Software Development and Maintenance Plan

This document summarizes development and maintenance activities.

Mapping of Standard Requirements to Document Sections

ISO 13485:2016 Section	Document Section
7.3.2 Design and Development Planning	1, 2, 3, 7

Classes	IEC 62304:2006 Section	Document Section
A, B, C	4.4.2 Risk Management Activities	1
A, B, C	5.1.1 a) (Processes)	1
A, B, C	5.1.1 b) (Deliverables)	1
A, B, C	5.1.1 c) (Traceability)	1
A, B, C	5.1.1 d) (Configuration and Change Management)	1, 5
A, B, C	5.1.1 e) (Problem Resolution)	1
A, B, C	5.1.2 Keep Software Development Plan Updated	1
A, B, C	5.1.3 Software Development Plan Reference to System Design and Development	2
C	5.1.4 Software Development Standards, Methods and Tools Planning	
B, C	5.1.5 Software Integration and Integration Test Planning	3, 8
A, B, C	5.1.6 Software Verification Planning	7
A, B, C	5.1.7 Software Risk Management Planning	1
A, B, C	5.1.8 Documentation Planning	6
A, B, C	5.1.9 Software Configuration Management Planning	5
B, C	5.1.10 Supporting Items to be Controlled	5
B, C	5.1.11 Software Configuration Item Control Before Verification	5

Classes	IEC 62304:2006 Section	Document Section
B, C	5.1.12 Identification and Avoidance of Common Software Defects	4
A, B, C	6.1 Software Maintenance Plan.	10

1 Relevant Processes and Documents

Please see the relevant processes for the following activities:

- Risk management activities incl. SOUP risks: SOP Integrated Software Development
- Problem resolution: SOP Problem Resolution
- Software development incl. deliverables, traceability, regular update of software development plan: SOP Integrated Software Development
- Change management: SOP Change Management
- SOUP List
- Usability engineering activities: SOP Integrated Software Development

2. Required Resources

2.1 Team

Role	Count	Responsibilities
Head of Development	1	Prioritizing tasks and technical oversight
Frontend Developer	2	Implementing Frontend Software Requirements
Backend Developer	1	Implementing Backend Software Requirements

2.2 Software

IEC 62304 Software Safety Classification The software safety classification for [enter device name] has been established as class [XXXX] per IEC 62304:2006/AMD1:2015 based on the decision-making process outlined in table 3 and in paragraph 4.3 of the norm. A malfunction of, or latent design flaw in the software device may lead to situations with unacceptable risks [for example: false-positive and false-negative diagnosis, resulting in unnecessary interventions or missed necessary interventions]. This excludes software safety class A. Serious injuries or death, however, can be ruled out because [XXXX]. Considering these risk control measures external to the software system, safety class C can be ruled out, resulting in class B.

Measuring Function The [enter device name] does not include a measuring function, as described in EU Regulation 2017/45 and relevant regulatory guidance documents. The definition of MEDDEV 2.1-5 for measuring functions does not apply because [XXX].

Combination With Other Products To achieve its intended purpose, the [enter device name] is intended to be used in combination with [for example: MRI/CT scanners that produce imaging data]. Specifications for compatible equipment are described in the List of Software Requirements as well as in the Instructions for Use. Relevant verification and validation tests will be added to the documentation.

Product Lifetime The software's lifetime is established to be [for example: three years]. This is what is expected to be the maximum time until the implementation of a significant change, by which the manufacturer is able to react to the relevant changes to the software device environment, such as SOUP changes, cybersecurity innovations, or the evolving technological or medical state of the art.

Programming Languages

List the languages you'll be using, including compiler and language versions.

Name	Version
Python	3.8

Development Software

List software used to support development, e.g., IDEs.

Name	Version
PyCharm	2020.1.4

2.3 System Requirement / Target Runtime

List your target runtime(s).

Name	Version
CPython	3.8

Specify system requirements, e.g., the minimum specifications of the server / compute instance you'll be running your software on

Minimum system requirements:

- *Server-grade dual-core CPU, e.g., Intel Xeon E3-1230 v5 or higher*
- *4 GB of RAM*
- *1 GBit/s up- and downlink*
- *20GB SSD storage*

3 Design Phases

The 13485 requires you to specify "Design Phases". Here are some suggestions which you could use.

Title	Estimated Completion Date	Description	Review method
Specification			Software Requirements Checklist
Implementation			Code Reviews
Testing			System Test
Validation			Usability Evaluation
Release			Release Checklist

4 Avoiding Common Software Defects Based on Selected Programming Technology

Discuss how your selected programming technology may introduce risks and how you plan to avoid them. With modern, dynamically-typed languages, an obvious risk is that you encounter runtime exceptions. So you could argue that your test coverage is great and compensates for that. You could also link to your risk analysis here if you analyse those risks further.

5 Configuration Management and Version Control

Describe which version control software you're using (probably git, like all human beings on this planet right now, except enterprise developers). Also describe your branching model, i.e., how your developers create branches during development, how you name them and how you merge them (pull requests? merge commits? squash before?). Your code review will be described in the next section.

Importantly, describe which things (code, build files, etc.) are put in version control. Describe how you name versions and how you tag them. Your goal should be that you can retrieve an old version and build it. Why? Something with a newer version may go wrong (harm patients) and you may need to roll back.

git is used as version control software. All source code and build files are committed to version control.

When implementing software requirements, developers create a new branch starting at master. During development, developers may create intermediate commits on this development branch.

When implementation is completed, a new merge commit to master is created.

This is also the activity which constitutes integration of software units.

For each release, the goal is to be able to uniquely identify it and retrieve all relevant files (code, configuration files like build scripts, SOUPs, etc.) at any time in the future.

When a new software version is released, its commit is tagged in git. The tag is constructed by adhering to semver (semver.org) 2.0.0 which results in a version of format MAJOR.MINOR.PATCH, e.g., 1.0.0.

6 Documentation Activities

> Describe your policy on what should be documented while you develop software. Maybe you want to require your developers to document all methods which are private. Maybe you want to keep an up-to-date software architecture diagram in the repository, etc. Make sure to mention how traceability between Software Requirements and Tests is maintained.

7 Implementation Verification Activities

> Describe verification activities, e.g. code review.

For each pull request, a code review is performed by a team member who was not the main author of the code under review. The code review is only marked as "approved" if the code complies with the code review criteria. This is:

- *Code fulfils the software requirements*
- *Adherence to PEP8 Style Guide*

8 Software System Test Activities

> Describe software system test activities. This could be continuous integration which is triggered by opening a pull request (e.g. Travis CI, Circle CI). Describe what is tested and how that automated system works.

Integration tests are included in software system tests.

9 Validation Activities

Validation is carried out as formative and summative usability evaluation as described in the software development process. A usability evaluation file (plan, protocol and report) will be prepared.

10 Maintenance Activities

> Describe how often you check SOUP issue trackers and how you document them.

SOUP issue trackers are checked at least once every 6 months. The verification date is updated in the SOUP list accordingly.

Template Copyright openregulatory.com. See template license.

Please don't remove this notice even if you've modified contents of this template.