Grant Agreement nº	732463
Project Acronym:	OpenReq
Project Title:	Intelligent Recommendation Decision Technologies for Community-Driven Requirements Engineering
Call identifier:	H2020-ICT-2016-1
Instrument:	RIA (Research and Innovation Action)
Topic	ICT-10-16 Software Technologies
Start date of project	January 1st, 2017
Duration	36 months

D1.4 Project standards and infrastructure document

Lead contractor:	TU Graz
Author(s):	TU Graz, ENG, HITEC, QT, SIEMENS, UH, UPC, VOGELLA, WINDTRE
Submission date:	June 2017
Dissemination level:	PU

Project co-funded by the European Commission under the H2020 Programme.

Abstract: This document describes the technological framework and scope as well as specifies the project standards and the development infrastructure in place.

This document has been produced in the context of the OpenReq project. The OpenReq project is part of the European Community's H2020 Programme and is as such funded by the European Commission. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability is respect of this document, which is merely representing the authors view.

Table of Contents

List of Figures

1. Introduction

This document presents the results of tasks T1.4 Specification of technological and project standards and T1.5 Project infrastructure for integration, testing, and deployment.

After an overall view of the technological infrastructure (Section 2) and a reminder, basically extracted from the DoA) of the OpenReq architecture (Section 3), the deliverable describes the development process that will follow the Scrum principles (Section 4) and exposes the main principles of the quality assurance to be carried within the project (Section 5),

Section 6 lists the main coding standards to be applied in the development process, and Section 7 describes in detail the proposed infrastructure for integration, testing and deployment of the OpenReq platform and its different components.

The last Sections (8 to 10) deal with the practical organization about deliverables' preparation and review, and evaluation responsibilities and procedures. This is further expanded in Annexes 1 to 3.

2. Description of technological infrastructure

For the OpenReq Services we plan to run state of the art Spring-Boot Applications offering RESTful services in the backend which connect to MySQL Databases. For presenting user interfaces to the end-users, we include the "Thymeleaf" Framework in combination with Bootstrap.

The Spring boot Applications run on a Java Virtual Machine and there is no need for installing an Application Server such as Tomcat.

Spring boot applications are programmed in the programming language JAVA / JAVA EE. If specific libraries are needed for the implementation of some components (e.g., dependency detection), other programming languages such as python (based on the used libraries) can be used but still RESTful services will be offered.

The following figure illustrates the schematic approach of the OpenReq Services.

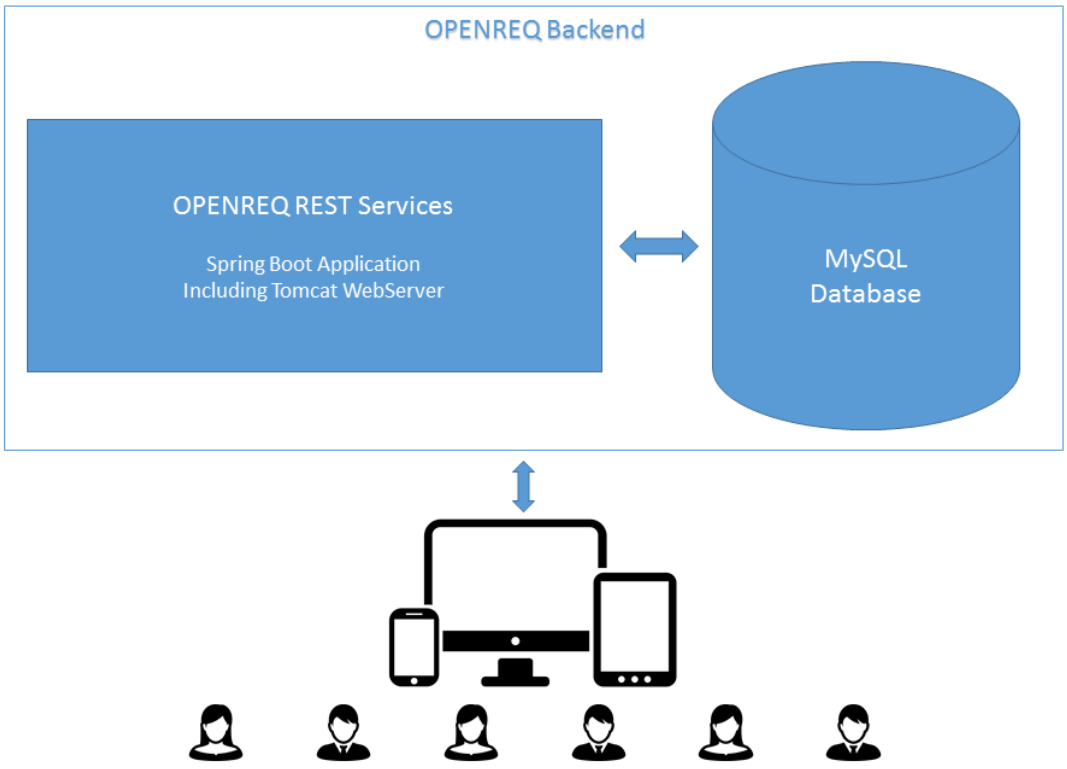

Figure 1 Schematic approach of OpenReq Services.

3. Overall architecture (system components and their relationships

The OpenReq overall architecture will be composed of different parts: the *OpenReq global database* which is based on the OpenReq Ontology, the *OpenReq REST Services* and the different *stakeholder applications* who use the services (see Figure 2). Besides the trial partner applications, the OpenReq Prototype (a showcase of major OpenReq functionalities) accesses the different OpenReq services. The OpenReq Prototype includes basic OpenReq functionalities offered to end users through an understandable and user friendly user interface. In all cases where end users interact with OpenReq functionalities, a special focus will be given on usability.

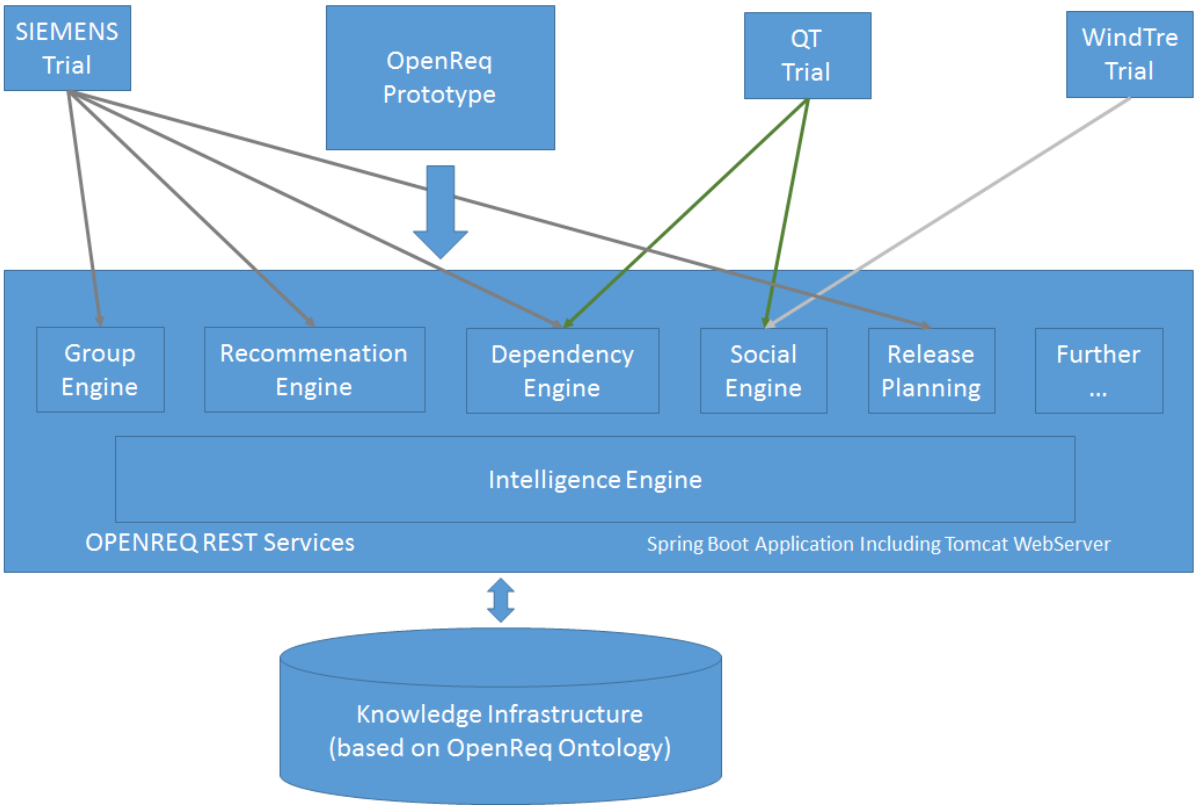

Figure 2: OpenReq overall architecture.

The *OpenReq (REST) services* include, among others, the following engines (components). The OpenReq Prototype and the industry trials exploit the basic functionalities provided by the OpenReq services. Furthermore, services themselves exploit / integrate the services of other components, for example, release planning uses functionalities of dependency detection, recommendation, and group decision making (for simplicity, this is not taken into account in the architecture figure). In the following we give a short overview on the different engines shown in Figure 2 (for more details we refer to the DoA).

a. Recommender Engine

The recommender engine (component) will provide a set of standard micro-services, e.g., querying for new requirements, responsible stakeholders, or reusable requirements. A conversation with the system can be triggered depending on the type of the query and the stakeholder activity. Simple recommendations will be delegated to collaborative or content-based recommenders.

b. Dependency Engine

This component uses a combination of content-based recommendation, sentiment analysis, and natural language processing that support the detection and extraction of requirements dependencies and requirements tracing. For repairing inconsistent requirements, a conflict resolver is used.

c. Group Engine

This component offers support for groups of users in group decision processes. In a first step, the preferences of the different stakeholders have to be collected (for example, regarding a specific requirement). After the preference acquisition is done, this component also supports the moderation of the decision process in such a way that consensus among stakeholders can be achieved (in case of contradicting preferences).

d. Intelligence Engine

This component will encapsulate an analytics backend and include text-mining algorithms that allow for analysis of natural language texts such as text-based documents or user feedback (user feedback can be either explicit or implicit - see Figure 3). In addition to that also interactive visualisation will be supported by this component. In particular, interactive visualisation supports stakeholders in visualizing descriptive and predictive analytics data. An example of such a visualisation is shown in Figure 4. Figure 4 presents the trend of different app review types (e.g., a user requests a new feature or a bug to be fixed) over time, for a specific app and over its different versions.

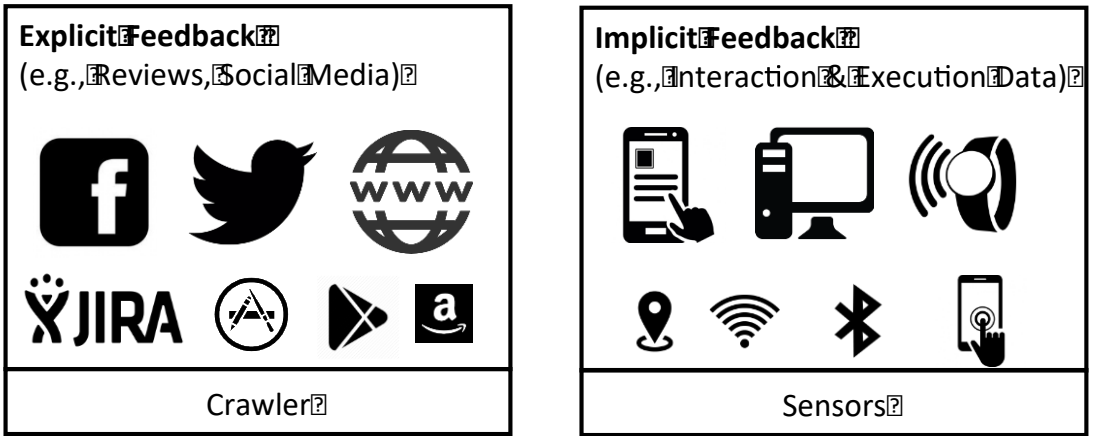

Figure 3: Examples for explicit and implicit feedback.

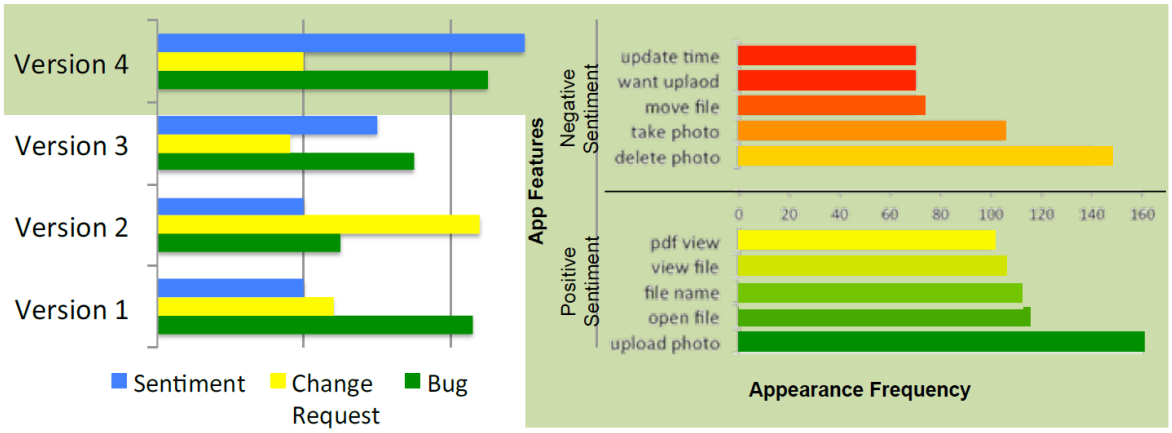

Figure 4: Example trend of different app review types.

e. Knowledge Infrastructure

This component will be responsible for managing OpenReq ontologies, glossaries, indexes (e.g., references), stakeholder profiles, user feedback and usage and interaction logs.

OpenReq Ontologies will be (1) the RE ontology (language for modelling requirements including concrete instances, i.e., the requirements model), (2) the reuse and patterns ontology (language for requirements reuse and patterns), and (3) different domain ontologies (modelling the domains of the applications, e.g. the trials).

The OpenReq reuse and pattern catalogue provides a reusable knowledge base about requirements. The catalogue will be organized according to one or more classification schemas that will be derived from some of the ontologies mentioned above.

Communication among OpenReq components and the integration of OpenReq components with the trial clients (e.g., the Siemens Doors client) will be supported by OpenReq Interfaces. The approach to provide these interfaces will be the following.

OpenReq Interfaces

The OpenReq Interfaces will provide open, unified interfaces for easily integrating OpenReq into external tools in form of connectors (see Figure 1). Examples for high-level interfaces include among others (a) register/Unregister for Recommendation, (b) push context, (c) display recommendation, (d) open artefact and (e) configure.

As a proof of concept we will integrate these interfaces into the tools of the OpenReq industry partners and associated Open Source communities within the scope of the trial implementations.

In order to assure the accessibility of OpenReq in Cloud contexts, OpenReq will be built upon the following service infrastructure.

OpenReq Cloud Services

Due to the massive amount of data which needs to be collected and managed, OpenReq will be made cloud-ready. In particular, OpenReq will be enabled to make use of the cloud data storage to have virtually centralized data storage of high scalability that is easier to access, process, and manage.

Furthermore, OpenReq will provide a dashboard, that summarizes information, exposes the interactive visualization driven by the *Intelligence Engine* (see **d. Intelligence Engine**), as well as various performance measurements. The dashboard will be used to maintain the OpenReq Cloud Platform.

4. Definition of development process

In the OpenReq project we will use an iterative and incremental software development process following the principles behind Scrum.

The software development process is managed with the tool Tuleap (https://mast-tuleap.informatik.uni-hamburg.de/).

The OpenReq product backlog contains four types of items: Description of Action (DoA), Epics, User Stories, and Work Items.

These items are tracked in three trackers: DoA and Epics (DE) Tracker, User Stories (US) Tracker, and Work Items (WI) Tracker.

The DE Tracker contains the high level tasks (as per Description of Action, DoA) and the Agile Epics. The DoA will be already populated, based on the WP descriptions, and should not be changed. Therefore, only the Epics, individuated from the DoA (e.g., after the interviews with the trial partners), need to be inserted in this tracker.

The US tracker contains the user stories (for example, from the requirements gathered in T1.2) that are identified to implement OpenReq. The user stories follow a traditional Agile template.

User stories are divided in smaller items which are tracked in Work Items. A user story has to refer to an Epic or a DoA Task in the Epics tracker (in the Artefact Link field).

The WI tracker contain tasks of different types. These items can be part of a user story, or not (e.g., an implementation task, a simple bug fix, a change request or a finalization of meeting minutes). The Work Items are not necessarily related to development, and should contain also other types of activities (e.g., *prepare presentation for next meeting*). Therefore, a Work Item can either directly relate to a DoA item in the *Epics* tracker or to a User story item.

Sprints contains items that belong to the same Scrum Sprint. A Sprint can contain *Epics (and relative User stories)*, and/or Work Items (for those work items that are not part of a user story but still contribute to a DoA).

As releases are tied to deliverables, an item in this tracker can contain either a set of *Epics*, and Work Items (most likely in the case the deliverable type is R). The release date corresponds to the latest date indicated in the DoA.

Release can be releases of documents or of software components.

In case there is a need to explicitly target a release when working on a Work Item, User Story or Epic, you can mention it in the description field. For example “this contributes to rel #id” will create a link to that release.

A release can be created and tracked using the Agile Dashboard, a Kanban-like visualization of the backlog.

Roles

The Development Team is composed of functional sub-teams divided according to what is indicated in the DoA of each WP Task. The resources within each partner's sub-teams are self-organized.

In general, the Task leader (i.e., an individual within the partner responsible for the task as indicated on the DoA) will take the role of Product Owner. A Scrum Master role, although not

specifically enforced will be taken by the Task leader itself (e.g., a different individual within the same partner organization of the Product Owner). However, the roles are not strictly enforced and can be fine-tuned or modified by the Task leaders on a case-by-case basis.

Events

Within each Development Team, a lightweight Sprint planning will happen before each Sprint to decide the artefacts (Epics/User stories/Work Items) that will be included in the Sprint. The recommended duration for a Sprint is between two and four weeks.

A cross-development team Daily Scrum "standup" meeting (i.e., between development teams in different partner organizations) will not be enforced. However, Daily Scrum should take place for a team in the same partner organization. A cross-development Sprint Review will not be enforced, however, each sub-team is expected to document the output (i.e., increment) of the Sprint in the trackers. A cross-development Sprint Retrospective will not be enforced. At the end of each Sprint each sub-team should have their own retrospective. However, a Retrospective will take place at the end of the Task.

Collaboration

In addition to the issue tracker Tuleap, the OpenReq Team collaborates also using the mailing list openreq-devel (available as an integrated instance of Mailman in Tuleap) and a chat channel (using Slack, openreq.slack.com).

Version Management

As Version Control System we use GIT where all partners can access the needed information. Special branches for all the work packages as well as for the tasks can be established. Deployment of new OpenReq Versions can also be done out of special GIT branches (for instance "releases_WP4"). For details see section on **Git branching policy.**

5. Quality assurance

The quality assurance process in the OpenReq project is, like the work process, also an iterative process. During this process, it will be defined which goals are fulfilled within a specific release. The quality assurance process will take place for each release of the project. Quality assurance operated by each of the partners of the OpenReq project will conform to the normal quality procedures operating within their organisation, at the discretion of the individual partner representatives. Procedures that are mandatory within the partner organisation will be implemented. The quality of the work undertaken in each WP is responsibility of the corresponding WP leader as specified in the DoA, under the supervision of the Scientific Manager and the Project Coordinator.

The quality of each deliverable is in the responsibility of the corresponding work package leader as specified in the DoA, under the supervision of the Project Coordinator. Before a deliverable is submitted to the EC, the quality assurance has to be done in a 2 stage process with responsibilities via representatives: 1 WP Leader, 2 Extern (not included in the Work Package / in some cases not included in the preparation of the deliverable). The procedure for reviewing these deliverables inside the consortium is presented in **Annexes A and B** of this document.

6. Implementation and coding standards

The implementation standard for the Java programming language will follow the guidelines of Eclipse Platform (https://wiki.eclipse.org/Coding_Conventions) regarding coding convention, which in turn are based on the Oracle Java Coding Conventions.

The Javadoc standard will be used to document the relevant code units (packages, classes, methods) following the conventions for writing Java API specification indicated by Oracle (http://www.oracle.com/technetwork/java/javase/documentation/index-142372.html).

The implementation standard for the Python programming language will follow the Google Python Style guide (https://google.github.io/styleguide/pyguide.html).

The requirements for standards and documentation for other programming languages and framework will be added to this document on a case-specific basis.

The OpenReq platform will be under EPL (Eclipse Public License) licence (https://www.eclipse.org/org/documents/epl-v10.html)

7. Project infrastructure for integration, testing and deployment

This section describes the work made within Task 1.5 and its results. In particular, this chapter illustrates the configuration and deployment of the OpenReq project infrastructure and the "Hello OPENREQ" environment which provides a basis for the OpenReq iterative process.

Enabling technologies

This section describes the integration and development technologies that have been selected for the realization of the OpenReq project infrastructure.
The chosen integration technologies are illustrated below:

Maven

Apache Maven [1] is a software project management and comprehension tool for building and managing any Java-based project. This tool automates the activities related to building a software project in order to minimize the risk of human's errors and to make the build process faster. In particular, it simplifies and standardizes the project build process by handling the compilation, distribution, documentation and team collaboration.

The Maven tool is centered around the concept of a project object model (POM). A POM is an XML file which contains information about the project structure and its resources (i.e. source code, test code, project dependencies). While executing a task or goal, Maven looks for the POM in the project directory and gets the needed configuration information to build the project.

The Apache Maven tool is used within the OpenReq integration infrastructure mainly for the following build process activities:

- Compiling source code;
- Packaging compiled code into JAR or WAR files.
- Supporting technology to Jenkins builds.

The Apache Maven tool provides to OpenReq developers a complete build lifecycle framework that automates and standardizes the build of OpenReq projects.

Gradle

Gradle [2] is an open source build management system supporting multi-project and multi-artefact builds. A project using Gradle describes its build using a build.gradle file based on a Domain Specific Language (DSL). A Gradle build consists of one or more projects and each project consists of tasks which represent a piece of work which a build performs (e.g., compile the source code, create a jar, generate Javadoc, publishing some archives to a repository).

In the context of OpenReq, Gradle will be used as build tool alternative to Maven to guide the build lifecycle of the platform components.

Jenkins

Jenkins [3] is an open source automation server that allows continuous integration and delivery of projects, regardless of the platform you are working on. Continuous integration means that the code provided by the developers is integrated as early as possible in order to avoid the problem of finding later issues in the build process lifecycle.

Jenkins enable the continuous integration defining a list of steps to execute (e.g. perform a software build, run a script etc.). The common practice is that whenever developers made changes to the source code a build should be triggered. In particular, it picks up the changes made by developers to source code and triggers a new build. The new build will be available in the Jenkins dashboard. Automatic notification about the build success or failure can also be sent back to the developers. Within the OpenReq integration environment the Jenkins server is configured in order to automate the build process of Maven projects. For build process details see section on Integration Process.

JMeter

jMeter[4] is an open source load and performance testing software. It can be used to simulate a heavy load on a server, group of servers, network or object (e.g. Java objects, Java Servlet, databases etc..) to test its strength or to analyze overall performance under different load types. Within the OpenReq project, JMeter is used to test the performance of the OpenReq capabilities available in the cloud.

Sonarqube

Sonarqube[5] is an open source platform for continuous inspection of code quality. It helps for various tasks and provide reports on duplicated code, coding standards, unit tests, code coverage, code complex, comments, bugs and security vulnerabilities. Within the OpenReq integration environment, Sonarqube is used in combination with Jenkins in order to provide fully automated analysis.

Docker

Docker[6] is an open-source project that automates the deployment of applications inside software containers. It is promoted by the company Docker, Inc.

Docker provides an additional layer of abstraction and automation of operating-system-level virtualization on Windows and Linux. Docker uses the resource isolation features of the Linux kernel.

Within OpenReq Docker will be used as one of the target containers to distribute the OpenReq platform.

JUnit

JUnit[7] is an open source unit test framework which uses annotations to identify methods that specify a test.

A unit test is a piece of code written by a developer that executes a specific functionality in the code to be tested and asserts a certain behaviour or state.

JUnit will be the main framework used for unit testing in the OpenReq development process.

Git Repository

Git[8] is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for software development, but it can be used to keep track of changes in any files.

Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history. In Git, a core assumption is that a change will be merged more often than it is written, as it is passed around to various reviewers. In Git, branches are very lightweight: a branch is only a reference to one commit. With its parental commits, the full branch structure can be constructed.

Being embedded in Tuleap and compatible with Jenkins, it has been chosen as the primary repository for OpenReq source code.

The releases will be organized using the eclipse branching strategy [12].

Tuleap

Tuleap[9] is a web-based, open source (GPLv2 licence) project management system for managing application lifecycles, Agile development and design projects. The platforms allows managers, developers and researchers to collaborate on a common ground using specific and integrated tools (e.g., git repositories for developers, or document management systems for managers). Tuleap supports the creation of trackers for requirements and user stories, as well as code (e.g., bugs), which facilitates the way of working with Scrum. It also supports Kanban for a general overview of the project progress. On top of the integrated trackers and repositories, in OpenReq Tuleap is used to manage the mailing lists, a project-wide wiki, and storing/versioning documents.

Gerrit

Gerrit[10] is a web-based, open source (Apache v2 license) code review management tool which integrates with Git. Gerrit is integrated with Tuleap (and the relative git repositories) to provide the possibility to do code reviews to the contributors working on the OpenReq components.

The chosen development technologies are illustrated below:

- **Eclipse**: an integrated development environment (IDE);
- **Spring Boot**
- **MySQL**
- **Thymeleaf**
- **Bootstrap**

Swagger

Swagger[11] is a powerful open source framework backed by a large ecosystem of tools that helps design, build, document, and consume RESTful APIs.

Swagger uses the OpenAPI Specification to describe API. The goal of OpenAPI is to define a standard, language-agnostic interface to REST APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code.

The ability of APIs to describe their own structure is the the main benefit Swagger: by reading API's structure, the tools can automatically build interactive API documentation. Swagger can also automatically generate client libraries for APIs in many languages and explore other possibilities like automated testing.

To build consistent documentation of the Microservices developed for the OpenReq platform, API will be described using the OpenAPI Specification.

Continuous Integration (Integration Process)

The OpenReq integration process will trigger the builds of the OpenReq Platform using the Jenkins tool hosted in Engineering. In particular, as depicted in Figure 5, the main steps of the integration process are:

- Pick up the latest releases of the OpenReq platform components from the GIT source repositories in Tuleap;
- Build a snapshot of the OpenReq platform. Jenkins will be configured to read the Maven POM files or Gradle build files of the component's projects and to check if there are dependency conflicts between components. If the build fails the developers will be notified;
- Once the build is successful, the resulting artefacts will be deployed on the Cloud Services Infrastructure, if applicable (see section Cloud services Infrastructure), and made available in the Git Repository;
- Junit tests, if any, will be performed.

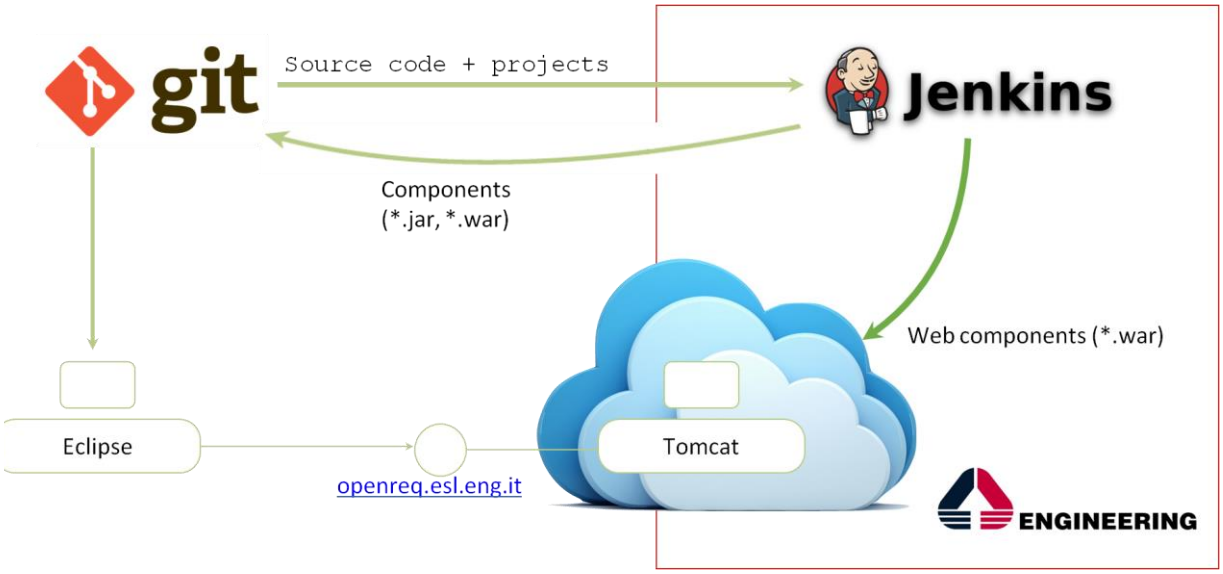

Figure 5: OpenReq integration process.

OpenReq integration process will work on two parallel environments:

- A testing environment where builds will be performed automatically every time a new commit will be performed in the releases branch (see section on Git branching policy). The build outcome will be notified to the developers by email. The outcome will be deployed in the cloud infrastructure and stored in the repository for local installations.
- A stable environment where builds will be performed using commits in the master branch. These builds will be semi-automatic, a build issue management will be performed by the team responsible for integration in Engineering to ensure all deployed releases for this environment are stable. Engineering will take care to perform all tests provided by developers and notify the results.

For both environments SonarQube is setup to perform a static analysis of the quality of the code, to give helpful feedback to the developers.

Git branching policy

The OpenReq components will be available in the Git repositories provided into Tuleap. The branching model used in the Git repositories is the one described at [12]. In particular, a Git repository will be provided for each component, and, according to the chosen branching model, these repositories will hold two main branches:

- *releases*: this branch contains the source code of different releases of the OpenReq components:
  - `<component_name>/releases/<release_name>`
- *master*: this branch contains the source code of the latest stable version of the OpenReq components:
  - `<component_name>/master/<stable_release_version>`

An `openreq-platform` repository will be created to manage the source code for the orchestration of the OpenReq components.

The artifacts generated during the build process (i.e. jar and war files) will be stored in the component specific repository (e.g. `<component_name>/releases/<release_name>/<artifact_name>`).

Cloud services Infrastructure

To host the OpenReq cloud services, a virtual machine (VM) has been set up at our partner ENG premises. This VM is available at the endpoint `openreq.esl.eng.it` and will host the testing and stable environments (see the integration process description) of the OpenReq cloud services.

It has been configured with a CentOs 7 operating system where http and https services are enabled. At the moment a basic configuration has been set up to host webapps (Tomcat 8.5 + MySQL Database) but the environment will be finalized once specific requirements from the developers are defined.

From the point of view of the security of data, the VM is configured to have incremental daily backups and monthly full backups.

Connection is secured and filtered using ENG corporate policies.

"Hello OpenReq" environment

This section describes the running "Hello OpenReq" prototype used to test the set-up of the OpenReq Integration Infrastructure and to provide a basis for the OpenReq Iterative process. The "Hello OpenReq" prototype has been released and its source code is available at the following Git Repository: https://mast-tuleap.informatik.uni-hamburg.de/plugins/git/openreq/hello-openreq.git.

To test the set-up of the Integration Environment provided by Engineering, the Jenkins tool has been configured to pick-up the source code of the "Hello OpenReq" prototype available in the Git repository and to build the project (i.e. a Maven project). The artefact generated after the build

is a Spring Boot web application that can be run in the OpenReq VM hosted by Engineering. In particular, the Jenkins tool has been configured to automatically deploy the generated artefact (named openreq-1.0.jar) into the OpenReq VM and to execute it.

Therefore, once the build process is successful completed, the services provided by the “Hello OpenReq” prototype are available at the following url http://openreq.esl.eng.it/openreq/.

8. Documentation

The documentation of the OpenReq project includes deliverables, technical meetings agenda and minutes, and other documents related to the research of the project. In the following, some standards are defined for each type of document.

Deliverables

Deliverables are the most important type of document in the project. They represent the result of the project work in the eyes of the European Commission, and their positive evaluation is a requirement for the correct evolution of the project.

Deliverables must adhere to the deliverable template that will be made available in Tuleap. The naming convention for a deliverable must be:

Dm.n - Name_vr.s

where *Dm.n* identifies the deliverable as declared in the DoA, then *Name* is the name of the deliverable (which can be a short name if it is too long) and then *vr.s* represents the version (*vr* is the major number and *s* the minor number), only in cases where the deliverable is updated during the project. An example for this deliverable would be "D1.4 – Project standards and infrastructure document".

Some of the deliverables could be updated throughout the project. In case of these deliverables, the major number of the version should be increased for each update. For example, D1.3 should be submitted in M6 and updated over the course of the project whenever significant changes arise. In this case, the first version delivered in M6 should be named as "D1.3 – Data Management Plan", and future delivered versions will be named as "D1.3 – Data Management Plan_v1.0", "D1.3 – Data Management Plan_v2.0", etc. If there are some changes in between delivered versions, the version minor number is the one to be increased. All the major versions will be stored in Tuleap.

The workflow for the review process of deliverables is detailed in annexes A and B.

Technical meeting agenda and minutes

Technical meetings will be organised when needed to ensure the success of the project activities.

The chairperson of a Consortium Body shall produce written minutes of each meeting which shall be the formal record of all decisions taken. S/he shall send the draft minutes to all Partners within 10 calendar days of the meeting. The minutes shall be considered as accepted if, within 5 calendar days from sending them, no Partner has sent an objection in writing to the chairperson with respect to the accuracy of the draft of the minutes.

Minutes will clearly identify Action Points with deadline and responsible. Also, the minutes will include the list of attendees. Agenda and final minutes will be stored in Tuleap. Their naming conventions are:

- For agenda: *date-TechMeeting_Agenda*, where *date* is the date in format *yyyymmdd* (filling with '0' for days or months lesser than 10). Preliminary versions of the agenda will have suffix *"_vm"*.

- For minutes: *date-TechMeeting_Minutes*. Preliminary versions of the minutes will have suffix *"_vm"*. The list of disagreements of partners to preliminary versions of the minutes will be stored in a file *date-TechMeeting-AmendmentRequests_vm*, with the identification of the person that raised the amendment request and the quotation of that request.

Other Documents

The rest of documents are recommended to be compliant to the following nomenclature:

Name - partner_vr.s

where *Name* is a significant name which provides information about the nature of the document, *partner* is the name of the partner that created this version, and *vr.s* represents the version. An example of valid name could be "Requirements Patterns SLR Protocol – UPC_v1.2". Please note that this convention will not be enforced. All the major versions of the document will be stored in Tuleap.

9. When to use which tool

For Development

The main tools for collaborative development used through the development process of the OpenReq components are: i) a version control system (i.e., a Git server accessible via Tuleap), ii) a continuous integration server (Jenkins) to execute tests and eventually merge the modifications, iii) a bug tracking system (inside Tuleap) integrated with the current trackers available in Tuleap.

Usage:

i) When working on a component the developer will checkout the relative repository from the git server (or update their local copy after someone else applied a modification), and apply his/her own modification. The modification are then sent back to the server.

ii) Before the accepting the modification, the review server will show what has been modified. In this occasion, other developers (i.e., not the one who authored the modification) will review the code for quality issues and decide whether accept the modification or ask the authors to address the issue.

iii) Once the modification are accepted, a set of regression testing will be run to check whether or not the modification caused any other parts of the system to fail. This step is done automatically by the continuous integration server, however, in case of failing manual action should be taken.

iv) When a issue or bug or missing feature is identified (e.g., after the previous steps) a ticket should be opened on the tracker. The ticket will describe the issue (or request) the motivations and the context. The request will be assigned to one developer who will address it (e.g., following the same procedure above).

For Reporting

- Google Docs: a set of shared Google Documents is the main way for collaborating on writing drafts of the reports. The responsible for the report creates and shares a link to the document with the collaborators; this also facilitates internal revisions. Once a draft is completed and revised, it is “freezed” in the Tuleap Document Management System
- Documents management system (Tuleap): Tuleap provides storage of files organized in a hierarchy through folders and subfolders. Although the system supports basic versioning functionalities, it makes collaboration cumbersome has a file needs to be downloaded, edited and re-uploaded on the platform each time. A unique id is assigned to each document so that it can be referenced throughout the system (e.g., in a tracker item, a wiki page, or a code commit). The final version of the reports will be stored here (see Annex C).
- Wiki (Tuleap): The wiki hosted on Tuleap are used for internal reports regarding OpenReq that are likely to change (e.g., a wiki page describing the project members). Wiki pages can be references throughout the Tuleap system by their id. However, a relevant report will be in any case saved to the Document Management System.

For Communication

- Mailing-Lists/Mail: In addition to traditional mail exchanges, one-to-many communication will be made possible using mailing lists. The mailing lists are hosted and managed through a MailMan instance integrated with Tuleap. For an overview of the available mailing lists, their purpose and members see Annex C.
- Slack: the role of the Slack channel openreq.slack.com is to enable quick communication and collaboration between the project members. However, this is not the main communication channel.

10. Evaluation of technical standards

This refers to recommenders, operating systems, programming languages and open source components.

Responsible work package leaders are in charge of the evaluation of conformance to the technical standards for the corresponding work package.

We will use and extend state of the art recommendation approaches (collaborative filtering, content-based recommendation, group recommendation approaches).

Java version 6+ will be the main programming language (see section “Implementation and Coding Standards”) and thus there are no further specifics about the underlying operating system. For further details regarding the installed operating systems of the virtual machines please refer to section Cloud services Infrastructure.

OpenReq uses the state of the art technology “Spring Boot Apps” (Running REST Services in the backend), “Thymeleaf” (Presenting User interfaces for the end users) in combination with “Bootstrap” (CSS styling framework) for the prototype which supports basic functionality of OpenReq. In case the trial partners need different user interfaces those specific interfaces use the REST services of OpenReq.

Annex A. Workflow of deliverable review

Each and every deliverable in the OpenReq project will undergo through a review process. Google Docs file will be used for the writing of the deliverables, as it allows to easily track changes and comments. Given a deliverable with the following characteristics:

- to be delivered at a *due date*,
- managed by a *lead beneficiary*,
- produced by a task that starts at *initial date*,

the reviewing process consists of the workflow explained below.

PHASE I. Preparation – 1st month

Step 1. The lead beneficiary takes care of setting up the Google Docs space and the calendar for the deliverable production in Tuleap. Deadline: One week after the *initial date*.

Step 2. The *lead beneficiary representative* communicates by email to the project coordinator (PC), scientific manager (SM) and dissemination manager (DM), who is the *person in charge* of this deliverable. This person will serve as contact point for the rest of the deliverable production. Deadline: Two weeks after the *initial date.*

Step 3. The PC, previous consultation with the SM and DM, will select two *reviewing partners* (different than the lead beneficiary) to take care of the review. He will notify this decision to the representatives of such partners and to the *person in charge*. Deadline: Three weeks after the *initial date*.

Step 4. The two *reviewing partner* representatives will nominate a project member from their organizations as *deliverable reviewers*, and will communicate these names to the PC, SM and DM. Deadline: One month after the *initial date*.

PHASE II. Writing – from the 2nd month to 1 month before the deadline

Step 5. The *person in charge* will e-mail a notification with a link to the Google Docs file to the PC and *deliverable reviewers* for a first check (already using the project deliverable template). Ideally, each section may contain a short (2-3 lines) description of the intended contents. Deadline: Two months before the *due date*.

*Step 6.*The PC and the *deliverable reviewers* may provide quick and short feedback to this index. Deadline: One week after Step 5.

Step 7. The *person in charge* will implement the feedback of the index and will e-mail a notification containing a link to the Google Docs file to the people involved in the production of the deliverable. Deadline: Two working days after Step 6.

Step 8. The *person in charge* will e-mail a notification containing a link to the Google Docs file to the *deliverable reviewers* when there is available a complete first draft of the deliverable. In the file, reviewers will track changes and provide comments. Deadline: Four weeks before the *due date*.

PHASE III. Review – last four weeks

Step 9. The two *deliverable reviewers* will complete their review according to the instructions given in Annex B. Once this is finished, the will notify the *person in charge* with copy to the PC. Deadline: Two weeks before the *due date*.

*Step 10.*The *person in charge* will analyse the *deliverable reviewers*' comments. Significant disagreements will be explicitly communicated to them and the PC. Deadline: One working day after Step 8.

*Step 11.*The *person in charge*, and if needed together with the people involved in the production of the deliverable, will implement the *deliverable reviewers*' comments. In parallel, any discussion on disagreements will be held and progressively solved. Deadline: One week before the *due date*.

PHASE IV. Finalization – due date

*Step 11.*The *lead partner* will upload the final version of the deliverable in the participants' portal (in pdf) and also in Tuleap (in MS Word and pdf) with notification to the PC. Deadline: The *due date*. See Note 5 below for additional information.

PHASE V. Evolution

Step 12. For subsequent versions of the deliverable, the *person in charge* and the two *deliverable reviewers* will be kept unless causes of force majeure. Steps 5 to 11 will be repeated for each such version.

Note 1: the appointment of reviewing partners (Step 3) will be proposed considering a fair distribution of reviewing workload along the project lifetime and also promoting diversity in the assignment of reviewing partners to deliverables produced by one partner.

Note 2: these instructions are separately available in Tuleap and updated with concrete reference to appropriate names of spaces, folders and files. In Tuleap, it will be possible to find the details of deliverables, including their concrete dates after applying the workflow steps and people involved in their production and reviewing.

Note 3: for deliverables at M12 and M24, considering Xmas season, all dates will be moved one week before the usual plan, so the deliverable is ready before Christmas.

Note 4: for deliverables produced by tasks started before the writing of this annex, Steps 1 to 4 redefined *initial date* by *1 month after Tuleap is ready*.

Note 5: when a deliverable has several versions, after the second version (v1.0) is produced, the file uploaded in the participants' portal will be the pdf file of the last version. Instead, in Tuleap, all the versions will be kept.

Annex B. Deliverable review: instructions for reviewers

As per Annex A, every *deliverable reviewer* will receive from the *person in charge* a link to a Google Doc file at two different moments:

- Two months before the deadline, an index of the deliverable.
- Four weeks before the deadline, a complete draft of the deliverable.

Feedback to the deliverable index

This feedback is optional and free format, by commenting and annotating the document submitted by the *person in charge*. If no feedback is given in one week, it is assumed that the *deliverable reviewer* agrees with the contents. This also applies to the PC, who also receives such an index.

Feedback to the deliverable draft

Roughly speaking, the draft review is expected to go along three directions:

1. **Content**. This is the most important part. Use comments in the Google Docs file for small issues, but do not hesitate to attach an evaluation form to your review (free format). Be practical and effective: remember that the proposed changes need to be implemented in one week. One particular issue to check is the length of the deliverable: although it is difficult to give a concrete number, short deliverables are preferred to long ones. In case of need for details, we recommend to use annexes, so that the reading of the main body does not get tedious.
2. **Style**. For typos and clear grammatical errors, directly modify the text using track of changes in the Google Docs file. For suggestions, use again comments.
3. **Formatting**. Since the document gives an image of the project to the reviewers and the outside (for public deliverables), it is important to pay attention to formatting details. As reviewer, you need to check that the *project deliverable template* has been strictly applied. Check also the first pages, which is a typical source of errors since there is a lot of information. Ensure that the fixed parts (section 1, conclusions, references, etc.) are kept. Double-check with the DoA: type of deliverable, deadlines stated, names of deliverable, tasks and work packages, etc.

Note 1: very important, reviewers must always use track of changes when modifying the submitted documents.

Note 2: reviewers are kindly invited to interact with the deliverable person in charge during the review process to solve doubts, to sort out major issues or in general, speed up later processing and ensure the deliverable quality. If agreed between both parts, staged reviews (e.g., by sections) may be considered, again for the sake of speeding up the process.

Note 3: for Demonstrator deliverables, further instructions will be given in later versions of this annex including specific questions to issues like installation instructions, user manual, licensing information, etc.

ANNEX C - Tuleap

Mailing lists

The lists currently available are as follows (the mail address associated with each list is *listName*@mast-tuleap.informatik.uni-hamburg.de)

List name	Purpose
OpenReq-team	For reaching *everyone* actively involved in the project
OpenReq-devel	For software development issues and support activities. Can also include external members, due to open source, open call, students etc.
OpenReq-trials	For coordination among the trial partners
OpenReq-dissemination	To notify about new publication using the reporting sheet provided in Tuleap
OpenReq-administration	For legal, financial and progress reporting data.

Documents

There are two different areas that can serve to store and access documents.

For final documents (e.g., documents that are not likely to change, please use the *Documents* page.

The Documents structure looks as follows:

- DoA: *for storing items specifically related to DoA tasks)*
  - WPX: documents regarding the particular WP
    - *Deliverables*: Ad-hoc folder for storing only the deliverables required by WPX with specific subfolders
      - *DX.Y*: a folder for Deliverable X.Y where at least three documents will be stored
        - the draft document to be sent to internal quality control
        - the internal quality control report
        - the final document to be submitted to the EC.
- Meetings: *for storing meeting related docs (minutes, agenda, etc)*

    - Year XXX*: documents related to meeting taking place during year XXX*
        - Board meetings*: only for board meetings taking place in year XXX*
        - Project review: *only for project review meetings taking place in year XXX*
        - Interview (or any other kind of relevant meetings)

- Evergreens: basic *documents that will be used over and over during the project*
    - Presentations: *slides (e.g., from KoM presentations) that can be reused (for example during the interviews)*
    - Templates: materials for needed when creating new documents/brochure/presentations

References

[1] Apache Maven, https://maven.apache.org/

[2] Gradle, https://gradle.org/

[3] Jenkins, https://jenkins.io/

[4] JMeter, http://jmeter.apache.org/

[5] Sonarqube, https://www.sonarqube.org/

[6] Docker, https://www.docker.com/

[7] JUnit, http://junit.org/junit4/

[8] Git, https://git-scm.com/

[9] Tuleap, https://www.tuleap.org/

[10] Gerrit, https://www.gerritcodereview.com/

[11] Swagger, http://swagger.io/

[12] Git branching policy,
https://wiki.eclipse.org/Scout/Contribution_Guidelines#Git_Branching_Policy