of containerized applications [26])
- Some run-time and monitoring tools for the containers.

In terms of hardware platforms, given that the project will deliver a distributed platform, HELIOS is different from the normal client-server approach. It is necessary to have a testing environment with different elements:



- Mobile devices (Android smartphones or tablets)
- Raspberry Pi or similar Linux-based system on a chip devices, needed for some multimedia functionalities.
- In order to encourage the adoption of the project, HELIOS may offer a private cloud infrastructure running some emulated Linux-based personal servers. Using this cloud, the users can learn certain aspects of HELIOS, such as multimedia or VR, without the installing a “personal server”, e.g., with Raspberry Pi. Using this private cloud is a good option for non-technical, but otherwise advanced users.

## 5.4 Operation Infrastructure

The operation infrastructure of HELIOS will be a replica of the testing infrastructure. But in the case of the operation, some of the elements are the property of the users. In this case, the challenge is the deployment of the application in user’s mobile device or personal server, and keeping the application updated for all the nodes. There is a potential risk of lack of stability in the system if several versions of HELIOS are running in the different user devices.

- In the case of mobile devices, the solution is the Android Play Store. The HELIOS platform modules will be offered as applications, so they can be downloaded from the official store, Google Play Store. After that, any updates will be provided automatically when available in the Play store, without the need of user intervention.
- The private cloud may be provided by the project, as it was stated in the last section, enabling installation and updates of certain functionalities of the HELIOS software using Continuous Deployment, the same as on the testing infrastructure.
- However, this does not relate to basic HELIOS functionalities running in a user’s personal server. The exact mechanism of deployment is undecided yet, so we can currently only present some alternatives. For instance, an advanced user would be able to “pull” some docker images from a public Docker hub, but the update process is not automatic. Another solution is that a HELIOS module could be packed as a Linux distribution packet, so the installation and updating would be similar to other software in the Linux distribution. Unfortunately, this is completely dependent on the distribution (Raspbian, Ubuntu, Fedora). There are some portable packet formats, such as flatpack [29] and snap [30], that are not very extended by the moment but can be a solution in the future. In any case, further investigation about the software distribution will take place in the next months.





## 6 First HELIOS software integration and delivery plan

---

As previously mentioned, the HELIOS project aims to follow a Continuous Integration (CI), Testing and Deployment Methodology (CITDM). This methodology is a part of well-known approach adopted for quality assurance and the evolution of the complex, heterogeneous and distributed systems as introduced in the Section 2 of this document.

This section highlights the system integration roadmap for the HELIOS project. The roadmap will be aligned with a three stages approach envisaged for the solution delivery. The Alpha (early release with just few functionalities) and Beta versions of the software will be delivered on M12 and M18, respectively. The final version (Release Candidate) of the HELIOS platform will be delivered at Project M28.

This HELIOS delivery approach has been aligned with the major milestones of the project, considering also the architectural design of the system (WP2 and WP3) together with feedback received from technical tasks, and other target operating environments for the system.

Figure 4 highlights the key milestones in the development and delivery of the HELIOS platform:

- At M4 (April 2019) the “System integration and operation (WP6)” activity starts, and the task T6.1 “Integration Strategy and Planning” kicked off. During this time, the consortium worked on the main outcome of the T6.1, this D6.1 report.
- By M6 (June 2019), the deliverable D3.1 drafts the first and preliminary HELIOS system architecture and basic API specification. On the other hand, the main tools to support the HELIOS Integration will be set up. These tools include:
  - An internal Git-repository to the HELIOS project (<https://gitlab.atosresearch.eu/ari/helios>)
  - Project management tools (see Section 4 of this document for further details)
  - A real-time communications tool (see Section 4.3)
- At M9 (September 2019) the HELIOS consortia will deliver the first Client-server core emulation software with the basic API’s of the core (D3.3). This software will mainly be produced by the WP3 but will rely on the tools set up by WP6.
- At M12 (December 2019) the first, standalone (alpha) HELIOS platform will be released integrating core functionalities and preliminary versions of system modules and services that are expected to be available at the time of release based on their individual timeline of implementation within the project (WP3, WP4, WP5). This release will support laboratory tests (from technical WPs) and limited trials and small pilots from WP7.
- The second release of the HELIOS platform will be published at M18. It will be the first “public” release with updated versions of components from WP3, WP4 and WP5. This release will



still support laboratory tests (from technical WPs) and, on the other hand, will deliver a code mature enough to support the execution of trials and pilots from WP7. Code will be finally published in a public repository, such as GitHub: <https://github.com/helios-h2020>.

- At M28 the third and final HELIOS platform software will be published with almost the final components from technical WPs. This release will mainly support pilots from WP7. Code is finally published in a public repository, such as <https://github.com/helios-h2020>.
- Finally, on M34 the consortium will deliver the public report deliverable D6.5 “HELIOS system operations”, in which all reports concerning the platform operation will be included.



## HELIOS D6.1 (REPORT)

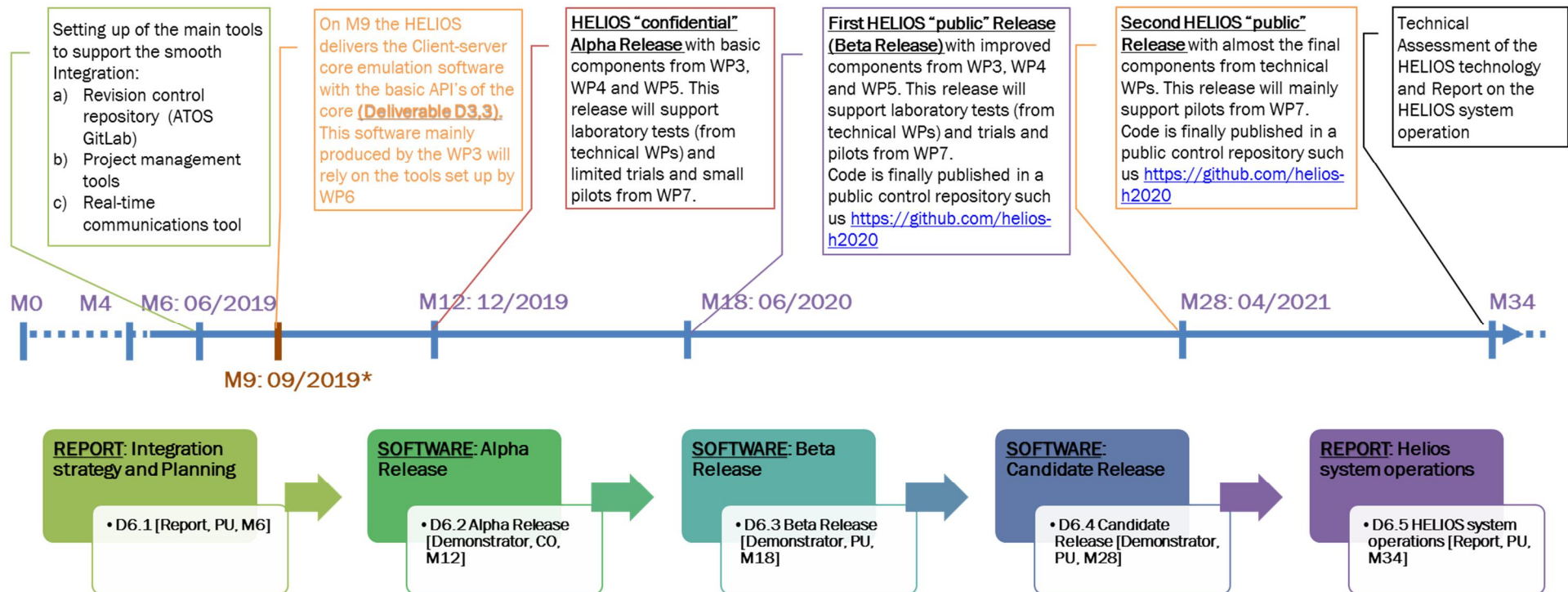


Figure 4. First HELIOS Integration roadmap



## 7 Conclusion

---

HELIOS aims to design, implement and deliver a decentralized social media platform relying on heterogeneous and distributed technologies that will address the dynamic nature of human-to-human and human-to-machine interaction.

This deliverable (named D6.1 “Integration strategy and planning”) aims to guarantee that the Continuous Development, Continuous Integration and Operation (DevOps) approach adopted by the HELIOS consortium will be executed in a coordinated way.

Consistently with this approach, D6.1 deliverable defined a methodology and set up relative support tools that the consortium will adopt and utilize on a daily basis, for the development and the integration of the HELIOS Social Network platform.

Finally, the document defined the preliminary integration roadmap with milestones and a calendar view for the integration of the different components and releases of the three main versions of the platform. However, such an integration roadmap will be revised and adapted during the duration of the project’s lifecycle. Thus, further updated version of the plan currently described in this document, will be delivered along with the upcoming reports that are to be produced and delivered within the framework of the activities of WP6.



## 8 References

---

- [1] HELIOS Project Contacts:
  - a. Website - <http://HELIOS-social.eu/blog/>
  - b. Subscribe to HELIOS Newsletter - <http://HELIOS-social.eu/blog/>
  - c. Follow us: Twitter – <https://twitter.com/HELIOS-EUProject> and Facebook – <https://www.facebook.com/HELIOS-EUproject/>
- [2] Continuous delivery approach: [https://en.wikipedia.org/wiki/Continuous\\_delivery](https://en.wikipedia.org/wiki/Continuous_delivery)
- [3] Principles behind the Agile Manifesto: <http://agilemanifesto.org/principles.html>:
- [4] Paper: “What Do We Know about Scientific Software Development’s Agile Practices?”: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6081842>
- [5] Jira by Atlassian <https://www.atlassian.com/software/jira>
- [6] Confluence by Atlassian <https://www.atlassian.com/software/confluence>
- [7] “Why You Should Switch from Subversion to Git”: <https://blog.teamtreehouse.com/why-you-should-switch-from-subversion-to-git>
- [8] “Git Branching - Branches in a Nutshell” <https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell>
- [9] Git Homepage: <https://git-scm.com/>
- [10] GitHub: <https://github.com/>
- [11] GitLab: <https://about.gitlab.com/>
- [12] Junit; <https://junit.org/junit5/>
- [13] pytest <https://docs.pytest.org/en/latest/>
- [14] Postman: <https://www.getpostman.com/>
- [15] Selenium: <https://www.seleniumhq.org/>
- [16] Tutorial: “What is V Model in Software Testing? Learn with SDLC & STLC Example”. Available in: <https://www.guru99.com/v-model-software-testing.html>
- [17] Tutorial: “What is Integration Testing” Available in: <https://www.softwaretestinghelp.com/what-is-integration-testing/>
- [18] Collaborative software [https://en.wikipedia.org/wiki/Collaborative\\_software](https://en.wikipedia.org/wiki/Collaborative_software)
- [19] Collaboration hub for work: <https://slack.com/intl/en-gb/>



- [20] Chat and networking platform that helps to manage, grow and connect communities through messaging, content and discovery: <https://gitter.im/>
- [21] High Trust Messaging for the Enterprise: <https://mattermost.com/>
- [22] All-in-one voice and text chat for gamers that's free, secure, and works on both your desktop and phone: <https://discordapp.com/>
- [23] Why should I consider Microservices architecture for my next application?  
<https://medium.com/@rasheedamir/why-should-i-consider-microservices-architecture-for-my-next-application-a8041e0d781>
- [24] Pattern: Microservice Architecture: <https://microservices.io/patterns/microservices.html>
- [25] What is Docker? <https://opensource.com/resources/what-docker>
- [26] Production-Grade Container Orchestration: <https://kubernetes.io/>
- [27] Jenkins – an open source automation server which enables developers around the world to reliably build, test, and deploy their software: <https://jenkins.io/>
- [28] Circleci: Automate your development process quickly, safely, and at scale:  
<https://circleci.com/>
- [29] Flatpack website: <https://flatpak.org/>
- [30] Snapcraft website: <https://snapcraft.io/>
- [31] Book: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. REF Humble, Jez and Farley, David (2010). "Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation", Addison-Wesley.
- [32] Continuous delivery: Humble, Jez (2010). "Continuous delivery",  
<https://continuousdelivery.com/>.
- [33] Duvall, Paul (2012), "Continuous Delivery: Patterns and Anti-Patterns in Software Lifecycle" [http://www.dccia.ua.es/dccia/inf/asignaturas/MADS/lecturas/10\\_Continuous\\_Delivery\\_Dzone\\_Refcardz.pdf](http://www.dccia.ua.es/dccia/inf/asignaturas/MADS/lecturas/10_Continuous_Delivery_Dzone_Refcardz.pdf)).
- [34] Samuelson, Eric and Mason, Carlton. "Automate continuous delivery through a delivery pipeline", [https://www.ibm.com/cloud/garage/practices/deliver/practice\\_delivery\\_pipeline/](https://www.ibm.com/cloud/garage/practices/deliver/practice_delivery_pipeline/)
- [35] Chen, Lianping. (2018). Microservices: Architecting for Continuous Delivery and DevOps. 10.1109/ICSA.2018.00013.
- [36] Avoiding Application Integration Fails: <https://devops.com/avoiding-application-integration-fails/>
- [37] Robotium: <https://github.com/RobotiumTech/robotium>
- [38] Nose Library <https://pypi.org/project/nose/>