D1.3 Software Quality and Architecture Plan

Author(s): Víctor Rodríguez Doncel, Khalil Ahmed, Philipp Cimiano, John McCrae, Maria Pia di Buono

Date: 28.06.2019

This project received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825182. The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Union.

H2020-ICT-29b
Grant Agreement No. 825182
Prêt-à-LLOD - Ready-to-use Multilingual Linked Language Data for Knowledge Services across Sectors

D1.3 Software Quality and Architecture Plan

Deliverable Number: D1.3
Dissemination Level:
Delivery Date: 30/06/2019
Version:
Author(s):

Document History

Version Date	Changes	Authors
11/03/2019	Started initial draft	UNIBI
29/03/2019	Completed initial draft and first peer review	UNIBI, NUIG
15/04/2019	Annexes added	UNIBI
18/06/2019	Final peer review	All partners

Table of Contents

D1.3 Software Quality and Architecture Plan

1. Document Scope and Overview

The development of novel tools for transforming and linking datasets, and the subsequent application of these to both data and metadata allow to provide multi-portal access to heterogeneous data repositories. The merits of these new language technologies can be found not only in being 'ready-to-use' but also underpinned by data value chains applicable to a wide-range of sectors and applications. The semantic-based integration of language resources and language technologies relies on the capability of being combined into complex pipelines to offer sustainable data services.

Achieving interoperability and usability in a multi-developer complex system requires a structured and quality approach. This guideline encourages such an approach in the development and management of Prêt-à-LLOD software in order to contribute to system life-cycle, improving its performance, stability and resilience. Other important benefits include the decrease of time and resources required for system maintenance and support, and the enhancement of coordination and effectiveness of the collaboration.

This guideline is intended to be used by all the stakeholders involved in the design and development of Prêt-à-LLOD software, with its primary users being who develop and test the technical components.

Figure 1 provides a summary of Prêt-à-LLOD technical components and Table 1 presents a summary of stakeholder involvement into each task of Prêt-à-LLOD software development.

Figure 1: Overview of the Prêt-à-LLOD technical components ("toolkit") and their interaction with the data value chain

Table 1 - Stakeholder involvement

Prêt-à-LLOD task	Stakeholder
T3.1 Transforming language resources and language data	GU, UZAR, UPM, UNIBI, SEM
T3.2 Linking conceptual and lexical data for language services	UZAR, NUIG, UPM, UNIBI, GU, DFKI, OUP, SWC
T3.3 Workflows for Portable and Scalable Semantic Language Services	NUIG, UZAR, UNIBI, GU, SEM
T5.2 Policy-driven language resource discovery and access	UPM, NUIG, UZAR, DFKI
T5.3 Repositories for Resources and Metadata	NUIG, UZAR UPM, GU,DFKI, DLX

It is worth ensuring that during the development process, software requirements are considered for each stakeholder and task. In order to maximise outcomes, partners should carry out all the activities by taking into account interactions between people, technology, and organizational aspects.
Such interactions, which guarantee outcome enhancement, can be supported addressing quality criteria and management aspects, together with a risk assessment, that need to be considered within the project.

The scope of this document is to provide a set of procedures and guidelines for Prêt-à-LLOD partners and developers to adhere to in order to develop a product usable, sustainable and maintainable.
Three main aspects in software development are considered: planning and management strategies, including risk management, a description of software architecture, and quality assessment criteria.

In order to fully document all the aspects of software architecture and quality, this guideline contains the following subsections:

- **Section 1**: introduces guideline scope and structure;
- **Section 2:** describes agile methodology for planning and management in software development, and a methodology to prevent and reduce related risks;
- **Section 3:** presents the software architecture providing different view, namely Logical, Process, Component, Deployment;
- **Section 4:** describes both Usability, and Sustainability and Maintainability criteria used to assess software quality.

2. Planning and Management

With reference to **project planning and management**, we introduce an agile methodology to provide techniques suitable for reaching project's goals more effectively. Agile methodology, implemented by using Scrum framework, allows for changes and modifications during the project process in case some components cannot be delivered on time. Scrum[1], developed by Jeff Sutherland in 1993 and based on iterative and incremental practises, focuses on "strategy, a flexible holistic product development where the development team

[1] https://www.scrumguides.org/

worked as a unit to achieve common goals" as opposed to "traditional approaches, a sequence" (Falls, 2004).
Scrum framework is supported by three pillars: transparency, inspection, and adaptation.
To the intent of optimizing the software development process within Prêt-à-LLOD project, in this guideline we will focus on inspection and adaptation pillars.
Inspection refers to the need of checking frequently the status of artifacts and progress in order to detect the possible variances towards the planned activities and deliverables.
When such deviances are revealed the process has to be adjusted through **adaptation** as soon as possible to minimize further deviations.
Scrum uses events to describe time-boxed activities, with a duration fixed at the beginning.
All events are represented by a **Sprint**, a time-box of one month or less, during which a product increment is created.
According to Scrum prescriptions, four formal events are necessary for inspection and adaptation: Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective.
Before defining the Sprint Planning, it is worth defining the **Product Backlog** (PB), which is a sorted list of all the products needed and the only source of product demand changes. The Product Backlog list has to be continuously improved in order to be aligned with changes in the product and development environment. **Product Backlog Items** (PBIs) should be sorted by value, risk, priority and necessity, in a sequence of highest to lowest priority. Product Backlog items, and associated workloads, are detailed into **Sprint Backlog and Planning** according to their goals.
Within the development process of Prêt-à-LLOD software, we suggest to produce a PB list for each product developed in order to create a Sprint Backlog for each WP. While the coordinator is in charge for the general management and coordination of all WP Sprints that will lead to the final software development, each WP leader is the owner of their own software deliverable and related Sprint Backlog and Planning.
Scrum methodology establishes a daily meeting, i.e., **Daily Scrum**, a time-boxed event for the development team. It is advisable to leave the management of this type of meeting at each partner own discretion, as **internal meeting** for each development group. Considering activities and meetings within the project, it is suggested to introduce a fortnightly **Sprint Review** meeting among all partners involved into a WP in order to proceed with a monitoring phase for each WP Sprint. The main Sprint Review aim is a review of Product Backlog items for the next sprint and a possible overall adjustment of Product Backlog, if needed. A general review and report for each WP Sprint will be presented during the management meeting with all WP leaders.
The **Sprint Retrospective** represents the opportunity to revise the Sprint Planning to improve and adapt work processes, making more effective next sprints. Each WP leader manage their own Sprint Retrospective, while all Prêt-à-LLOD partners are involved into a monthly retrospective session to identify overall improvements for the whole project development.

To ensure a simpler collaboration among partners, ad-hoc tools for software project management, e.g., Trello, will be adopted.

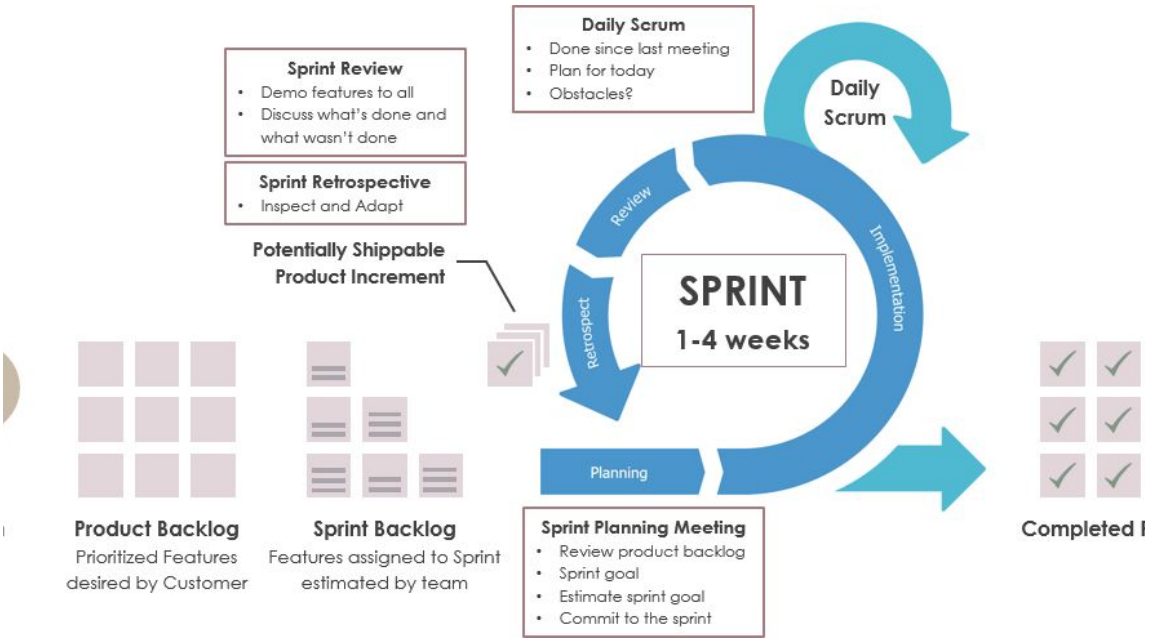

Figure 2: Overview of Scrum Construction lifecycle[2]

2.1 Project Risk Management

Within a project it is worth to quantify risks, evaluate the probability of its occurrence, and its potential impact (Huang and Han, 2008). Since the Scrum life cycle is divided into several iterative Sprints, usually two to four weeks long, it should be easier monitoring the product being developed and identifying impediments or risks (Tavares et al., 2017).
Furthermore, the use of Sprints also supports risk management, as it limits risk to one calendar month of cost (Schwaber and Sutherland, 2013).

In compliance with an incremental development approach, a best practice to mitigate project risks suggests that all the project partners decide what features have to be prioritized to be delivered, so that the highest priority features are delivered first. Indeed, prioritisation establishes a forced ranking of features (or deliverables) to guarantee that the highest value work is completed first. Prioritisation should happen continuously throughout the project to enable changing priorities and embed new information into the development process.
Among different existing prioritization techniques, we suggest to use the **MoSCoW Method**, which is based on four criteria: **M**ust haves, **S**hould haves, **C**ould haves, and **W**on't haves. These four criteria should drive the process of setting requirements by order of priority, starting from the Product Backlog.
If, for any reason, the overall project schedule turns out to last longer than expected, it is recommended to prune or delay some of the lower priority features in order to meet the schedule.

[2] https://www.visual-paradigm.com/scrum/what-is-product-backlog-in-scrum/

3. Architectural Representation and Methods

According to IEEE Standard Glossary of Software Engineering Terminology[3], architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution. Therefore, the structure of that system, comprising all of its software elements, the properties of these elements, and the relationships existing among standard them are described in the architectural representation for a system.

Even if the conclusive software architecture is ultimately expressed in the executable code, there exist representation models suitable to visualize and represent such an architecture. These representations models are used in order to describe the public interfaces of software elements, how these elements are used, relate to each other and interact. There is a clear distinction between low and high level specifications for software architecture. The former refers to internal implementation details of software elements, e.g., algorithms, and are generally omitted from the architectural representation, the latter describes the general and more relevant specifications of a system from the perspective of a particular set of concerns. Different sets of concerns define different types of view, namely different representations of different concerns of multiple subjects involved in or affected by the software development. The main goal of architectural views is dealing with software complexity by focussing on a small number of constituents.

Several classifications of architectural views have been proposed. In this guideline we refer to the 4+1 View Model (Kruchten, 1995), which describes software architecture using four basic views, i.e., logical, process, component, development. Together with these views, there is an additional view, called Scenarios, which illustrates the four basic views, proposing selected usage scenarios from the perspective of a use case view to demonstrate the software architecture (Figure 3).

[3] IEEE 1417-2000 standard https://standards.ieee.org/standard/1471-2000.html

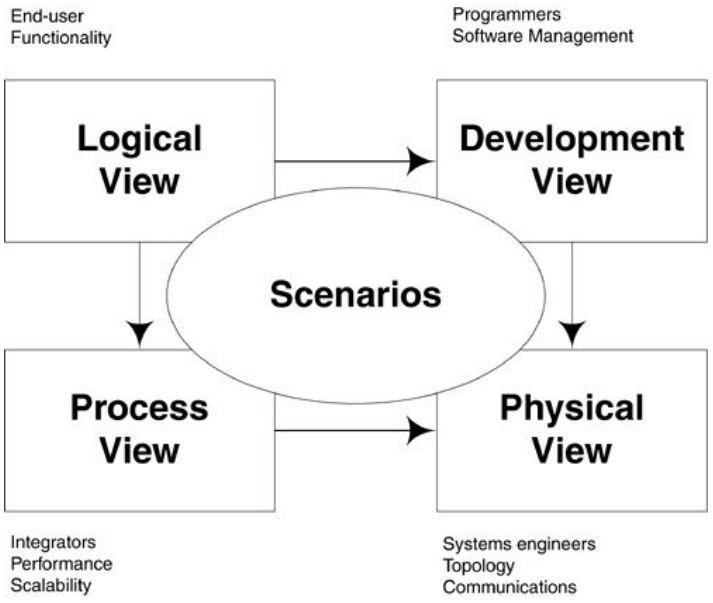

Figure 3: Illustration of the 4+1 Architectural View Model (Kruchten, 1995)

A description of these views is provided in the following sections to represent Prêt-à-LLOD software architecture.

3.1 Logical View

The Logical view describes software functionalities as they are provided to end-users, modeling the translation of software requirements into functional aspects. The main goal is providing a description about how specific functionalities are satisfied by architectural abstractions.

As shown in Figure 1, Prêt-à-LLOD software is structured into several distinct layers, which are in turn distributed through containerization technologies, i.e., Docker.

The high-level Prêt-à-LLOD tiers are:

- **Prêt-à-LLOD Discovery.** This tier provides tools to share and discover resources extending the functionality of current metadata repositories for linguistic data. The main functionality is represented by a data search tool, covering major dataset sources (EUDAT, Datahub) and language resource repositories (ELRA, LDC, Metashare, CLARIN, ELG), which allows a faster and better access to resources.
- **Prêt-à-LLOD Transform.** It focuses on improving the simplicity and transparency of the transformation process of multilingual language resources into and between LLOD representations
- **Prêt-à-LLOD Data Manager.** This functionality provides methodologies for describing the licenses of data and services with the ability to retrieve and automatically process provenance and licensing information, and mechanisms for representing and dealing with trust and confidence, in order to enable smart contracts and ease exploitation strategies. This contributes to a better and automated handling of legal constraints and also a search by licenses as part of Prêt-à-LLOD Discovery.
- **Prêt-à-LLOD Linking.** It allows cross-lingual integration of datasets coming from different sources and covering the 24 European official languages, contributing to

future generations of multilingual solutions being able to process these languages and operate without cross-border limitations. Prêt-à-LLOD will demonstrate a qualitative increase in multilingual data integration and data reuse across sectors relevant to the Digital Single Market. In order to demonstrate this, the project plan includes four pilots addressing different mission-oriented challenges .

These tiers are integrated into a further tier which allows to improve portability and scalability of language technology services:

- **Prêt-à-LLOD Workflow.** This last tier improves the interoperability and semi-automatic integration of language services in the cloud through exchangeable NLP components achieving a higher level of portability than previously known, and allowing the development of multilingual solutions that remain useful and re-usable in multiple sectors.

Further specifications of key features and technical details will be described at the end of the project within the Final Software Quality Evaluation (D.1.2.2).

3.2 Process View

This view allows to capture the concurrency and synchronization aspects related to software design. The Process View can be considered as a set of independently executing logical processes. A process is a grouping of tasks that form an executable unit and can be distinguished into major tasks, i.e., architectural elements that can be uniquely addressed, and minor tasks, i.e., additional tasks introduced locally for implementation reasons.

3.3 Component View

Such a view contributes to describe a software in its physical layers or components, as opposed to the logical layers in the Logical View, and define communication lines among layers.
The main components are:

- Vocabularies and Language Resources
- Lexical Data
- LLOD Datasets
- NLP Components and Language Technologies.

Further specifications of key features and technical details will be documented during the development phase through the duration of the project and described within the Final Software Quality Evaluation (D.1.2.2).

3.4 Development View

The Development View describes the software module organization in the development environment. The software is packaged in small chunks - program libraries or subsystems - organized in a hierarchy of layers, each layer providing a narrow and well-defined interface

to the layers above it (Kruchten, 1995). This view is represented by module and subsystem diagrams, showing input and output relationships. It is worth taking into account internal requirements related to the ease of development, software management, reuse and constraints imposed by the toolset

3.5 Scenarios

This view encompasses all the previous views, from a use case perspective. In Prêt-à-LLOD project three different use cases are considered.

Technology Companies: Prêt-à-LLOD will allow this market-leading dictionary data to become much more interlinked and manageable, greatly improving its application for the use cases described. The improved efficiency will also allow to focus on the creation and development of world-leading content, targeted towards speakers of languages worldwide, especially under-resourced languages through the Oxford Global Language project[4].

Pharma: It consists in gathering evidence for the effectiveness and safety of a drug product, outside of the controlled settings of a clinical trial, in order to provide a proof of the added value of a drug in a large population. The solutions developed to address the mission-oriented technical challenges above will be adopted by Semalytix to develop multilingual applications that can produce real world evidence by analysing multilingual data including patient forums, social media, electronic healthcare records and CMS data as provided by IQVIA 19.

Government Services: In Prêt-à-LLOD, we will address the dual challenges of (i) providing cross-border public services, which is essential to achieve an inclusive Digital Single Market, and (ii) the portability of public services and knowledge sharing across jurisdictions for improved collaboration and cost savings. We will deploy the Prêt-à-LLOD technology stack to support the provision of multilingual statistical data across a number of member states as well as the rapid development of integrated urban solutions.

These three scenarios will be implemented by means of four pilots, as specified in the Business Pilot Specification (D4.1) and described within the Final Software Quality Evaluation (D.1.2.2).

4. Quality assessment Criteria

Quality is assessed with reference to two aspects: general quality criteria of the deliverable/software produced and a more specific adhesion to Linked Open Data (LOD) prescriptions.

Concerning general quality criteria, we refer to Criteria-based Software Evaluation Guide (Jackson et al., 2011) by the Software Sustainability Institute (SSI), which provides a set of assessment criteria to be used to evaluate a software, and to the guidelines for software quality & sustainability for CLARIAH[5] (van Gompel et al., 2016). The proposed assessment criteria are split into sub-criteria, referring to the main aspects useful in software development: usability, and sustainability and maintainability (Table 2).

[4] https://www.oxforddictionaries.com/ogl
[5] https://github.com/CLARIAH/software-quality-guidelines

Table 2 - Assessment Criteria by Jackson, Crouch & Baxter (2011)

Criterion	Sub-criterion	Notes – to what extent is/does the software…
Usability	Understandability	Easily understood?
	Documentation	Comprehensive, appropriate, well-structured user documentation?
	Buildability	Straightforward to build on a supported system?
	Installability	Straightforward to install on a supported system?
	Learnability	Easy to learn how to use its functions?
Sustainability and maintainability	Identity	Project/software identity is clear and unique?
	Copyright	Easy to see who owns the project/software?
	Licensing	Adoption of appropriate license?
	Governance	Easy to understand how the project is run and the development of the software managed?
	Community	Evidence of current/future community?
	Accessibility	Evidence of current/future ability to download?
	Testability	Easy to test correctness of source code?
	Portability	Usable on multiple platforms?
	Supportability	Evidence of current/future developer support?
	Analysability	Easy to understand at the source level?
	Changeability	Easy to modify and contribute changes to developers?
	Evolvability	Evidence of current/future development?
	Interoperability	Interoperable with other required/related software?

The software must be stored in a version control system (VCS). The source code and related resources must be published. Prêt-à-LLOD has a dedicated repository on Github (https://gitlab.insight-centre.org/pret-a-llod), it is recommended to host the VCS here.
Since developing semantic web technologies requires the compliance with a certain set of attributes for the published data, we take into account LOD prescriptions to assess such an aspect in our software quality assessment.

4.1 Usability

In this section, we refer to all quality assessment sub-criteria for usability, namely understandability, documentation, buildability, installability, learnability.

Understandability
With reference to understandability, it is necessary:

1. Providing a clear and concise high-level description about the software. These information have to be stored in both the README files as well as in the project website.
2. Specifying the intended users for the software. Where appropriate, it would be advisable offering multiple interfaces (i.e., command-line interface (CLI), graphical user interface (GUI), web-user interface (WUI))
3. Clarifying how the software works through a high-level description, links to publications, and a schema offering an architectural overview.

4. Motivating the software
5. Being clear about the stage of software development.
6. Being clear about whether the software is actively supported, and if so until when.

Documentation

Documentation refers to a set of documents available for the software. Those documents may consist of different type of documentation for different audiences and may include published papers.

A bare level of documentation includes a README which provides a high-level overview of the software, makes use of adequate examples for the interface described. For CLIs it is appropriate providing examples of invocation, input, and output. GUI examples require screenshots or screencasts. For APIs it is necessary providing source code examples of usage.

Furthermore documentation should include information about troubleshooting and a frequently asked question (FAQ) section.

In order to provide a clear documentation it is recommended using step-by-step and task-oriented instructions and covering the entire software, including advanced features, as well as documenting all the multiple tools.

Different groups of users require different documentation due to their different level of expertise.

Documentation should be linked from the project website and it should be under version control like the source code, alongside the code itself.

Buildability and Installability

Buildability and installability criteria concern the pre-requisites for building and/or installing the software on a build and/or target platform.

To meet this criterion, instructions for building/installing the software have been provided on the Web site and in source distributions. Furthermore, buildability and stability of the code-base should be supported by the use of continuous integration services oriented to build and test projects hosted on GitHub, e.g., AppVeyor, Travis.

All third-party dependencies that are not bundled, mandatory or optional, along with Web addresses, suitable versions, licenses have to be currently available and listed in source distributions and on the Web site. Where appropriate, a dependency management can be used.

All source and binary distributions should be provided with a README file with project name, Web site, how to get help, version, date, license and copyright (or where to find these information), location of entry point into user doc.

Learnability

This criterion focuses on how straightforward it is learning to use the software.

In order to abide by the learnability criterion, it is necessary providing a *Getting started* guide to outline a basic and practical example about how to quickly started with the software.

Instructions should be provided for at least basic use cases.

The interface should include help options. For CLIs describing usage and all options through *-h/--help*. For GUIs using tooltips/hints for their widgets.

For programming library, providing API documentation. For software as Web services, providing a specification about the Web API together with a description of API endpoints, operations parameters and return values. API documentation should be auto-generated from comments in the source code and include a description of class/methods with parameters, expected results and exceptions.
Software configuration options and their effect should be clearly documented.

4.2 Sustainability and Maintainability

Sustainability and maintainability criteria refer to aspects related to both communication and copyright/licensing elements, and enduring software development processes.

Identity and Copyright & Licensing
With reference to the first group of sub-criteria it is worth stressing the need of a clear and unique identity, and a distinct name of the project/software, together with the respect of existing trade-marks.
Copyright and licensing should be clearly stated on the Web site, jointly with authorship and funding acknowledgment. Each source code file should present a copyright statement and a license header.

Community
The Community sub-criterion refers to the presence of an active user community for the software. It is recommended stating in the Website the number of users/developers/members, success stories, important partners or collaborators, and the list of the project's publications and third-part publications that cite the software. Users should be required to cite a boilerplate citation if publishing papers based on results derived from the software.

Accessibility
To ensure an enduring software, source code must always kept by under version control to allow collaboration and maintaining a version history. The version control repository should be public (read only) to preserve the spirit of open source and as well as scientific methods of transparency, peer review, and reproducibility. Each release of the software should be clearly marked with the version number, according to a consistent scheme, and identifiable tags that marks the state of the repository at the time of his release.

Testability
The software should have unit tests to automatically test individual units of the source code and verify the data and logic flow, and integration tests to combine individual parts of modules and see how they function as a group.

Portability
Portability across platforms and browsers should be guaranteed, as well as portable deployment across machines.
Prêt-à-LLOD relies on Docker, a technology which defines a format for bundling an application and all its dependencies into a single container. Such a container can be

transferred to any Docker-enabled machines. Docker guarantees that the execution environment is the same in the development, testing, and production.

Supportability
It should be clear to what extent the product will be supported currently and in the future. It is strongly recommended providing a public issue/bug tracker. This tracker allows to post bugs as well as features request.

Analysability
The sub-criterion of analysability deals with aspects related to the source code. It is necessary structuring the source code into multiple modules/packages, respecting a clear relationship to the architecture or design of the software.
The source code should contain comments explaining what major blocks do.
It is advisable that the comments use a mark-up that allows them to be used directly as the source for the generation of the API reference documentation.
There should be recommended coding standards, consistent with the larger community of generic coding standards for the programming language, to which contributors should adhere to.

Changeability & Evolvability
A project, open to outside contributions, should have guidelines, publicly available, for contributors.
Software still under active develop should present a roadmap, that may be explicit or implicit in the issue tracker through the assignment of milestones.
It is necessary providing information about when the software is no longer actively developed.

Interoperability
The software should meet appropriate open standards in order to ensure its interoperability with required and optional third-part components.

4.3 Linked Open Data Quality

In Prêt-à-LLOD project, we assess LOD quality, referring to LOD data quality dimensions presented by Zaveri et al. (2016). Starting from the classification introduced by Wang & Strong (1996), Zaveri et al. identify six groups of main dimensions, formalizing and adapting their definition to the LOD context[6].

Accessibility
All the dimensions belonging to this group refer to aspects related to data access and retrieval in order to obtain either the entire or some portion of the data for a particular use case. Five sub-dimensions are part of this group, namely availability, licensing, interlinking, security, performance.

[6] In this guideline, we report the description of the six main dimensions by Zaveri et al. (2016), referring to Annex III in this document for the assessment of sub-dimensions within each group.

Intrinsic Dimensions
Intrinsic dimensions, independent of the user's context, focus on whether information correctly and compactly represents the real world data and whether the information is logically consistent in itself. Three sub-dimensions belong to this group, which are accuracy, consistency, conciseness.

Trust Dimensions
This group holds the sub-dimensions which refer to perceived trustworthiness of the dataset, i.e., reputation, believability, verifiability, objectivity.

Dataset Dynamicity Dimensions
Dynamicity dimensions refer to the capability of datasets to preserve their freshness over time, and over time for a specific task, together with the capability of enduring over time. Three sub-dimensions capture these aspects, namely currency, volatility, timeliness.

Contextual Dimensions
Contextual dimensions are those depending on the context of the task. Three sub-dimensions describe those aspects, which are completeness, amount-of-data, relevancy.

Representational Dimensions
In this group, the sub-dimensions refer to aspects related to data design, such as representational-conciseness, representational-consistency, understability, interpretability, and versatility.

4.4 Quality Assessment Methodology

The methodology used to assess quality in Prêt-à-LLOD project relies on two assessment forms: a Quality form and a Linked Open Data Quality Form. The former has to be filled in by each partner involved in the development of Prêt-à-LLOD deliverables, using a self-assessment of quality level for the produced outcome. The latter, beside the self-assessment, exploits an automatic procedure to calculate values and degrees of some LOD dimensions.
Together with the deliverable, each partner has to submit both forms, that will be reviewed by a compliance committee formed by two members.

References

Ambler, S. W., & Lines, M. (2017). *An Executive's Guide to Disciplined Agile: Winning the Race to Business Agility (Volume 1)*. CreateSpace Independent Publishing Platform.

Falls, M. (2004). Inside the minds the software business : how top companies design, develop & sell successful products & applications, Inside the minds. Boston, Mass., Aspatore.

Huang, S.-J., Han, W.-M., 2008. Exploring the relationship between software project duration and risk exposure: a cluster analysis. Information and Management 45, 3, 175–182.

Jackson, M., Crouch, S., & Baxter, R. (2011). Software evaluation: criteria-based assessment. *Software Sustainability Institute*.

Kruchten, P. B. (1995). The 4+ 1 view model of architecture. *IEEE software*, *12*(6), 42-50.

Sutherland, J., & Schwaber, K. (2013). The scrum guide. *The definitive guide to scrum: The rules of the game. Scrum. org*, *268***.**

Tavares, B. G., da Silva, C. E. S., & de Souza, A. D. (2017). Risk management analysis in Scrum software projects. *International Transactions in Operational Research*.

van Gompel, M., Noordzij, J., de Valk, R., & Scharnhorst, A. (2016). Guidelines for software quality. *CLARIAH Task*, *54*.

Wang, R. Y., & Strong, D. M. (1996). Beyond accuracy: What data quality means to data consumers. *Journal of management information systems*, *12*(4), 5-33.

Zaveri, A., Rula, A., Maurino, A., Pietrobon, R., Lehmann, J., & Auer, S. (2016). Quality assessment for linked data: A survey. *Semantic Web*, *7*(1), 63-93.

D1.3 Software Quality and Architecture Plan Annex I - Architectural Representation

This project received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825182. The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Union.

D1.3 Software Quality and Architecture Plan
Annex I - Architectural Representation

1. Logical view

Partner Name	
Package Name	
Package Description	
Classes and other Packages used (list and diagram if possible)	
Class name	
Class Description	
Major Responsibilities	
Major Operations	
Major Attributes	
Relationships	
Realized use case	
Description of the realized use case	

Significant descriptions of the Flow of Events - Design of the use-case realization.	
Significant interaction enumeration or class diagrams related to the use-case realization.	
Significant/Derived Requirements of the use-case realization	

2. Process View

Partner Name	
Process Name	
Processes involved	
Interactions between processes (collaboration diagrams if possible)	
Process Behavior	
Process Lifetime	
Communication Characteristics	

3. Component View

Partner Name	
Component Name	
Component Description	
Component Scope	
Major Responsibilities	
Major Operations	
Major Attributes	
Dependencies	

4. Development view

Partner Name	
System Name	
System Type	
Development Methodology	
Implementation specifications	
Subsystems located in the system	
Subsystem Name (abbreviation or nickname)	
Subsystem Description	
Subsystem import dependencies (including a component diagram, if possible)	
Subsystem Properties	
If appropriate, indicate subsystem relationship to elements in the logical or process view.	

D1.3 Software Quality and Architecture Plan Annex I - Architectural Representation

This project received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825182. The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Union.

D1.3 Software Quality and Architecture Plan
Annex II - Quality Assessment Form

Note that criteria that are deemed not applicable can be striked through.

Partner	**Deliverable**	**Date**

Criterion		**No** 0	**Min** 1	**Adq** 2	**Good** 3	**Perfect** 4	**Comments**
Understandability							
Q1	Is it clear what the software does?						
Q2	Is there a specification about the intended users?						
Q3	Is it clear how the software works?						
Q4	Is there a software motivation?						
Q5	Is the development status clear?						

Q6	Is the support status clear?						
Documentation							
Q7	Is there a software documentation?						
Q8	Is the documentation accessible?						
Q9	Is the documentation clear?						
Q10	Is the documentation complete and accurate?						
Q11	Has a high-level overview of the software been provided?						
Q12	Does the documentation provide adequate examples?						
Q13	Is there a troubleshooting information file?						
Q14	Is there a FAQ file/section?						
Q15	Is the documentation available from the project website?						
Q16	Is the documentation under version control?						

Buildability and Installability							
Q17	Are there instructions for building/compiling the software?						
Q18	Are the dependencies listed and available?						
Learnability							
Q19	Is there a *Getting started* guide?						
Q20	Are there instructions for basic use cases?						
Q21	Is there a help reference?						
Q22	Is there API documentation for developers?						
Q23	If the software is configurable, are the configuration options documented?						
Identity and Copyright & Licensing							
Q24	Is there a clear and unique software identity?						
Q25	Does the software have a website?						

Q26	Does the software name not violate existing trade-marks?						
Q27	Has an appropriate open-source license been adopted?						
Q28	Are copyright, licensing, authorship, and funders acknowledgement clearly stated?						
Community							
Q29	Is there evidence of the software being in use by others?						
Q30	Is there evidence of external developers?						
Q31	Are statistics on software use available?						
Accessibility							
Q32	Is the source code maintained under a version control system?						
Q33	Is the source code in a public version-controlled repository?						
Q34	Are formal release of the software clearly marked?						
Q35	Is the software deposited in a persistent store with a unique DOI?						

Testability							
Q36	Are there unit tests?						
Q37	Are there integration tests?						
Q38	Are tests run automatically?						
Portability							
Q39	Is it clear for what platforms the software is written?						
Q40	Is the software portable for multiple platforms?						
Q41	Does the software work multiple browsers?						
Supportability							
Q42	Is the support contact clearly marked?						
Q43	Are there public support channels available?						
Analysability							

Q44	Is the source code structured adequately?						
Q45	Is the source code commented adequately?						
Q46	Do comments generate API documentation?						
Q47	Are sensible names used?						
Q48	Are there no blocks of commented out code or obsolete files?						
Changeability & Evolvability							
Q49	Is the project open to contributions from third parties?						
Q50	Does the project have guidelines for contributions?						
Q51	Are code changes and their authorship publicly available?						
Interoperability							
Q52	Does the software use appropriate open standards for data?						

D1.3 Software Quality and Architecture Plan Annex III - Linked Open Data Quality

This project received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825182. The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Union.

D1.3 Software Quality and Architecture Plan
Annex III - Linked Open Data Quality Form

The definitions in this form are by Zaveri et al., 2016.
Please, note that the metrics marked with * will be assessed by the committee using an automatic checking system. You can add comments, if needed.

Note that criteria that are deemed not applicable can be striked through.

Partner	**Deliverable**	**Date**

Criterion		**Self-assessment**	**Comments**
Accessibility			
Availability *Availability of a dataset is the extent to which information (or some portion of it)is present, obtainable and ready for use.*			
LQ1	Does the server respond to a SPARQL query?		
LQ2	Is a RDF dump provided? Can it be downloaded?		

LQ3	Is there a URI returning an error response code or a detection of broken links?		
LQ4	Are there dead links or a URI without any supporting RDF metadata or no redirection using the status code 303?		
LQ5	Are there all local in-links or back-links?		
LQ6	Are there all forward links?		
LQ7	Is the content is suitable for consumption and accessible?		
Licensing *Licensing is defined as the granting of permission for a consumer to reuse a dataset under defined conditions.*			
LQ8	Is there the indication of a license in the VoID description or in the dataset itself?		
LQ9	Is there a license indicating whether reproduction, distribution, modification or reproduction is permitted?		
LQ10	Is the work attributed in the same way as specified by the author or licensor?		
Interlinking *Interlinking refers to the degree to which entities that represent the same con-cept are linked to each other, be it within or between two or*			

more linked data sources.			
LQ11*	Which are the measures of interlinking degree, clustering coefficient, centrality and sameAs chains, description richness through sameAs		
LQ12	Are there external URIs and owl:sameAs k=links?		
Security *Security is the extent to which access to data can be restricted and hence protected against its illegal alteration and misuse.*			
LQ13	Are login credentials or SSL or SSH used?		
LQ14	Are data of proprietary nature?		
Performance *Performance refers to the efficiency of a system that binds to a large dataset, thatis, the more performant a data source the more efficiently a system can process data.*			
LQ15	Are there slash-URIs (wrt large amount of provided data)?		
LQ16*	How is the delay between submission of a request by the user and a reception of the response from the system?		
LQ17*	What is the throughput?		

LQ18*	What is the scalability of data source?		
Intrinsic Dimensions			
Accuracy *Accuracy is defined as the extent to which data is correct, that is, the degree to which it correctly represents the real world facts and is also free of syntax errors. Accuracy is classified into (i)syntactic accuracy, which refers to the degree to which data values are close to its corresponding definition domain and (ii) semantic accuracy, which refers to the degree to which data values represent the correctness of the values to the actual real world values.*			
LQ19*	Did you detect outliers?		
LQ20	Did you detect inaccurate values?		
LQ21	Did you detect inaccurate facts?		
LQ22	Are there malformed datatype literals?		
LQ23	Are there literal incompatible with datatype range?		
LQ24*	Are there erroneous annotation/representation?		
LQ25*	Are there inaccurate annotation, labelling, classification?		
Consistency *Consistency means that a knowledge base is free of (logical/formal) contradictions with respect to particular knowledge representation and inference mechanisms.*			

LQ26*	Did you detect entities as members of disjoint classes? (Please, report the number in the comment field)		
LQ27*	Did you detect the usage of homogeneous datatypes? (Please, report the number in the comment field)		
LQ28	Are there misplaced classes or properties?		
LQ29*	Is there a misuse of owl:datatypeProperty or owl:objectProperty?		
LQ30	Is there a use of members of owl:DeprecatedClass or owl:-DeprecatedProperty?		
LQ31	Are there bogus owl:Inverse-FunctionalProperty values? (Please, provide a list)		
LQ32	Have external classes/properties been used (ontology hijackings)?		
LQ33	Is there a misuse of predicates?		
LQ34*	Are there ambiguous annotations?		
Conciseness			

Conciseness refers to the redundancy of entities, be it at the schema or the data level. Conciseness is classified into (i) intensional conciseness (schema level) which refers to the case when the data does not contain redundant attributes and (ii)extensional conciseness (data level) which refers to the case when the data does not contain redundant objects.			
LQ35*	What is the degree for intensional conciseness?		
LQ36*	What is the degree for extensional conciseness?		
LQ37*	Are there duplicate instances?		
Trust Dimensions			
Reputation *Reputation is a judgment made by a user to determine the integrity of a datasource.*			
LQ38	Please, assign an explicit ranking to the dataset		
LQ39	Have external links or page rank been analyzed (semi-automatically)?		
Believability *Believability is defined as the degree to which the information is accepted to be correct, true, real and credible.*			
LQ40	Is the provider/contributor contained in a list of trusted providers?		
LQ41	Are there title, content and URI of the dataset (namely provenance information)?		

LQ42*	What is the value of the trustworthiness of RDF statements?		
LQ43*	What is the value of the trustworthiness of entities?		
LQ44*	What is the value of the trustworthiness of entity pairs?		
LQ45	Did you acquire content trust from users?		
LQ46	Please, assign trust values to data/source/rules		
LQ47	Did you determine trust value for data?		
LQ48	Did you compute personalized trust recommendations?		
LQ49	Did you detect reliability and credibility of data source?		
LQ50	Did you compute the trustworthiness of RDF statements?		
LQ51	Please, assign a level of reliability and credibility of the dataset publisher		
Verifiability *Verifiability refers to the degree by which a data consumer can assess the correctness of a dataset.*			

LQ52	Did you verify the authenticity of the dataset?		
LQ53	Did you use digital signatures?		
LQ54	Did you verify the correctness of the dataset?		
Objectivity *Objectivity is defined as the degree to which the interpretation and usage of data is unbiased, unprejudiced and impartial.*			
LQ55	Did you check the objectivity of the information?		
LQ56	Did you check the objectivity of the source?		
LQ57	Is the dataset biased?		
Dataset Dynamicity Dimensions			
Currency *Currency measures how promptly the data is updated.*			
LQ58*	*What is the currency of documents/statements?*		
LQ59	*What is the time since modification?*		
LQ60*	*Have outdated data been excluded?*		

Volatility *Volatility refers to the frequency with which data varies in time.*			
LQ61	What is the frequency of change (wrt `changefrequency` attribute in the Semantic Sitemap)?		
LQ62*	What it the time validity interval?		
Timeliness *Timeliness measures how up-to-data data is, relative to a specific task.*			
LQ63	What is the difference between last modified time of the original source and last modified time of the semantic web source?		
LQ64	What is the difference between current and expiry time of the resource?		
LQ65*	What is the difference between the idea freshness and the data source freshness?		
Contextual Dimensions			
Completeness *Completeness refers to the degree to which all required information is present in a particular dataset. In terms of LD, completeness comprises the following aspects: (a) Schema complete-ness, the degree to which the classes and properties of an ontology are represented, thus can be called "ontology completeness", (b) Property completeness, measure of the missing values for a specific property, (c)Population completeness is the percentage of all real-world objects of a particular type that are represented in the datasets and (d) Interlinking completeness has tobe considered especially in LOD and refers to the degree to which instances in the dataset are interlinked.*			

LQ66	Schema completeness: Number of classes and properties represented / Total number of classes and properties		
LQ67	Property completeness: Number of values represented for a specific property / Total number of values for a specific property		
LQ68	Population completeness: Number of real-world objects represented / Total number of real-world objects		
LQ69	Interlinking completeness: Number of instances in the dataset that are interlinked / total number of instances in a dataset		
Amount-of-data *Amount-of-data refers to the quantity and volume of data that is appropriate for a particular task.*			
LQ70*	ratio of no. of semantically valid association rules to the no. of non-trivial rules		
LQ71*	Number poor predicates based on the occurrence dependencies among predicates		
LQ72	Number of triples present in the dataset		
LQ73	Scope (no. of entities) and level of detail (no. of properties)		

Relevancy *Relevancy refers to the provision of information which is in accordance with the task at hand and important to the users' query.*			
LQ74*	What is the usage of meta-information attributes?		
LQ75*	What is the relevancy of retrieved documents for a given query?		
Representational dimensions			
Representational-conciseness *Representational conciseness refers to the representation of the data which is compact and well formatted on the one hand and clear and complete on the other hand.*			
LQ76	Are there long URIs or containing query parameters?		
LQ77	Did you use prolix RDF features (i.e., RDF reification, RDF containers, RDF collections?)		
Representational-consistency *Representational-consistency is the degree to which the format and structure of the information conforms to previously returned information as well as data from other sources.*			
LQ78	Did you use existing terms from other vocabularies?		
LQ79	Did you use established vocabularies?		
Understandability *Understandability refers to the ease with which data can be comprehended, without ambiguity, and used by a human in-formation consumer.*			

LQ80	What is the percentage of entities having an rdfs:label or rdfs:comment?		
LQ81	Did you use rdfs:label to attach labels or names to resource?		
LQ82	Has the pattern of URIs been used?		
LQ83	Did you use regular expressions that match the URIs?		
LQ84	Have examples of SPARQL queries been provided?		
LQ85	Did you provide a list vocabularies used in the dataset?		
LQ86	Is the usage of the mailing list and/or the message boards effective and efficient?		
Interpretability *Interpretability refers to technical aspects of the data, that is, whether information is represented using an appropriate notation and whether it conforms to the technical ability of the consumer.*			
LQ87	Did you use self-descriptive formats?		
LQ88	Did use various schema languages to provide definition for terms?		
LQ89	Did you use blank nodes?		

LQ90	Is there an atypical use of collections, containers and reification?		
Versatility *Versatility refers to the availability of the data in an internationalized way,the availability of alternative representations of data and the provision of alternative access methods for a dataset.*			
LQ91	Are the data available in different serialization formats?		
LQ92	Are the data available in different languages?		
LQ93	Are the data available as SPARQL endpoint and for download as RDF dump?		
LQ94	Can the data be retrieved in accepted formats and languages by adding a corresponding accept-header to an HTTP request?		