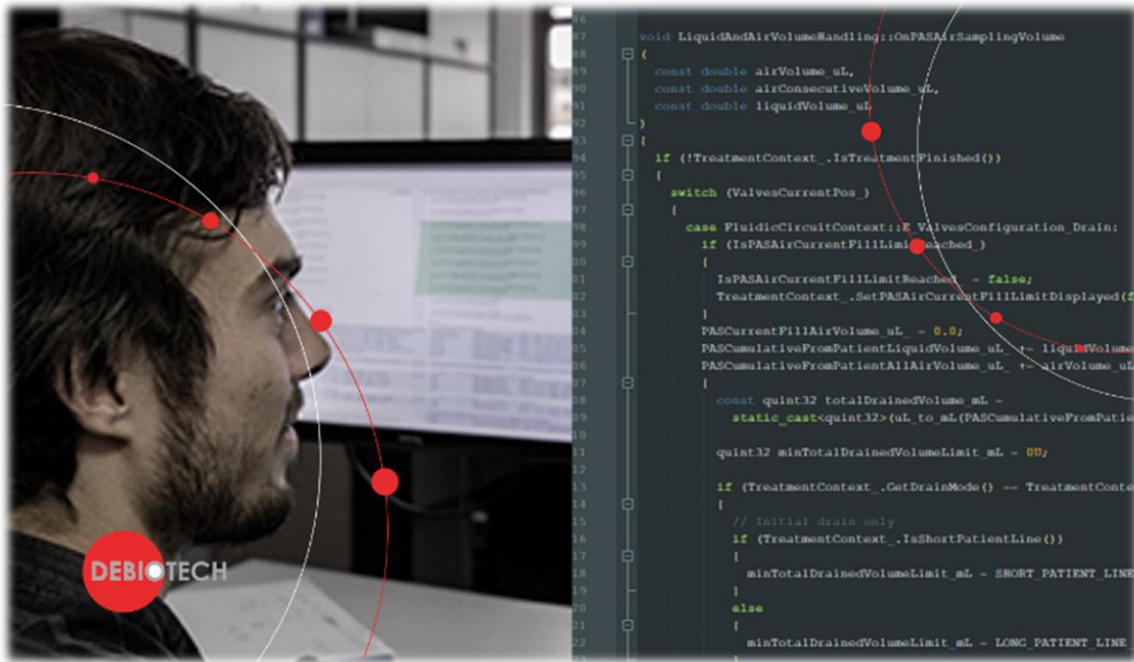


# Good practices for Medical Software development environment setup



## 1. Goal of this publication

The goal of this publication is to help you in the setup of your medical software development environment. The earlier you start with good practices application the more value and time you will gain from it.

## 2. Targeted audience

The information gathered in this publication should be particularly useful for:

- Head of Software Development Team,
- Software Project Managers,
- DevOps team,
- Software developers.

### 3. Table of content

1. Goal of this publication .....	1
2. Targeted audience .....	1
3. Table of content .....	2
4. Introduction.....	3
5. Good practices .....	3
5.1. Source code management platform .....	3
5.2. Define coding guidelines .....	7
5.3. Setup automatic code formatting tools.....	8
5.4. Pointers and references management .....	9
5.5. Exceptions and errors management .....	9
5.6. Use of event logging .....	10
5.7. User input validation & sanitization .....	10
5.8. User of compiler warnings and errors.....	10
5.9. Testing tools .....	11
5.10. Continuous integration .....	12
5.11. Management of external libraries.....	13
5.12. SBOM and CVE generation .....	13
5.13. Static Application Security Testing Tools.....	15
5.14. Issues & Tasks management system.....	16
5.15. Systems for technical documentation writing.....	18
6. Applicable regulatory landscape .....	19
7. Authors.....	19
8. Next steps .....	20

## 4. Introduction

Software innovation often starts with the writing of a significant number of lines of code to test the feasibility of an idea. Code clarity, structure and testing may not be perceived as priorities in this phase. Once first positive results are obtained it is easy to fall into a loop where more and more code and features are added but no significant effort is put on the code clarity, maintainability, and robustness. This can easily result in a chaotic situation where it becomes very hard for a developer to update or even just fix a piece of code written by someone else or even worse written by themselves some months ago.

The more developers you have the higher the challenge to ensure good productivity. To allow your company to follow a fast innovation track, you need to start as soon as possible with good practices and collaborative software development culture.

In this publication, Debiotech experts will describe what they perceive as the essential tools and practices in software development in general, and for medical software development specifically.

## 5. Good practices

### 5.1. Source code management platform

#### 5.1.1. Basic code repository management

The source code management platform is the cornerstone of your development environment. It is the basic tool for any developer even for single individual teams. It allows you and your team to share and manage the different versions of your source code. Finally, it ensures a decentralized backup to avoid the loss of long hours of work in case of destruction or inaccessibility of your local data.

Multiple platforms exist, including:

- Git
- SVN
- Mercurial

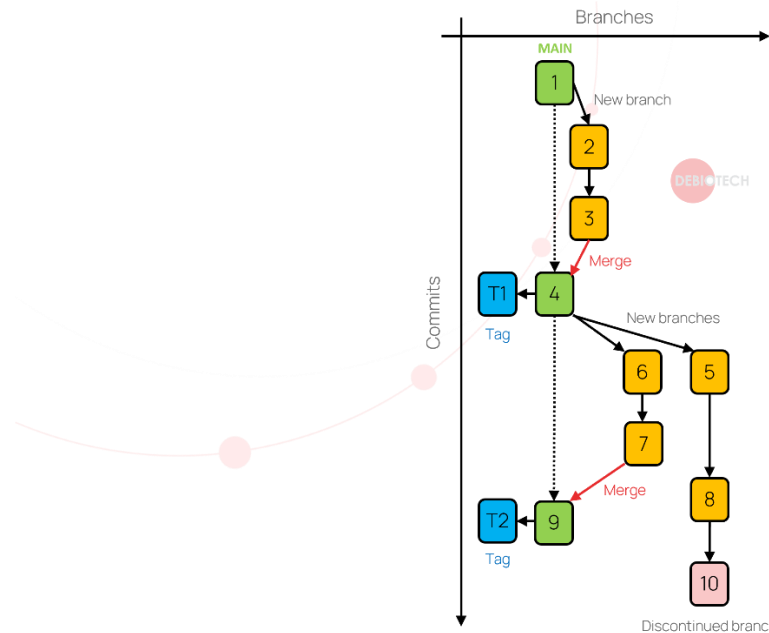


Figure 1. Illustration of basic source code management platform item & features

Basic items are available within those platforms, including:

- **Repositories:** they are the basic containers of your source code and contain a collection of files of different versions of a project.
- **Branches:** they are the main item allowing you to follow the different versions of your code. A branch is an evolving entity that changes through commits of code. Branches can be created and deleted; some will exist all along the lifecycle of your product. More details about branch types and structure are provided in the Chapter 5.1.3.
- **Tags:** they allow to identify specific versions of your code. A tag is a static item associated to a given branch at a given point in time (and therefore to a specific commit). It allows to quickly find important versions of your code as for example: versions used for clinical trials and the ones used for your deployed product.

Basic actions are available within those platforms, including:

- Basic actions associated to repositories:
  - Create: it creates from scratch the centralized repository to store the various branches of your code.
  - Fork: it creates a deep copy (complete and independent copy) of the centralized repository.
  - Clone: it creates a local copy of your centralized repository.

- Basic actions associated to branches:
  - Checkout: to get a local copy of a given branch on your computer.
  - Create: It creates a branch from an original one and keeps track of its modifications independently from the ones of the original one.
  - Commit: to put local changes within the corresponding branch of the centralized repository.
  - Merge: to update the content of a target branch with the modifications performed on another branch. This does not copy and replace the target branch but only push the modifications performed on the merged branch. Therefore, those two branches can have evolved independently before the merging and the modifications brought on both branches will be combined. This can result in conflicts (if modifications are brought to the same piece of code). Conflicts resolving is not always straightforward but will not be developed further within this publication.
  - Pull request is described further in Chapter 5.1.2.
  - Other features are available but will not be described further in this publication.
- Basic actions associated to tags:
  - Create tag: associate a specific commit with a tag that allows to quickly identify it and retrieve the associated content.
  - Get commit associated to a tag: get a local copy of the content associated to the commit to which the tag is associated.

### 5.1.2. Advanced code repository management tools

Some advanced solutions exist to help you and your team in the management of your source code repository and to create interfaces with other platform such as task/issue management systems (refer to Chapter 5.14) and continuous integration platforms. And finally, they provide user friendly code version comparison tools that allow to quickly identify differences between branches or commits.

Those advanced tools also give the possibility to define pre-requisites before validating a commit in a controlled branch (development branch) for example:

- Pull request review and validation by different persons
- Continuous integration tests all successful.

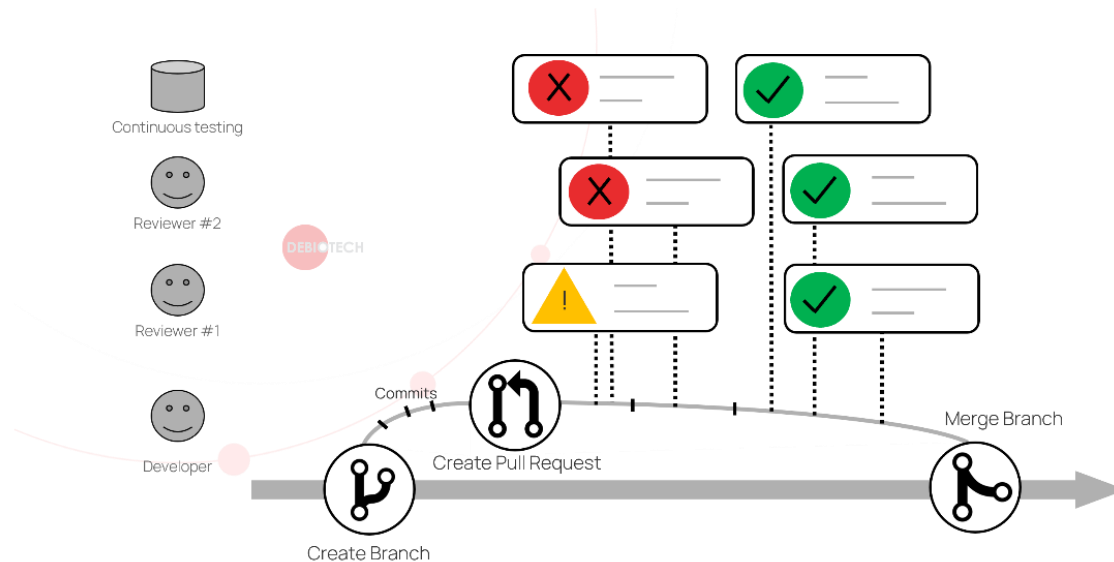


Figure 2. Pull request & Continuous integration principles illustration

Commonly used platforms include:

- Github
- Gitlab
- Bitbucket

Debiotech recommends you to:

- Integrate an issue management system to record features, user stories, epics and bugs (refer to Chapter 5.14)
- Put in place peer reviews to review and approve pull requests.
- Implement a continuous integration environment, including :
  - Static code analysis (refer to Chapters 5.12 & 5.13),
  - Automated tests, ideally including hardware in the loop for embedded systems (refer to Chapter 5.10).

### 5.1.3. Use of branch types and tags

During the development of your software multiple branches will be created. Each branch has a different purpose and different characteristics.:

- You want to have branches that are stable and well tested for example for demonstration to your customers or investors.
- Other branches should be opened and merged quickly to allow a developer to work on a new feature freely before adding it to a more controlled branch.
- You will need dedicated branches that will correspond to your release candidates.

- Some branches will be there to provide hot fix for an identified bug on a controlled branch.
- Finally in the medical device industry you will also need a branch for fully tested (verification and validation) version that can be used for clinical studies or as products.

To identify the different versions of your product, use tags on your main branch (the branch with the highest level of control).

Debiotech recommends you to:

- Define different branch types,
- Adapt the automatic testing you perform on a branch depending on its type (no need to perform complete testing for commit on a new feature branch),
- Clearly differentiate branches based on the level of control and security you want to have on their content.

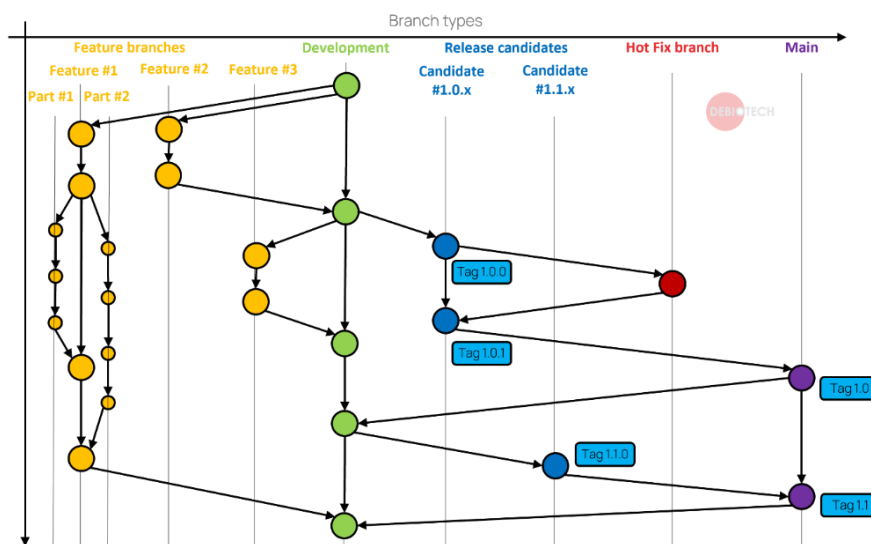


Figure 3. Illustration of possible branch type structure

## 5.2. Define coding guidelines

For clarity, homogeneity, and ease of review, you must establish common rules for your own code, including:

- Naming conventions for variables, functions, and files,
- File and folder organization,
- Format for comments,
- Code indentation and formatting.

Coding guidelines are specific to programming languages and shall not be applied to external libraries. For the code developed by external partners, you should request them to respect the guidelines you established or review and validate their coding guidelines. Debiotech recommends that you choose to apply the same coding guidelines for all projects in the same programming language. And perhaps a naming convention to be used regardless of programming language would be to include the unit information on all variables that contain physical units (distance, weight, pressure, temperature, speed, etc.).

Debiotech does not recommend any specific naming conventions but here is an example of a set of simple rules:

Name	Convention
Class name	Should start with uppercase letter and be a noun (e.g., String, Color, Button, System, Thread, etc.)
Interface name	Should start with uppercase letter and include an adjective (e.g., Runnable, Remote, ActionListener, etc.)
Method name	Should start with lowercase letter and include a verb (e.g., performAction (), main(), print(), etc.)
Variable name	Should start with lowercase letter (e.g., firstName, orderNumber, etc.)
Package name	Should be in lowercase letter (e.g., java, lang, sql, util, etc.)
Constant name	Should be in uppercase letter (e.g., RED, YELLOW, MAX_PRIORITY, etc.)

Table 1. Simple illustration of naming convention

### 5.3. Setup automatic code formatting tools

Having a homogeneous format for your source code is a way to simplify the work of your developers and increase their productivity. They will be at ease with consistent formatting while heterogeneous formatting will use some of their energy just because there is one space more or an additional empty line compared to what they are used to.

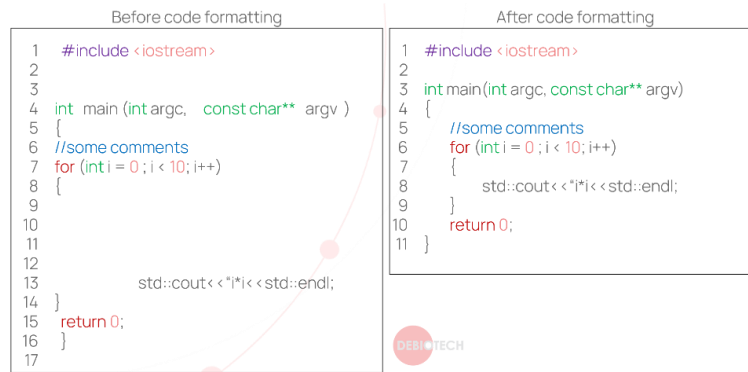
You can define simple formatting rules and ask everyone to follow them, but why wasting energy on this when automatic formatting solutions exist and are simple to integrate in your development environment:

- Clang-format
- Astyle
- Uncrustify

Debiotech recommends you to:

- Integrate such automatic formatting tools,
- Run them automatically at every compiling request on modified files.





```
Before code formatting
1  #include <iostream>
2
3
4  int main (int argc, const char** argv )
5  {
6  //some comments
7  for (int i = 0 ; i < 10; i++)
8  {
9
10
11
12
13      std::cout << "i" << std::endl;
14  }
15  return 0;
16  }
17

After code formatting
1  #include <iostream>
2
3  int main(int argc, const char** argv)
4  {
5      //some comments
6      for (int i = 0 ; i < 10; i++)
7      {
8          std::cout << "i" << std::endl;
9      }
10     return 0;
11 }
```

Figure 4. Simple illustration of added value of code proper formatting

## 5.4. Pointers and references management

To optimize memory size and time to pass parameters to one function or to an object, pointers and references are very useful. However, for critical code it is recommended (for example by MISRA guidelines) to avoid the use of dynamic memory allocation and therefore of pointers to dynamically allocated memory.

Therefore, Debiotech highly recommends to:

- For critical code (e.g. Class C software components as defined in IEC 62304): use references and static memory allocation instead of dynamic memory allocation.
- For non-critical code: use smart pointers to help your developers in the optimization of allocated memory but use references to pass inputs to functions or objects.

## 5.5. Exceptions and errors management

When creating a new object or function, the developer usually has in mind what values and behaviors are expected. It may seem so obvious that no control is done during code execution. However, another developer might modify the code or create new code that will take an unexpected path and result in non-expected code execution, potentially leading to software crash. In a way, it is the ideal scenario as this crash will push you to identify the root cause and fix it. In the worst case this unexpected behavior is not perceived and results in inaccurate performances that might endanger patients. Integrating simple exceptions and errors throwing mechanism is a way to avoid those unexpected behaviors and to help efficiently the developer to identify what's wrong. However, it is common in medical software development to forbid the use of exceptions to minimize complexity, awkwardness, and performance overhead.

Debiotech recommends to:

- Develop control mechanisms and propagate the information through errors.
- Associate errors with clear messages allowing to quickly identify their origin.
- Test your errors in your unit test and ensure 100% test coverage.

## 5.6. Use of event logging

To ensure traceability of software and hardware events, it is a common practice to use event logging. It provides a standard and centralized way for applications and operating systems to record important software and hardware events. The event logging service records events from various sources, components or objects and stores them in a single file: the log file.

Debiotech recommends to:

- Define different types of events to be logged ( Debug, Warning, Error)
- For each component and for the different branch types log only the level of events that is meaningful for this specific branch type.

## 5.7. User input validation & sanitization

Input sanitization is a cybersecurity measure of checking, cleaning, and filtering data inputs from users, APIs, and web services of any unwanted characters and strings to prevent unexpected behavior, crash or even injection of harmful codes into the system.

Debiotech recommends to:

- Develop simple mechanisms to check data inputs validity: check length, characters, string, and values.
- Block data inputs when invalid content is identified or filter out the unexpected content and throw an exception to keep track of this event.

## 5.8. User of compiler warnings and errors

Congratulations, after adding an important feature or additional external libraries, you finally get to the point where you have no more compilation error! Don't stop fighting and remove now all compilation warnings. They are usually the sign that you do not have full control on your code cleanness, reliability, and robustness.

Some of those warnings might look acceptable but having a warning free source code is the only way to quickly identify new warnings that will block you further in the development. If you already have 10 compilation warnings, having an additional one might be perceived as a minor issue. While if you introduce compilation warnings in a warning free code, you will quickly see them and process them.

Debiotech recommends to:

- Do not mask compiler warnings,
- Fix compiler warnings.

## 5.9. Testing tools

When adding new features or objects to your code, adding corresponding tests should be mandatory. Errors and mistakes are human. They will always happen, the earlier you identify them the less impact they will have on your productivity. Having a systematic testing approach allows you to drastically improve work efficiency by identifying where and when an error is introduced.

Multiple test level should be defined and run with different frequency or triggers. Typical test levels are illustrated in the next figure:

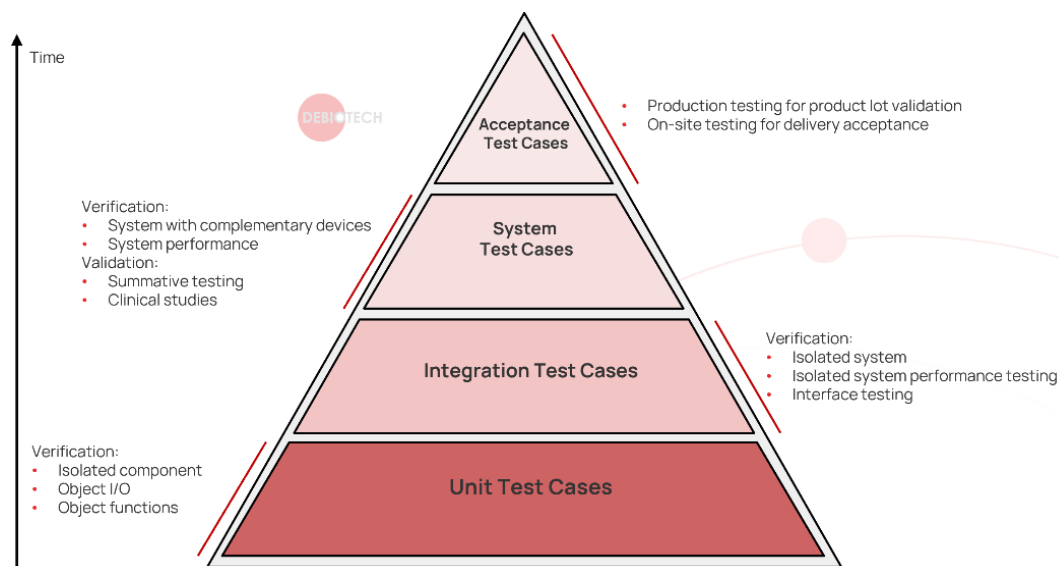


Figure 5. Tests hierarchy illustration

In addition to develop your own testing tools Debiotech recommends using existing solutions that can be easily tuned to match your needs. Here is a list of some of those tools:

- VectorCAST

- Google test
- QtTest

Debiotech recommends you to:

- Avoid committing new modifications/features/object without providing its respective unit test.
- Add integration testing of a new object/feature at least when you expect to use this new object/feature in your code.
- Establish and perform system testing. This is the minimum level you should implement, even for software systems that aren't considered critical.
- Add performance testing, when relevant.

## 5.10. Continuous integration

The only way to ensure systematic testing at any commit or when committing into critical branches (development, release, or main) is to integrate it in your development environment using a continuous integration tool. In some cases, this systematic testing cannot be performed at every commit because of the necessary time to perform those tests. In this condition, you can define various set of tests to perform at different frequency/triggers.

Multiple platforms exist to support continuous integration. The most used platforms include:

- GitLab CI/CD
- Bamboo
- Jenkins
- CircleCI
- Azure Pipelines

Debiotech recommends you to setup your continuous integration platform to:

- Block a commit if build and compilation do not work.
- If you target multiple platforms or multiple versions, include all the targeted platforms in this build and compilation verification.
- Systematically perform unit testing at commit into critical branches (if it is not too much time consuming)
- Ideally perform systematically integration testing at commits into critical branches (again if the total duration of testing is reasonable).
- For longer test runs (full integration and performance testing), perform them at night, when the activity of your software development team is lower.

- Daily review the results of those tests to ensure no test have been broken by the commits of the previous day.

### 5.11. Management of external libraries

“Never reinvent the wheel and use available libraries” is a great advice for fast innovation. Nevertheless, take only the wheel you need and not the entire car! You can quickly get polluted with useless libraries. Build and compilation time quickly increases with the number of libraries and the time you spend in cybersecurity management is exponentially impacted by the number of libraries.

Debiotech recommends you to:

- Identify your needs and the available solution and check the associated licenses and the known vulnerabilities before integrating a specific library.
- Keep track in a Software Bill Of Materials (manual or ideally automatic) of the libraries you use, their version, their description and the reasons why you use them.
- Every request for a new library to be integrated by a developer should be reviewed and challenged before being approved.



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <bom version="1" xmlns="http://cyclonedx.org/schema/bom/1.0">
3   <components>
4     <component type="library">
5       <name>alsa-lib</name>
6       <version>1.0.29</version>
7       <licenses>
8         <license>
9           <id>LGPLv2.1+</id>
10        </license>
11      </licenses>
12      <modified>false</modified>
13    </component>
14    <component type="library">
15      <name>bluez_utils</name>
16      <version>4.101</version>
17      <licenses>
18        <license>
19          <id>LGPLv2+ LGPLv2.1+</id>
20        </license>
21      </licenses>
22      <modified>false</modified>
23    </component>
```

Figure 6. Example of an automatically generated Software Bill Of Materials

### 5.12. SBOM and CVE generation

Quickly your software project will have a level of complexity that won't allow you to manually follow the libraries you use, their version, the associated license, and other metadata. Reliable automatic code analysis tools exist and can automatically do this work for you and bring you important additional features as Critical Vulnerabilities identification. Those tools allow you to generate the Software Bill Of Materials (SBOM) as well as the list of applicable Common Vulnerabilities Enumeration (CVE).

For proper Cybersecurity management those tools and documents are a must have and shall be used early in the development process, on a regular basis AND during the entire lifetime of the product (even long after release to market). Those tools will allow you to develop your Cybersecurity BOM, requested by the FDA in its latest [cybersecurity guidance](#). You can also perform this search for vulnerabilities manually using [CVE](#) or [NVD](#) database for example.

Major build systems (e.g., [Buildroot](#), [Yocto](#)) are able to generate SBOMs which must preferably be in standard format like [CycloneDX](#) or [SPDX](#). Alternatively, tools that will automatically scan the projects dependencies can be used (e.g., using makefiles for C/C++ projects or pip requirements for python).

Some of the well-known tools are [WhiteSource](#) or [Black Duck](#).

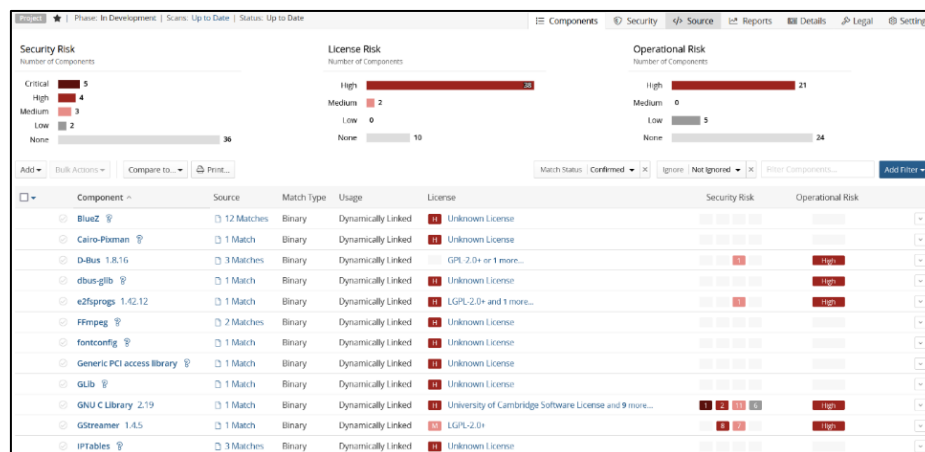


Figure 7. Sample output of Black Duck SCA tool

Debiotech recommends you to:

- Quickly integrate SBOM automatic generation tool in your software development environment to follow use libraries, their version, their license, and other metadata.
- Quickly integrate CVE automatic identification tool into your software development environment to start rising awareness on cybersecurity management within your team.
- Once your software development environment is fully setup, start investigating more in details:
  - Requirements for proper medical software development management
  - Requirements for Cybersecurity for your connected devices.

Debiotech will regularly publish expert opinions on those topics.

## 5.13. Static Application Security Testing Tools

### 5.13.1. Introduction

A very good practice to automatically detect bugs during development, also recommended by UL 2900-1 18.1, is to regularly scan the code using a Static Code Analysis tool (SCA, although SAST which stands for “Static Application Security Testing” should be preferred to avoid confusion with “Software Composition Analysis”).

SAST tools are available for any programming language. They analyze the source code (and/or compiled code) without running it on the device to detect the most common errors and security flaws.

A lot of different SAST tools are available on the market, whether they are open source like [Clazy](#) or [cppcheck](#), or commercial like [Fortify](#) or [CodeSonar](#). Each of them comes with pros and cons and must be selected based on the project needs and integrated in the software development workflow as early as possible. To help in this process, a list of SAST tools is maintained by the OWASP (click [here](#) to see the list).

### 5.13.2. How to Select your SAST Tool?

First thing to consider is selecting a tool compatible with the used programming language and ideally the used libraries and frameworks (this may avoid a lot of false positive).

The tool must be able to detect a lot of error types, depending on the underlying technology, like SQL injection if a database is used, hardcoded passwords for a server application, or the well-known buffer overflows or null pointer dereference in unmanaged languages like C and C++.

Being compliant with the common bug classification types, like [OWASP Top 10](#) or [CWE Top 25](#) and coding guidelines like [MISRA C](#) is a real plus. Some tools like [Perforce Klocwork](#) or [Parasoft C/C++test](#) also come with standard compliance certification for IEC 62304, which will exempt you from validating the tool and automatically generate the required documentation.



CWE Category	Priority			
	Critical	High	Medium	Low
CWE ID 120	1	0	0	0
CWE ID 125	0	0	0	1
CWE ID 129	1	0	0	1
CWE ID 131	1	0	0	1
CWE ID 195	0	1	0	0
CWE ID 252	0	0	0	1
CWE ID 253	0	0	0	1
CWE ID 328	0	0	0	1
CWE ID 398	0	0	0	1
CWE ID 401	0	421	0	0
CWE ID 476	0	1	0	0
CWE ID 561	0	0	0	126
CWE ID 615	0	0	0	75
CWE ID 690	0	0	0	1
CWE ID 754	0	0	0	1
CWE ID 787	1	0	0	0
CWE ID 805	0	0	0	1

\* Reported issues in the above table may violate more than one CWE requirement. As such, the same issue may appear in more than one row. The total number of unique vulnerabilities are reported in the Issues by Category table.

Figure 8. Fortify SAST is compatible with CWE classification

The next step is to ensure that the tool integrates smoothly with your development workflow. Some questions that should be asked at this stage:

- Is it possible to run it automatically from the continuous integration server?
- Can it be integrated in the IDE to provide easy to understand feedback to the developers?
- Can generated reports be customized?

### 5.13.3. Debiotech recommendations

Debiotech recommends you to:

- Identify the SAST tool that fits the best to your needs,
- Use a SAST tool as early as possible within your development process,
- Integrate the SAST tool within your continuous integration server,
- Integrate the SAST tool within your IDE to provide direct feedbacks to your developers,
- Customize SAST tool reporting to your needs.

## 5.14. Issues & Tasks management system

To ensure completeness of your project and efficient development you will need to define multiple tasks and assign them to members of your team. It is important to make the distinction between project management tasks (for example: organize risk analysis session, write down user manual, ...) and development tasks (for example: add a text field to display patient name). Both of those types of tasks might benefit directly from task management software, but it is especially true for development tasks where the number of tasks increases quickly and where the



level of details of the task (or sub-task) shall be important to ensure alignment between expectations and deliverable content.

To ensure efficient collaborative development, you will quickly need tools to ensure task management and follow-up. A list of tasks within an Excel document or any similar document will quickly show its limitation. This document will require giving editing rights to multiple users to update frequently the task list and the tasks status. This quickly increases the risk of having unexpected edition and loss of information and will limit the additional features you benefit from, as for example: notification when a task changes status.

Commonly used task management platforms include but are not limited to:

- Jira,
- GitLab,
- Azure DevOps,
- Polarion.

For those reasons, Debiotech highly recommends you to:

- Setup a task management tool. No need for fancy brand new solution. Simple task management platform already provides a significant set of features that will increase the productivity of your team.
- Select a tool that allows:
  - To create a task and to divide it into sub-tasks.
  - To group task into sprints, features or targeted release versions.
  - To define different task status. For example: Opened, assigned, on-going, under review, closed and abandoned.
  - To define different task priority levels (for example: High, medium and low), expected delivery date and effective delivery date. No to be used as a support to blame anyone but to be able to continuously improve your planning skills.
  - To block task deleting. Use “Abandoned” status instead to keep track of what was thought as necessary or useful at some points.
  - To define who has the rights to assign task to someone else. The rules there should depend on the way you want to work. It is sometimes useful to limit task assignment only to the software development project manager. Within other teams, you might want each of your developer to be able to assign tasks to other developers even if the

software development project manager might want to change this assignment later.

- To generate reports at various times to have an overview of your task management activities within a given period.
  - To assign tasks to given software releases.
  - Ideally select a tool that can integrate with a code repository management platform to allow you to have automatic links between code commits and tasks.
- Ensure your team updates the status of their tasks systematically and immediately and not once a day or even worst once a week. Nothing worst to have not fully updated tasks status.
  - Congratulates your team on major task or task group achievements!

Finally, a last point that can be useful is to integrate your task management platform with your technical documentation management platform to make links between specifications or use cases and tasks and software versions.

This might allow you to have a complete traceability between your specifications and test cases and your software versions. However, this is not mandatory and might create more execution challenges than solve any real problem. Debiotech do not recommend trying to develop such solution unless you really perceive a need there.

## 5.15. Systems for technical documentation writing

A critically important activity for software and especially for medical software development is technical documentation writing. Medical device regulations impose multiple requirements on the content and structure of such documentation. The way to structure such documentation, the items characteristics and the different tools helping you in such writing will be the subject of a specific publication by Debiotech.

## 6. Applicable regulatory landscape

This publication does not aim at answering completely to a specific standard or regulation but provide recommendations on multiple aspects that should be established early on in your development to ensure the efficiency of your team. However, it covers directly or indirectly multiple requirements from following standards and regulations:

- Europe: MDR
- US: CFR Title 21 Part 820,
- International: ISO 13485, IEC 62304, UL-2900-1/2

## 7. Authors

This publication has been written and reviewed by:



Rémi Charrier  
Business Development Director  
[r.charrier@debiotech.com](mailto:r.charrier@debiotech.com)



Laurent Colloud  
Software Project Manager  
[l.colloud@debiotech.com](mailto:l.colloud@debiotech.com)



João Budzinski  
R&D Director  
[j.budzinski@debiotech.com](mailto:j.budzinski@debiotech.com)



Gilles Forconi  
Software Quality Manager  
[g.forconi@debiotech.com](mailto:g.forconi@debiotech.com)

## 8. Next steps

Debiotech is glad to have the opportunity to share its knowledge with innovative companies from the MedTech industry. Your feedbacks on this publication are welcome and will be used to update it or to create new publications on topics you care about.

Continue your education on medical device development by:

- Accessing Debiotech historic publications:  
<https://www.debiotech.com/news-grid/>
- Following Debiotech on LinkedIn to be notified on new publications:  
<https://www.linkedin.com/company/debiotech-sa>
- Contacting us to ask a question or request personalized support:  
[contact@debiotech.com](mailto:contact@debiotech.com)

Debiotech would be proud to be your partner and support you with:

- Medical device design & development services:
  - Software: Digital Health, Firmware, Embedded, SaMD
  - Electronics: Design, Verification and Validation
  - Mechanics: Design for micro-fabrication & fluidics systems
  - Supply chain development and optimization
- Support in medical innovation management:
  - Market analysis and segmentation
  - IP management
  - Business plan consolidation
  - Partnership development

