Document title:	Software best practices
Dissemination level:	Public
Document type:	Other
Version:	1.2
Date:	December 20th, 2021

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

Document Information

Project Number:	825003
Project Acronym:	DIH-HERO
Project Title:	Digital Innovation Hubs in Healthcare Robotics
Work Package:	WP7
Author(s):	Ainara Garzo, Anthony Remazeilles, Michael Obach, Ana Cruz, Beatrice Sangregorio
Dissemination Level:	Public
Document Type:	Report
Total Number of Pages:	30

Table 1. Document Information

Revision History

Version	Description	Responsible Person	Date
1.0	First draft of the document	Ainara Garzo	26/11/2021
1.0	Document development	Anthony Remazeilles Ainara Garzo	03/12/2021
1.1	Review	Michael Obach	06/12/2021
1.1	Review	Ana Cruz	16/12/2021
1.1	Review	Selene Tognarelli	16/12/2021
1.2	Reviews integration and final version of the document	Ainara Garzo	20/12/2021

Table 2. Revision History

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

Abstract

While many standards for medical devices and robotics are well-established and published, at the beginning of the project, it was detected that most industry-led *de facto* standards and especially best practices in healthcare robotics are not yet documented, collected, structured, harmonized or promoted. For this reason, in the present document, a first version of best practices summary for software development for robotics in healthcare is included.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

Table of Contents

List of Tables

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

List of Figures

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

List of Abbreviations

CD	Continuous Development
CI-CD	Continuous Integration and Continuous Deployment
DIH-HERO	Digital Innovation Hub in Healthcare Robotics
FSTP	Financial Support for Third Parties
ISBP	Industry-led Standards and Best Practices
ROS	Robot Operating System
SME	Small-to-Medium Enterprise
SOP	Software Operating Procedure
SRA	Software Risk Assessment
WP	Work Package

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

1 Introduction

The work required to translate concepts and prototypes into commercial products within the healthcare robotics market is often highly complex. Although multiple standards exist, companies, and especially medium-sized enterprises (SMEs) struggle to identify and apply them to obtain the required product certification. The present document describes the analysis carried out within work package (WP7) to ease navigation through complex certification pathways, with a focus on existing standards and best practices for the development of software in healthcare robotics.

1.1 Purpose of this document

The objectives targeted in T7.4 are:

- Identify the difficulties faced by the healthcare robotics industry regarding standards and best practices collection and interpretation.
- Identify the standards and best practices used by healthcare robotic companies for the development of software.
- Accelerate the process of certification to launch products on the market.

The current version of this deliverable provides information about the preliminary conclusions taken to accomplish the T7.4 objectives as of December 2021.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

2 Methodology and collection of data

The methodology was based on gather information, contrast and promote software best practices that enhance software quality and management to: 1) reduce development efforts, 2) add value to the industry, and 2) expose existing barriers in healthcare robotics.

2.1 Survey on best practices for robotics in healthcare

With the aim of identifying needs and gathering information on software development best practices for healthcare robotics, an online survey was created. The questions on the survey are open and optional and they are divided in five groups: processes and methodologies, middleware and libraries, community hardware, burdens for software certification, and community activities. Moreover, some information about the participant's organisation is requested, such as the country, type and size, or the application domain. The questionnaire can be found in the following link: https://forms.office.com/r/xpT20krHsn

This questionnaire was launched during the Knowledge Conference (see section 2.2) and promoted via LinkedIn (see Figure 1). The questionnaire is still open to send any feedback and complete the current document.

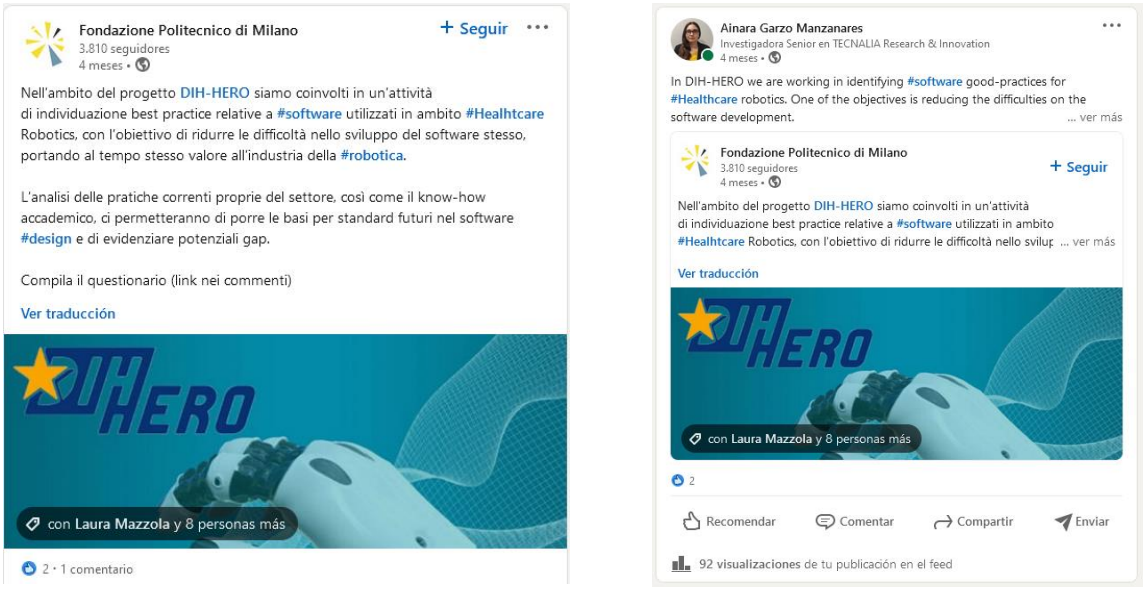

Figure 1. LinkedIn posts to promote the prepared survey on software development best practices

2.2 Certification and standards session during the DIH-HERO Knowledge Conference

On the 19th and 20th of May of 2021 an online conference was organized by the DIH-HERO consortium to share the knowledge of different experts on key topics in healthcare robotics[1]. A specific session on certification and standards was organized by TECNALIA, featuring the following participants:

- Dr. Thierry Keller and Dr. Anthony Remazeilles (TECNALIA)

[1] https://dih-hero.eu/dih-hero-knowledge-conference/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

- Dr. Jan Veneman[2] (Hocoma, Action Chair from the COST Action Weareable Robotics[3])
- Dr. Eduard Fosch[4] and Hadassah Drukarch[5] (Leiden University, LIAISON project[6])
- Gerdienke Prange (Roessingh Research and Development[7] centre)

In this session, results from the DIH-HERO project regarding certification, standards and best practices in healthcare robotics were presented. Dr. Anthony Remazeilles made a summary of best practices applicable to software development and introduced the aforementioned survey (see section 2.1). This session was interactive, and panellists organized a discussion with the attendees about their experience with standards and best practices applied to robots for healthcare.

2.3 Results and Perspectives

The feedback gathered during these sessions, as well as the input from DIH-HERO core partners was used to draft the first version of present document on best practice for software development. The information will be completed in the following months with additional inputs from experts in software development best practices. At the end of the project, collected information will be published online on the DIH-HERO portal.

[2] https://www.utwente.nl/en/techmed/database/biography/jan-veneman/

[3] https://www.cost.eu/actions/CA16116/

[4] https://www.universiteitleiden.nl/en/staffmembers/eduard-fosch-villaronga#tab-1

[5] https://www.universiteitleiden.nl/en/staffmembers/1/hadassah-drukarch#tab-2

[6] https://liaison2020.eu/

[7] https://www.rrd.nl/en/mensen/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

3 Best practices for software development in healthcare robotics

3.1 Code generation methodology

Gathering good practices at a higher level. All this is likely to be connected to the coding best practices as well.

3.1.1 Team organization, roles & responsibility

When developing software for robots or any other product, and like in any team-based activity, it is important to assign to each team member a clear role and associated responsibilities. These responsibilities should be in-line with the development process, stating who will generate, review and validate any item or component, etc.

The roles and responsibilities strongly depend on the team size and on the work methodology, however it is quite common to define profiles like the one mentioned in Table 3.

Role
Product Manager
Project Manager
Software Architect
Software Developer
Software Tester

Table 3. Proposal of roles for the members of a development team.

A proper definition of roles and responsibilities ensures a common understanding of each team member's contribution, boundaries, expected interactions, etc.

3.1.2 Objectives / requirements, verification & traceability

The definition of the purpose of the software, together with its scope, allows the team to define and agree on boundaries for each component. Clearly stating the objectives and/or requirements, simplifies the implementation of the verification processes needed to ensure correct component behaviour and capabilities. The traceability consists then in connecting each requirement with its verification item, and (possibly) with the related code component.

3.1.3 Development model

In addition to establishing the team's organisation and roles (see section 3.1.1), it is also important to define a mutually agreed development methodology defining how the team will work progressively towards the desired software product. In some cases, these methodologies can even be linked with the version control strategy used for code generation.

Numerous models exist, as shown in Wikipedia[8]. Some of the most popular are described in section 5.1 of this document, such as waterfall, v-model, scrum or kanban (based on agile), lean or devops (some of which can be combined).

[8] https://en.wikipedia.org/wiki/List_of_software_development_philosophies

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

The developer should always follow an established development model to avoid falling into a chaotic process and unexpected results. Strict adherence to the model is not required, and it may be adapted to fit the team and project needs, but once a methodology is established it should be respected throughout the development cycle.

3.1.4 Development planning

Development planning consists in defining when a given documentation or software component will be produced. Setting milestones is the best way to track progress, detect deviation issues, and avoid spending too much time on un-relevant items. Most of the development models in section 3.1.3 capture this aspect.

3.1.5 Software architecture description

It is a good practice to describe the software through graphical representations explaining how it is structured and how it works, including the interaction among the different packages or with the user. When certification is envisioned, the description of the software architecture should be prepared before its implementation. Depending on the potential risk on human integrity, the software description may even have to cover each of the different items involved in the system. In any case, such description is useful to get all the development team aligned, and even for the individual developer, in particular if the development has to be resumed after a certain period of time.

3.1.6 Risk analysis and management

The risk analysis consists in (i) detecting any item that may affect the outcome of the software and (ii) characterizing the impact of this misbehaviour. Mitigation actions are then proposed to either prevent the risk from occurring or to reduce its impact to an acceptable outcome. Making such effort is a way of getting a more robust software.

It is recommended to maintain a parallel document, usually called Software Risk Assessment (SRA) where risk analysis and risk mitigation measures are considered. Each software item contributing to a hazardous situation described in the System Risk Analysis is listed alongside the corresponding control measure in the SRA.

A follow-up of the risks must be done continuously when a new release is performed, as new risks may appear when the software functionality or implementation is changed.

3.1.7 Verification & testing material

It is a good practice to verify that the system behaves as expected, which is the purpose of code testing. Ideally, testing should not only verify the generated output given a reference input, but also that the system is able to detect and cope with invalid inputs, to minimize the risk of unexpected behaviours. Testing types are divided into functional and non-functional categories. Unit tests, integration test, regression and acceptance tests are considered functional. Non-functional testing includes system performance, security, scalability, portability and static code analysis and evaluation. Different testing types are mentioned in section 5.2.

The implementation of a test framework is the only way to systematically verify the quality of the software system, in terms of functional and non-functional objectives and requirements.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

The inclusion in the code of testing material and the related reference data is also a convenient way to have some testing traceability and to show others how to use the software, so that they can quickly launch the system by their own without looking for sample data.

3.1.8 Configuration management: version, change, build, release

When getting close to the code generation phase, it is necessary to establish and respect a development production control including every steps. Looking towards the certification requirements, this control should provide a means of traceability from the user and systems requirements, to the code implementation, and testing results.

The configuration management should cover all items, such as the source code, libraries, build scripts, software tools, SOUP (see below), and the documentation. Version control aspects are more thoroughly explained in section 3.3.1.

3.1.8.1 SOUP

Nowadays a significant number of software components are being shared with licenses that are more or less permissive. The concept of "don't reinvent the wheel" clearly applies here, and new solutions should be built upon available material, allowing developing time to be invested on producing real added value.

The term SOUP (Software of unknown pedigree) refers to such type of external code: any piece of code that is integrated in the software, without prior knowledge and control of its development process to guarantee its quality (particularly around testing).

If the use of such SOUP is beneficial for our software (to reduce production cost for example), it also adds a dependency that must be properly characterized. At minimum, we should capture the following information:

- the name of SOUP
- the manufacturer of SOUP
- the Version of SOUP

3.1.8.2 Change control

As described in section 3.3.1, the production of code is progressive, and it is important to track its evolution, the reason why it changes, while also proving that the change produces the expected behaviour without disturbing the overall system behaviour. A policy must be taken to define the desired change (e.g., adding a functionality, handling a bug, extending features, etc.), and to establish its implementation, validation, review, and integration within the global solution. Section 3.3.1 provides means to do so from a more pragmatic point of view.

When the software development undergoes significant changes, the following steps must be taken:

- It should be ensured that no unintended behaviour has been introduced.
- A regression test or additional test shall be performed.
- A risk management process shall be reviewed as appropriate.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

3.1.8.3 Release and Deployment

The software deployment cannot be conducted without considering how and when it will be used by others and deployed in the targeted environment.

Its release may be associated to a stage of development of a given maturity, such as a milestone. When the software is used by others, the release version can be used to identify the version of the software, and to identify the underwent update (extension of the functionalities, improvement of existing functionality, bug resolution, …).

Furthermore, it is also important to associate each release to quality milestones in terms of testing, documentation, etc... Even if the software may not be at a ready-for-use status, it is an important stage of the development. From a practical point of view, at this moment, the state of all related material should be logged (for instance, through a tagging mechanism as described in section 3.3.1.7), testing-verification campaign should be conducted, and appropriate documentation material should also be provided.

When the releases reach a maturity level enabling the use of the system in the intended environment or by intended users, the deployment strategy must be defined: how the software can be built, made accessible for further use. In that respect, CI-CD concepts (see section 3.3.4) enable to envision such process even at an early stage of development.

Before the software is released, the following activities should be completed:

- Software verification is complete.
- Residual anomalies are evaluated and documented.
- Information regarding the release process is documented.

3.1.9 Problem resolution process

Software development is not a straightforward process, and it is important to define how to handle any problem that may occur during the development phase, i.e. during integration testing or system testing, or after the software is released or launched. A problem may be a bug (unexpected behaviour) which hinders performance, but also a new feature request (which was not planned). Recent code hosting systems associate problem resolution process to issues (see section 3.3.1.6), which can be directly handled at that level by the developer team. Ways of handling this problem resolution must be defined to have a good control of them. Once the software is developed, this would be part of the maintenance process, which should also specify how external users can report issues, how these are communicated to the developer team, and once solved, how to inform users of its resolution

The developer team must define procedures to:

- Assign the issue to appropriate developer;
- Identify the issue, and propose a solution for it;
- Implement the appropriate change;
- Verify that the changes do not affect the rest of the system (through testing for instance);
- Comment on its resolution;
- Review, decline, and accept the change.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

Most code version control systems provide an appropriate interface to conduct this type of process.

The outcome of problem evaluation should be documented. If the decision is made that no action will be taken to solve the problem, the justification of this decision shall also be documented. The relevant parties will be advised as appropriate.

In each phase, the approach for the problem resolution process is defined as follows:

During the development phase: A problem report can be made in a form of bug report. The corresponding responsible person shall investigate the problem and decide whether changes to the software are needed.

After the software is released/launched: A problem report will be created. Identified problems must be investigated and evaluated. After the decision is made that changes to the software are needed, the change control process will be triggered.

3.1.10 Software maintenance process

Software development is a never-ending process as user expectations, hosting and devices are permanently evolving. It is therefore important to also define how to perform the following activities:

- Establishing a plan for collecting, documenting, evaluating, resolving, and tracking of feedback.
- Establishing criteria for determining whether feedback should be considered as a problem.
- Maintaining the risk analysis updated with any evolution of the code.
- Applying the problem resolution process when needed.
- Maintain a control on the code versions deployed, with a strategy to share any updates with the client.
- Handling of SOUP items regarding evaluation and implementation upgrades, bug fixes, patches and obsolescence.

3.2 From good practices to regulation processes

If the software development is intended to be used as or within a medical device, it is necessary to comply with the required regulation processes. From a software perspective, the objective of the norms is to ensure that software-related risks are maintained tolerable for human beings. One of these norms is IEC 62304, which focuses on software life cycle processes. It defines a rigorous methodology that must be followed and documented to make sure these risks are well identified and controlled.

That norm defines methodologies of development that must be followed from the beginning of the project, i.e. before writing any lines of code. It covers main items like risk management, development maintenance, configuration, software-related problem solving.

Figure 2 illustrates all the documentation that should be generated while applying the norm. Note that most of the required items are strongly related to the aforementioned *good practices*. Indeed, these practices are not only required for developing software efficiently, but also to

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

conduct the certification. The certification requires a rigorous application of best practices, to maintain an overall and systematic traceability of all the items in the development process, and to document all stages exhaustively.

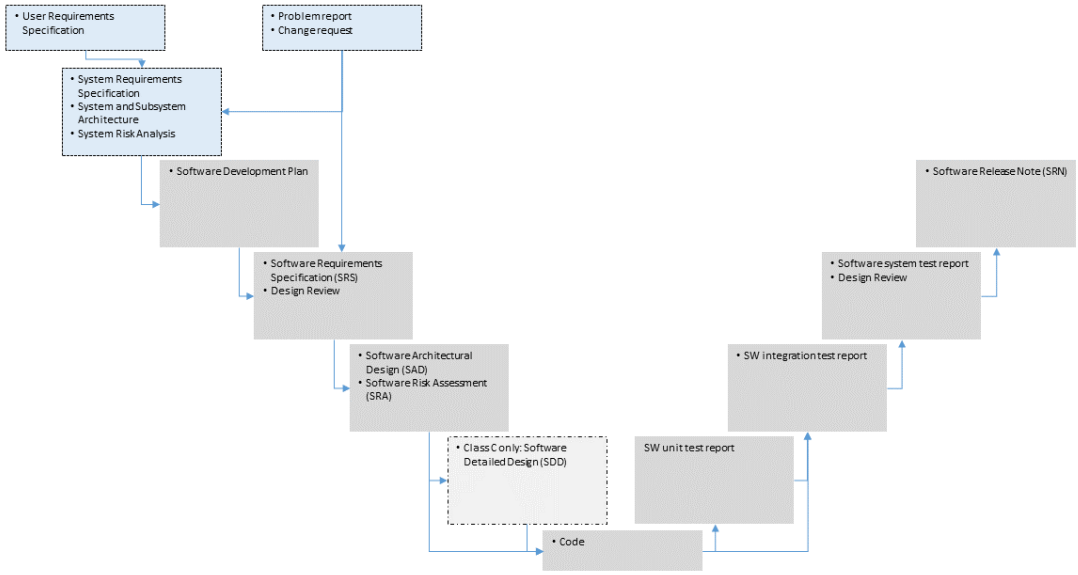

Figure 2. Software document model, as expected by the 62304 norm. The figure follows a V development model, but other software development models are compatible with the norm.

An important point mentioned in the norm and not covered so far is the software safety classification. Depending on the intended use of the software, one must define whether (class A) no injury or damage to health is possible, (class B) non serious injury is possible, or (class C) death or serious injury is possible. Depending on the class, the requirements placed on the software development methodology differ, and higher classes require more documentation items as illustrated on Figure 2.

Outside from a regulation process, depending on the maturity of the software developed, it may be difficult to define the intended use and thus, its classification. We could even argue that raising such question for investigation or exploratory development may be useless, as the development may still be at a very early stage. Nevertheless, it is quite common for development to start in an exploratory stage, and to progressively consolidate towards stable versions. This mature version may demonstrate that a proper medical software can be envisioned, for which the regulation process should be thoroughly followed and applied.

There is very little chance that a software development that has started as an exploration or investigation, is compliant with all processes that should be followed for developing certifiable software. Nevertheless, if some good practices are used from the beginning, the transition towards its certifiable version will be easier. This is yet another reason for trying to apply these good practices for any software development on a daily basis!

In this section, we have seen that a significant amount of methodological choices should be made explicitly when developing software. And most of the decision items may not be specific

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

to a given software but may be relevant to the team's general development practices. It is a good practice to define all these elements in a document named *Software Operating Procedure* (SOP). The advantage of creating such document is that all developers can refer to it to know the practices applied in the group. Also, a specific piece of software can refer to such document to describe the development strategy and avoid information repetition.

Note that in the certification process, a major document to prepare at the beginning is the *Software Development Plan*, or SDP. The norm specify what processes should be executed, but not how to conduct them. This is part of the objectives of the SDP a required document for any medical software. Nevertheless, if the development team already follows a defined SOP, it is possible to create an SDP with references to the relevant SOP section(s) containing development choices that are followed by the team.

3.3 Code-related content

Getting closer to the source code, with an emphasis on the version control system, and how its use allows answering to the item mentioned in the section 3.1.

3.3.1 Version control

The use of a version control system is mandatory for any development, for several reasons such as tracking changes, collaborative development, workflow, and backups.

Among all available version control systems used, git[9] is the most popular.

Git is available on all major programming platforms, i.e. Windows, Linux or Mac. Under Linux, Git is frequently used from the terminal, but relevant interfaces are available as well[10].

3.3.1.1 Repository hosting

Git hosting systems are servers where the Git repositories are hosted. The most famous are Gitlab[11] GitHub[12] and Bitbucket[13]. All of them enrich the repository with several tools for collaborative work and workflow management, such as issue tracking, merge request management, tags and release management, documentation generation, Wikis, CI-CD pipelines embedded, etc.

3.3.1.2 Repository storage

For modularity and reuse purposes, git promotes the use of small repositories, so that an application can be divided into different repositories. After some time, the number of repositories produced can be significant, so a proper organization strategy must be defined.

[9] https://git-scm.com/

[10] https://git.wiki.kernel.org/index.php/Interfaces,_frontends,_and_tools#Graphical_Interfaces

[11] https://about.gitlab.com/

[12] https://github.com/

[13] https://bitbucket.org/product/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

3.3.1.3 Repository membership visibility and accessibility

Git hosting services usually propose a concept of namespace into which different repositories can be stored. It is up to the developer team to agree on a policy for organizing the repositories under this namespace. Note that access rights can be configured at the level of the namespace directly, being then common to all repositories placed in it.

Git is a distributed hosting system, and therefore one repository can be hosted in various hosting services at the same time, at the cost of the logistics for maintaining all up to date.

3.3.1.4 Code

3.3.1.4.1 Binaries and large file storage

The basic versioning of Git does not fit well with files that are not simple code text (binaries, videos, images, …), and the repository size can increase quickly when such files are included in the repository (as the history of versions is also stored locally).

Git comes with a Large File System extension[14] , which enables to keep in the code repository pointer to such large files that are then stored onto a more specialized server.

3.3.1.5 Development workflow

3.3.1.5.1 Commit

A commit must be a logical unit, addressing a single atomic change. Any commit is associated with a message that should be sufficiently explicit to describe the change performed.

3.3.1.5.2 Branch-based development

The workflow defines the methodology used to interact with the repository. A workflow is always used (even though it is not always deliberately and consciously selected). Several models are available and described elsewhere:

- This Bitbucket document[15] compares different workflows,
- GitHub flow[16],
- The advanced branching model proposed by Vincent Driessen[17]

If the context justifies it, a release branch can be created to host the successive releases of the code (to a client, or to highlight important milestones, etc.). In that case, a tagging mechanism is frequently used to highlight the main changes applied from one version to another.

3.3.1.6 Issue Tracking

The issue tracking is strongly related to the problem resolution process previously mentioned. Most Git hosting servers provide an interface for registering and managing founded issues. The

[14] https://git-lfs.github.com/

[15] https://www.atlassian.com/git/tutorials/comparing-workflows

[16] https://guides.github.com/introduction/flow/

[17] https://nvie.com/posts/a-successful-git-branching-model/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

management of the issues is strongly related to the workflow used, but generally speaking, it is good practice to:

- Create an issue as soon as a problem or bug is detected. Use the given interface to detail the encountered problem.
- Assign the issue to a Developer that should handle it.
- Use the interface to possibly discuss the item and or how to solve the problem.
- Create a branch from the main branch and solve the issue in it.
- Create a merge request to request the review of the solution proposed.
- Once merged to the main branch, close the issue created.

For traceability matters, it is good practice to follow this pattern even if the issue management is assigned to the same person that detected the issue.

It is also a good practice to create an issue even for commenting a code extension. This highlight the development taking place and connects it to the branch where the development is performed.

It is a good practice to associate to each issue the branch in which it is handled. Gitlab enables for instance to create a branch directly from the issue description page, which improves traceability.

3.3.1.7 Tagging strategy

Software versioning is the "*process of assigning unique version names or numbers to unique states of computer software*"[18].

The unicity of the versioning is ensured per the unique commit hash number. This number is automatically assigned by Git.

The Tagging process enables the identification (at a lower frequency) of special versions of the development. The tagging methodology is supposed to provide some human understandable message.

We can highlight the following methodologies:

- Semantic versioning[19], using a Major.Minor.Patch triplet related to the severity of the change performed.
- Sentimental or romantic versioning[20] applies subjective criteria to the 3 previous numbers.
- Sequential versioning, by incrementing a simple integer per release.
- Date-based versioning, where the tagging pattern is based on objective dates.

More models are detailed in InedoBlog[21].

[18] https://en.wikipedia.org/wiki/Software_versioning

[19] https://semver.org/

[20] http://sentimentalversioning.org/

[21] https://blog.inedo.com/blog/release-numbering-scheme

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

If a tagging model is used, it is a good practice to handle a release branch, where specific Continuous Development (CD) processes can be automatically launched.

Note that when a tagging strategy is implemented, it is a good practice to associate it with a Changelog[22] file, to be placed in the code repository.

3.3.2 Code guideline

3.3.2.1 Style guide

Using standard code style is encouraged. Standard styles exist for most programming languages. The standard used is decided by the development group.

Under ROS1, the package roslint[23] can be used to check the compliance with the reference ROS style guides. ROS is provided with some style guidance[24], and an update has been proposed for ROS2[25]. The C++ style is based on the Google Style Guide[26], with some slight differences. The Standard C++ foundation refers to other styles as well[27]. The python style guide in ROS is based on PEP8[28].

For Matlab code, Richard Johnson from Mathworks proposed the "Matlab Style Guidelines 2.0"[29].

When possible, the IDE (Integrated Development Environment) should be configured to highlight code style violations (frequently named as *code linters*). To name a few, Sublime Text and Visual Studio Code provide such type of tools. Main IDE can also be configured to automatically generate file, function, or class documentation.

PickNickRobotics provides clang files for ROS enabling automatic C++ code formatting, which can be loaded into several editors[30]. ROS2 is associated to a set of linters which should be automatically used to ensure code quality[31].

For Matlab code, Mathworks provide its own tool, *mlint* which is now known as checkcode[32]. This linter is for instance used in the Matlab extension for Visual Code from Xavier Hahn[33].

---

[22] https://keepachangelog.com/

[23] https://wiki.ros.org/roslint

[24] See http://wiki.ros.org/StyleGuide for general guidance, http://wiki.ros.org/CppStyleGuide for C++ code, and http://wiki.ros.org/PyStyleGuide for python code

[25] https://docs.ros.org/en/galactic/Contributing/Code-Style-Language-Versions.html

[26] https://google.github.io/styleguide/cppguide.html

[27] https://isocpp.org/wiki/faq/coding-standards#coding-std-wars

[28] https://www.python.org/dev/peps/pep-0008/

[29] https://www.mathworks.com/matlabcentral/fileexchange/46056-matlab-style-guidelines-2-0

[30] https://github.com/PickNikRobotics/roscpp_code_format

[31] https://docs.ros.org/en/galactic/Contributing/Developer-Guide.html#quality-practices

[32] https://fr.mathworks.com/help/matlab/ref/checkcode.html

[33] https://marketplace.visualstudio.com/items?itemName=Gimly81.matlab

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

https://awesome-linters.hugomartins.io/ & https://asmen.icopy.site/awesome/static-analysis/ provide an exhaustive list of linters per language. Note that some of these linters are also doing static code analysis, which allow the identification of possible code errors, even without launching the code.

3.3.2.2 Packaging and building

We promote the use of standard code packaging tools for easing (i) the code installation and (ii) the code reuse. The use of standalone building mechanism provides less dependence to the IDE (when the language permits it).

For instance, under Python it is a good practice to generate packages similarly to the ones we frequently install (through pip mechanisms for instance). The installation and dependency management are facilitated, in particular in combination of virtual environments[34].

For C and C++, the use of CMake[35] enables to get a building mechanism independent of the IDE and the compiler used. The organization of the development in components which can be compiled as libraries is also a good practice. A well designed CMake structure can permit compiling the very same code under Windows and Linux.

Under **ROS**, we recommend some guidelines for developing the aforementioned ROS components. Most of these are standard amongst the community, such as how to configure the build/install system for each component, how to declare it's dependencies, or how the different types of files (source, configuration, etc.) are to be organized inside each component.

The community practice for dependency resolution relies on dependencies being publicly available in ROS' federated package repository.

3.3.2.3 Repository meta information

Ideally a code repository, additionally to the code, should also contain documentation on:

- The purposes of the code
- Indication on the developers and copyright guidelines
- Licensing terms
- Clear indications on the dependencies of the component, and how to install them
- Clear indications on how to compile and install the repository content
- Clear indications on how to use the code, with (when possible) clear examples of program launch with input and expected output results (when appropriate).
- Documentation on the system architecture

[34] More information at https://packaging.python.org/tutorials/packaging-projects/ and https://docs.python-guide.org/writing/structure/

[35] https://cmake.org/cmake/help/latest/guide/tutorial/index.html . Package templates can be found from Google search, like (note tested): https://opensourcelibs.com/lib/modern-cpp-template . Daniel Pfeifer talk at C++ now provides a good overview of modern CMake: https://www.youtube.com/watch?v=bsXLMQ6WgIk

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

All this information may be placed in a single documentation file, named README.md (markdown format[36]) or README.adoc (AsciiDoc format[37]), placed at the root of the repository. This way the file is automatically rendered onto the GitLab or GitHub server. Some information may also be placed in separate files, but they should be always mentioned in the main README.

Illustration of Readme content can be found here[38]. ROS 2 presents some guidelines on the expected documentation of a package[39].

3.3.3 Code testing

The code testing is promoted for verifying that each development provides the expected outcome and unexpected outcomes will not appear. Some software development working approaches even advise to write the test before the functionality[40].

In C++, GoogleTest and Boost Test are common frameworks for testing. Python provides pyUnit for unitary test. In ROS, the rostest suite can be used to test at the component and application level.

Several applications allow also some static code analysis and can identify several implementation errors without executing the code. SonarQube[41] or FindBug[42] are some examples. This type of testing is not considered within the regular testing framework.

3.3.4 CI-CD

Continuous Integration and Continuous Deployment (CI-CD) consists in automating some operations (of integration, deployment, but not only). Usually, this automation is brought to the version control system, where operations can be triggered automatically when a defined operation is executed on the repository (for a new push, on a given branch, on a release, …).

Several tools are available for defining such automation. To name a few:

- Gitlab provides CI-CD functionalities[43] using either specific machines to run specific operations or using docker mechanisms to define the required operations.
- GitHub is providing the Action concept[44], which has the advantage of being collaborative, so that actions defined by some can be easily integrated into another CI process.

---

[36] https://daringfireball.net/projects/markdown/

[37] https://asciidoctor.org/

[38] https://www.makeareadme.com/

[39] https://docs.ros.org/en/rolling/Contributing/Developer-Guide.html#documentation

[40] https://en.wikipedia.org/wiki/Test-driven_development

[41] https://www.sonarqube.org/

[42] https://findbug.io/

[43] https://docs.gitlab.com/ee/ci/

[44] https://github.com/features/actions

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

- Both hosting systems are able to interact with standard CI-CD environment, such as CircleCI[45], Travis[46] or Jenkins[47].

The purposes of a CI-CD process can be numerous as it is usually associated to a script being launched, but we promote the use of CI-CD to:

- Verify the building process on a fresh machine (through docker).
- Launch tests.
- Automatic generation of code artifacts (executable, library, docker image, …)
- Deployment of these artifacts to external hosting mechanisms, possibly accessible by Third Parties without direct access to the code repository.
- …

[45] https://circleci.com/

[46] https://travis-ci.org/

[47] https://www.jenkins.io/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

4 Conclusions and future work

4.1 Conclusions

The previously outlined actions resulted in a preliminary identification of the industry's needs regarding standards and best practices for the development of software. According to the information gathered a document compiling best practices and tools for software development has been built with the participation of several DIH-HERO's project partners. This document will be completed with feedback from FSTP participants and other SMEs and partners from the DIH-HERO network.

4.2 Future work

In the following months, interviews and expert panels will be organised featuring members from associate and core partners. These experts will review, evaluate, and validate the present document, and contribute to future work package outputs, all of which will be gathered into a new, publicly available deliverable at the end of the project.

A workshop titled “Industry experience with healthcare standards” will be held within the European Robotics Forum 2022 (ERF2022) on June 2022. SMEs will be invited to present their experiences with healthcare standards and best practices, specifically regarding software development and systems interoperability. Other workshops will be organised with SMEs to collect the essential information needed to:

- Identify harmonisation needs on human-machine interaction, software development and interoperability best practices and standards.
- Propose a harmonization on ISBP to increase acceptance and reduce gaps.
- Reach consensus on discussed practice or classify it as disputed and feedback to expert groups.

The results of the interviews and workshops regarding software best practices will be included in a new version of this document.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

5 Annexes

5.1 Development models

5.1.1 Waterfall model

This model is based on linear sequence of the project activities, once one activity has finished, the next one can be started[48]. This type of model is not iterative. Figure 3 is based on the Royce's waterfall model, which is divided into the following phases:

1. System and software requirements.
2. Design, resulting in the software architecture.
3. Implementation or coding, resulting on the software itself.
4. Testing.
5. Maintenance or support once the system is installed.

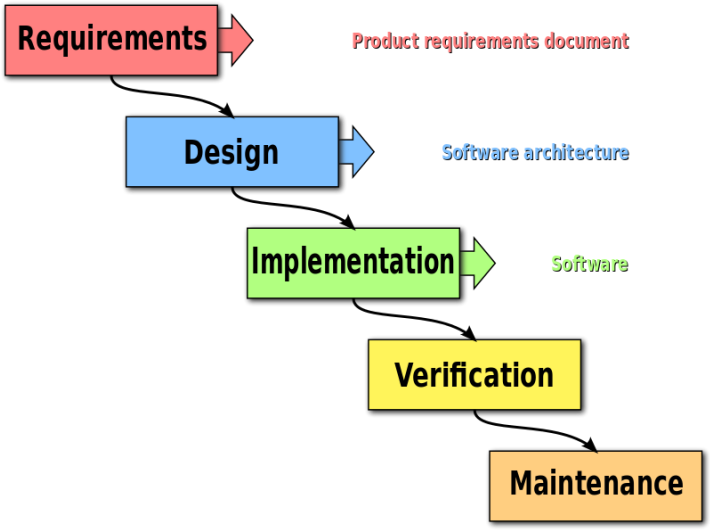

Figure 3. Waterfall Model of System Development. Peter Kemp / Paul Smith, CC BY 3.0 via Wikimedia Commons[49]

5.1.2 V-model (validation and verification model)

The V-model consists of several stages, each of which include a testing activity[50]. This model has a very high-quality control, which makes it quite expensive and time-consuming. Figure 4 shows a V-model proposal.

[48] https://en.wikipedia.org/wiki/Waterfall_model

[49] https://commons.wikimedia.org/wiki/File:Waterfall_model.svg

[50] https://www.scnsoft.com/blog/software-development-models

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

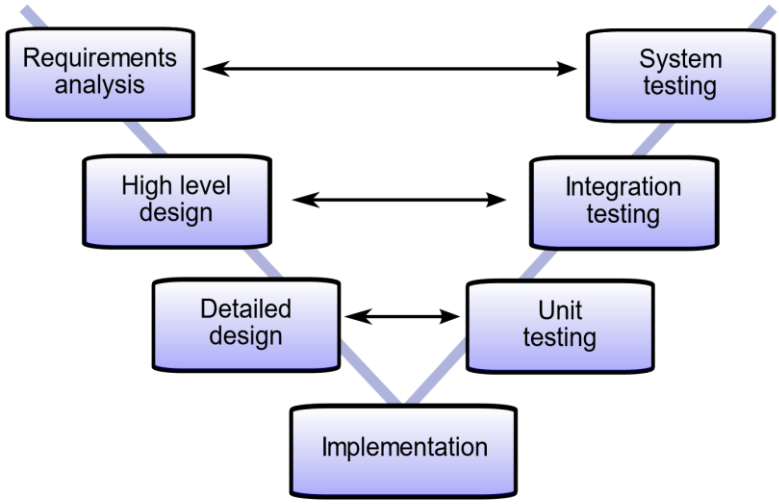

Figure 4. V-model for system development. Herman Bruyninckx, CC BY-SA 3.0, via Wikimedia Commons[51]

5.1.3 Scrum and Kanban: Agile models

The Agile models are incremental and continuous iteration approaches. One of the main objectives of these models is to identify the problems as soon as possible to make the corresponding changes on the requirements or the implementation.

Scrum[52] aims to deliver software releases within a shorter time frame by working in 'sprints' and orienting the work towards a defined goal.

On the other side, Kanban[53] is a highly flexible and very visual method for managing the project tasks. This methodology requires collaboration which means that every person in a team work together to achieve the specified result.

[51] https://commons.wikimedia.org/wiki/File:V-model.svg

[52] https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)

[53] https://www.wesquare.nl/scrum-vs-kanban-a-fair-comparison/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

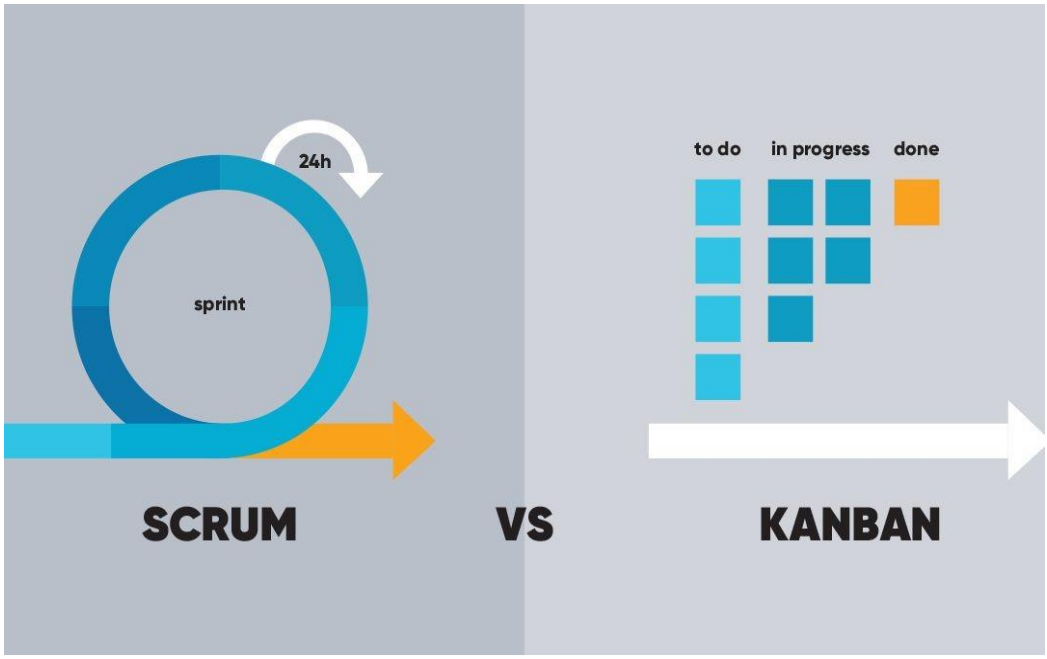

Figure 5. Scrum methodology vs. Kanban methodology[54].

Generally speaking, Agile models are convenient when the software objectives or requirements are not defined from the beginning. These are defined progressively through the gradual insertion of characteristics in each sprint or the selection of tasks in the Kanban model.

5.1.4 Lean model

Lean was proposed by the Toyota company[55] [56]. It is considered an evolution of the Agile model and includes a set of principles to improve the project development process, instead of a methodology. Those principles are:

1. Eliminate waste: remove everything that does not give us any kind of profit, for example code that is not used anymore.
2. Amplify learning: the working team must accumulate knowledge and share it.
3. Build integrity in: include any process of quality such as testing or bugs fixing from the early beginning.
4. Fast delivery: this approach is based on the iterative developments with the aim of getting feedback as soon as possible.
5. Empower your team: maintain the members motivated.
6. Delay in making decisions: the purpose is to take the important decisions which have high impact on the development as late as possible to avoid additional risks and to redo some work.

[54] https://www.wesquare.nl/scrum-vs-kanban-a-fair-comparison/

[55] https://lvivity.com/lean-development-key-principles

[56] https://es.wikipedia.org/wiki/Lean_software_development

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

7. Optimize the whole: the team must have an overview of the development process, the concept and the strategy when they are working in their tasks.

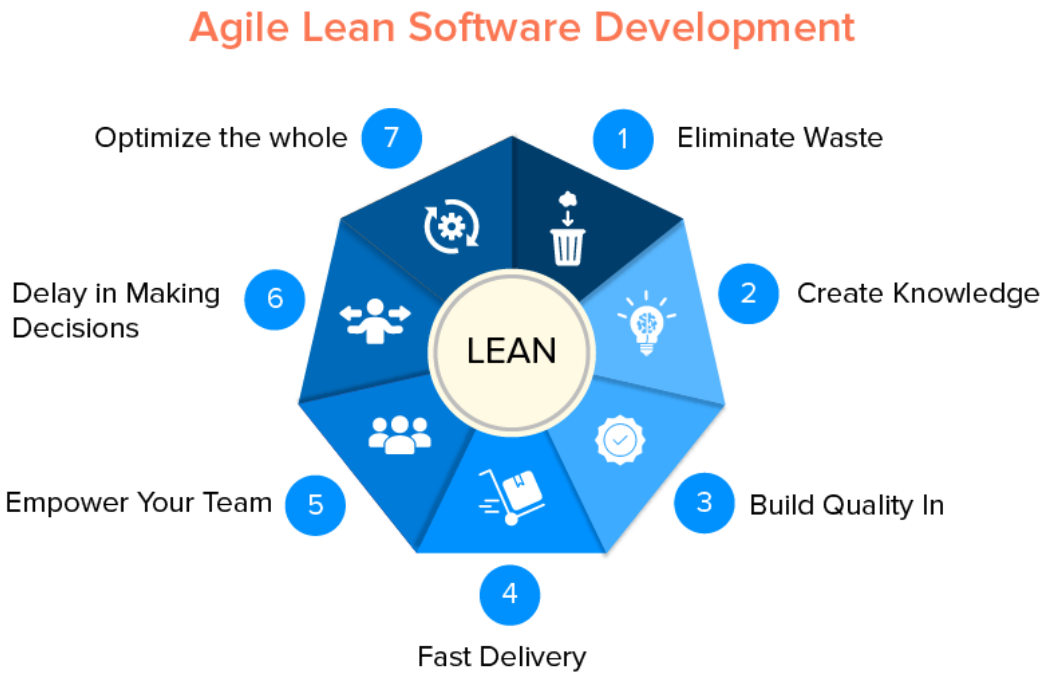

Figure 6. Lean Model for Software Development[57]

5.1.5 DevOps

This is a new approach that uses the different tools of agile procedures to help developers and operations staff work together. One of the aims of DevOps is to use tools for automatizing some of the tasks. This approach prevents errors and improves the integration in terms of quality and time. DevOps is composed of a set of interrelated steps, as shown in Figure 7.

[57] https://lvivity.com/software-development-methodologies

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

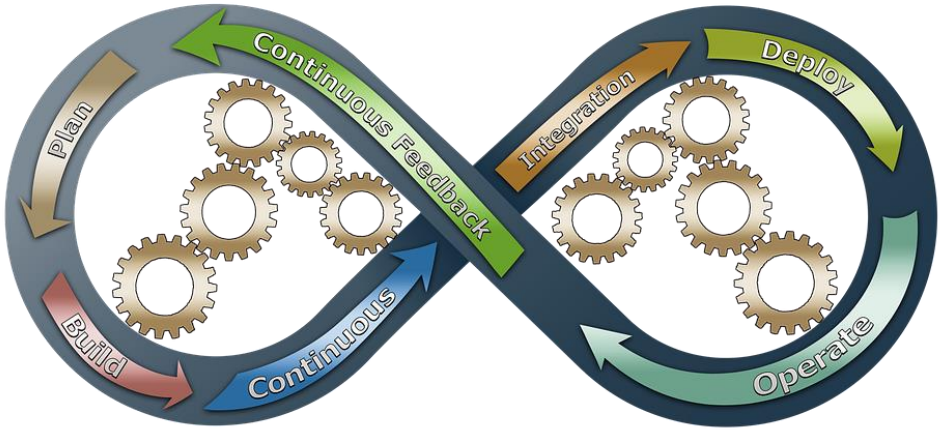

Figure 7. Devops model[58].

5.2 Testing concepts

5.2.1 Functional testing

5.2.1.1 Unit tests

Software Unit verification can be either actual unit testing or other forms of verifying the correctness of implemented units, like code reviews versus the specified behaviour.

Software unit testing is a test where interfaces to other software units are mocked. It shall ensure that the software unit functions correctly. The following criteria shall be taken into account:

- The test should focus on detecting functional problems of the corresponding software unit.
- The test should cover aspects that are not considered in the later verification and validation activities. Code review might play an important role since some problems which are difficult to detect via other test methods can be identified, e.g. race conditions, memory leak issues.

A unit test report must be created with the work done and the obtained results to keep track of the system's quality.

5.2.1.2 Integration testing

The aim of integration test is to ensure that all software units function together correctly as a software item. As appropriate, a software item should be mocked against other software items during testing. It is acceptable to perform integration test together with the system testing.

Anomalies are documented and covered by software problem resolution process.

[58] https://pixabay.com/illustrations/devops-business-process-improvement-3148393/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

5.2.1.3 End-to-end testing

This type of testing is used to validate a complete use case or functionality. End-to-end testing is also known as user testing. These tests are usually done manually, because the automation of them is highly time consuming.

5.2.1.4 Regression testing

Regression testing is used to verify that new code additions (to solve an identified bug, for example), do not have any negative effect on the previous development. This testing is usually performed during software maintenance.

5.2.1.5 Acceptance testing

Acceptance testing is usually linked to the contract with the final client. The contract defines the acceptance criteria for each development stage. For example, passing the 90% of the functional testing, or no security issues appear during testing. Actions should be taken to ensure that the contract is fulfilled.

5.2.2 Non-functional testing

5.2.2.1 Performance testing

Performance testing can be important, especially for web or apps software development. The aim of that testing is to verify the functional requirements defined at the beginning of the project for the software performance. The performance of a development will be evaluated according to the previously defined infrastructure. If it fails to fulfil the expected outcomes, decisions must be taken for instance to expand the infrastructure or review the code. The system quickness is measured, and this could be a handicap depending on the application.

5.2.2.2 Load testing

In this type of testing, the system load is evaluated according to some previously defined conditions. To perform load testing, different conditions of the system use must be considered, and the critical processes answer time must be measured.

5.2.2.3 Stress testing

In those cases where the system request is variable, the system answer with an unexpected requested is evaluated with stress testing. In case that the service fails during the stress testing, the infrastructure dimension must be reviewed.

5.2.2.4 Bottle neck analysis

All the systems involved in a service must be reviewed to identify that all of them are working alone, and that none of them have more load than expected. However, over dimensioning of the systems is not recommended. This analysis could help to establish a balance on systems infrastructure.

5.2.2.5 Security testing

Software security must be checked such as, for instance, the possibility of stealing the user's credentials or other relevant data, and including external malicious code or system blocking.

Some good practices on code development to avoid some of those risks are:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

- Avoid the access to code vulnerabilities by not restricting access to the source code.
- Backend restrictions to protect the rights and/or data validation.
- Not giving too much information on the error messages.
- Make special tokens or log files to protect the sensitive data.
- Use only frameworks that have been validated, and if there is any update for security issues, include the latest version.
- Control networks connections.

Some semi-automatic tools can be found to detect vulnerabilities, such as OWASP[59].

5.2.2.6 Scalability testing

The scalability is the capacity of increasing the hardware resources. The scalability plan will indicate the resource augmentation when the demand is also increasing. Scalability and performance are linked because the scalability proposed will be tested using the performance testing.

5.2.2.7 Portability testing

If the designed software needs to work on different devices (computer, tablet and phone) or under different operating systems or browsers, the functional, security and design aspects must be reviewed.

[59] https://owasp.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825003. This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.