Towards RegOps: A DevOps Pipeline for Medical Device Software

Henrik Toivakka[1,3][0000−0003−4906−4614], Tuomas Granlund[2,3][0000−0003−3955−0926], Timo Poranen[3][0000−0002−4638−0243], and Zheying Zhang[3][0000−0002−6205−4210]

[1] Mylab Oy, Tampere, Finland henrik.toivakka@mylab.fi
[2] Solita Oy, Tampere, Finland tuomas.granlund@solita.fi
[3] Tampere University, Tampere, Finland
{timo.poranen,zheying.zhang}@tuni.fi

Abstract. The manufacture of medical devices is a strictly regulated domain in the European Union. Traditionally, medical software compliance activities have been considered manual, document-centric, and burdensome. At the same time, over the last decade, software companies have maintained competitiveness and improved by relying on essential practices of DevOps, such as process automation and delivery pipelines. However, applying the same principles in medical software can be challenging due to regulatory requirements. In this paper, we utilize a systematic approach to align the essential medical device software regulatory requirements from the standards IEC 62304 and IEC 82304-1 and integrate them into the software delivery pipeline, which is the main contribution of our work. The outcome supports practitioners to establish more efficient software delivery models while maintaining compliance with the medical device standards.

Keywords: Medical device software · medical device standards · regulatory compliance · DevOps · RegOps

1 Introduction

The EU regulation strictly controls the manufacturing of medical devices. In order to place a medical device on the EU market, the manufacturer must prove the conformity of the product with the applicable EU regulatory requirements. With a CE mark, the manufacturer affirms conformity to regulatory requirements, and a medical device without a CE mark cannot be sold or distributed, even free of charge. In addition to specific product-related requirements, the processes by which the device is being manufactured and maintained must comply with the regulations. Both standalone and embedded medical software products are regulated under the same EU regulation as physical devices.

The DevOps paradigm has significantly changed the way how software is being developed today. The technology transformation is supported by modern toolchains that are designed with automation in mind. In addition, public cloud platforms offer flexible computational environments with high availability, automated infrastructure, and reliable software delivery. As the software development industry continues to improve, relying on the DevOps best practices [1], it seems inevitable that DevOps will become the norm regardless of the industry. However, certain DevOps key practices, such as short lead time for changes and high deployment frequency, can be problematic from a medical device compliance perspective [2].

In this work, we aim at improving the medical device software development process by utilizing the automation capabilities of the DevOps paradigm while achieving compliance with the regulations. We focus on standalone medical software and strive to combine the DevOps goals of short process lead time and efficiency with regulatory goals of product safety and clinical effectiveness. To achieve the goal, we systematically address the most relevant medical device regulatory requirements, align them with DevOps automated software delivery concept, and propose a software delivery pipeline compliant with the regulatory requirements. Our research is based on several years of hands-on engineering of standalone software medical devices in the industry, covering in-house development and consulting roles.

The rest of the paper is structured as follows. In Section 2, we provide the background for the paper. In Section 3, we present the basics of DevOps practices and a reference model for continuous software delivery in an unregulated environment. In Section 4, the requirements of the standards IEC 62304 and IEC 82304-1 are aligned. In Section 5, we present our proposed Regulated DevOps (RegOps) pipeline for regulated continuous software delivery. Finally, in Section 6, we discuss the proposed pipeline and draw some conclusions.

2 Background

In the EU region, the manufacturing of medical devices is regulated by Medical Device Regulation (MDR) and In Vitro Diagnostics Regulation (IVDR). According to the legislation, a medical device must be clinically effective for its intended medical purpose, and it must be safe to use. Therefore, medical devices are classified according to their potential risk for a person's health. Determining the correct device classification is essential, as the device class defines applicable conformity assessment procedure and the extent of a third-party conformity assessment body involvement within the process. Thus, the intended purpose and technical properties of a device define how heavily it is being regulated. In addition, also the technical documentation of the device is part of the product. It is not uncommon that regulatory requirements are seen as burdensome activities for the manufacturers [4].

The EU regulatory framework can be interpreted to consist of four layers: 1) Union harmonized legislation, 2) national legislation, 3) harmonized standards, and 4) guidance documents endorsed by the Medical Device Coordination Group (MDCG) [3]. Arguably, the most convenient way to conform with the EU legislation is to utilize harmonized, European versions of the international standards that apply to the device in question as they provide *presumption of conformity* to legislation. At present, there are no software-specific harmonized standards against MDR and IVDR, which creates a certain level of uncertainty as to which are the appropriate standards to apply. However, the EU Commission's recent standardization request [5] is an excellent source of information to get an insight into the expectations of regulatory authorities. For all medical software, the applicable set includes general requirements for health software product safety (IEC 82304-1), software life cycle process (IEC 62304), risk management process (ISO 14971), usability engineering (IEC 62366-1), quality management system requirements (ISO 13485), and security activities in the product life cycle (IEC 81001-5-1). Depending on the intended purpose and technical properties of the software product, also other standards may be relevant.

From our experience, the appliance of medical device standards to software manufacturing can significantly slow down the development process. The time delay between development and release can be measured with the *process lead time* of a software change, which essentially means the latency between initiation and completion of a process [6]. Modern software development paradigms, such as DevOps, could assist in reducing the process lead time. However, a clear, universally accepted definition for DevOps and the related practices/activities does not exist. Despite that, DevOps tries to close the gap between the development (Dev) and the operations (Ops) [7] with a set of practices that developers and operators have agreed upon. From the technological viewpoint, the goal of DevOps is to reduce and join repetitive tasks with process automation in development, integration, and deployment [8]. In practice, process automation is implemented in the Continuous Integration and Continuous Delivery (CI/CD) pipelines. A pipeline consists of an automated and repeatable set of software life cycle processes. The key to efficient software delivery is automation, repeatability, and reliability of the software deployment [9].

In this paper, the concept of DevOps leans towards process automation and pipelines, which can perform automated tasks repeatedly to shorten the process lead time and reduce the risks related to the software delivery with deterministic deployment practices. For this reason, the term DevOps pipeline is used as the primary term for referring to a set of sequential activities/tasks that can be performed repeatedly and reliably.

3 DevOps and Pipelines

In this section, we present an overview of the key characteristics of DevOps that are relevant from the viewpoint of software integrity and delivery process automation, followed by a reference *Continuous software delivery model* [10].

3.1 Building blocks for a DevOps Pipeline

A DevOps pipeline helps teams to build, test and deploy software through a combination of tools/practices. Common tasks performed by a DevOps pipeline are software integration and deployment. Continuous Integration (CI) is the practice of integrating new code frequently, preferably as soon as possible [11]. The pipeline builds the software, runs an automated set of verification tasks against the software, and places the built software artifact into the dedicated software artifact repository [6]. A software artifact is a piece of software, such as a binary file, which can be copied into different computational environments [9]. The artifact repository stores the builds generated by the CI pipeline alongside the metadata of the build [9]. The purpose of the CI is to prevent the code from diverting too much between the developers and keep the code constantly intact, ready for release. CI is enabled by storing the code in a source code repository, which any modern distributed version control system, such as Git, can offer.

Continuous Deployment is a concept for an automated software delivery model. By applying this practice, the software is deployed into a specific computational environment after all automated verification activities are passed, without a human-made approval, but not necessarily available to end-users [9, 10]. To establish the practice, a high level of automation in the software development, testing, and delivery processes is required [6]. The pipeline handles the deployment to different computational environments. Staging environments are utilized for testing and verifying the functionality of the software. Finally, the production environment is the environment for the final end-users of the software product.

3.2 DevOps Pipeline Reference Model

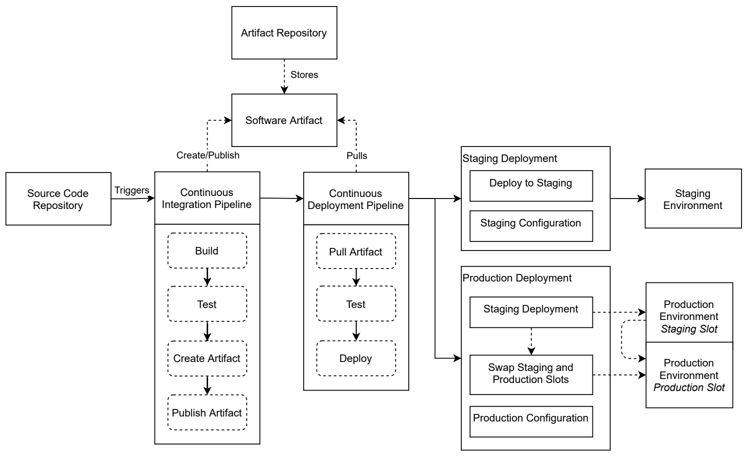

Fig. 1. Continuous software delivery model, adapted from [10].

For this work, we chose a Continuous software delivery model presented by Google [10] as the primary reference for the concept of the DevOps pipeline, as illustrated in Figure 1. We also considered models from Humble and Microsoft [9, 12]. Google's model was selected because, based on our own experience from the industry, it is well-fitting for practical use. In addition, it illustrates the basic pipelines and relationships between different concepts well.

In the reference model, the software verification and delivery are highly automated. A check-in into the source code repository triggers the CI pipeline, which builds the software and runs automated checks and tests against the generated build. Software artifacts are created and published into the artifact repository. If the CI pipeline passes successfully, the build is deployed into the staging and the production environments. The software can be tested and verified in the staging environment manually before being released to the end-users. However, if the CI pipeline fails, the original developer is notified to fix the defects. A failing build will not be deployed.

The reference model was altered to present the sequential order of the CI/CD pipelines and the activities more accurately. Finally, the model was polished with the concept of Blue-Green Deployment [9], which enables more frictionless deployments. In practice, there are two identical copies of the customer environment present, which are swapped on software release. Thus, the rollback can be performed by swapping the slots back to the original position if any problems occur. However, databases can be quite challenging to manage while using this technique; thus, the issue should be recognized in the system's architectural design.

4 Integration of Regulatory Requirements into the DevOps pipeline

We aim to improve the medical device software development process by utilizing the automation capabilities of DevOps pipelines while simultaneously achieving compliance with the most central regulatory requirements. Therefore, we selected the two most important international standards related to medical device software development for this research, namely IEC 62304 [13] and IEC 82304-1 [14].

Table 1. Requirements implemented in the pipeline.

Gate ID	Gate title	62304 reqs.	82304-1 reqs.
G1	Continuous Integration	5.6.3, 5.6.5, 5.6.7, 5.7.4 7.3.3, 8.1.2, 8.1.3, 9.8	n/a
G2	Change Review and Approval	5.3.6, 5.4.4, 5.5.5, 5.6.1 5.7.4, 8.1.2, 8.1.3	n/a
G3	Deployment Pipeline	n/a	n/a
G4	Integration Verification	5.6.2 - 5.6.7, 5.7.4, 5.7.5 7.3.3, 8.1.3, 9.8	n/a
G5	Manufacturer Release Approval	5.6.6, 5.7.4, 5.7.5, 5.8.1 5.8.3, 5.8.4, 5.8.6, 5.8.7 7.3.1, 7.3.3, 8.1.3, 9.8	6.2, 6.3, 7.1, 8.3

Although the selected standards do not cover all regulatory requirements that must be met to place the product on the EU market, in our experience, they represent exactly the part of the requirements that can benefit from technical DevOps practices.

4.1 Requirements of the Selected Standards Aligned

In our research, we aimed to identify the requirements from the standards that can be automated or which can otherwise be implemented in a similar pipeline process as illustrated in Figure 1. In our systematic approach, first, we went through both standards and collected the requirements at the level of numbered clauses. Second, we divided the requirements into three categories:

1. requirements that could be implemented in a pipeline (presented in Table 1),
2. requirements that could be partially implemented in the pipeline (presented in Table 2), and
3. requirements excluded from the pipeline (presented in Table 3).

Finally, we mapped the requirements from categories 1 and 2 to different logical stages, presented as Gates, in our pipeline. Because of the nature of medical device regulatory requirements, certain Gates in our model are not fully automated but instead contain manual, human-made decision steps. It should be noted that the process to divide the requirements into categories and further map them to logical stages was done iteratively.

The IEC 62304 contains three software process rigor levels based on the risk level of the software. The levels are A, B, and C, from the lowest risk level to the highest. Even if rigor levels A and B allow the exclusion of certain clauses of the standard, our pipeline addresses the full spectrum of the requirements, making it suitable for also the software with the highest risk classification.

4.2 Burdensome Requirements Arising from the Standards

From a compliance perspective, the essential aspect of the development process is that all regulatory requirements are considered and implemented appropriately. These requirements create an additional layer of challenge to the usual complications related to software projects. The standards IEC 62304 and IEC 82304-1 contain some burdensome requirements, the automation of which could significantly improve the efficiency of the development process.

As discussed previously, the importance of documentation is crucial in medical device software development. As the software evolves during every development iteration and change, the corresponding technical documentation must be kept up to date, often challenging and laborious if done with manual processes. For instance, IEC 82304-1 requires the manufacturer to have comprehensive accompanying documentation containing information regarding the safety and security of the software product. These documents include, for example, instructions for use and the technical description. The standard contains fairly detailed requirements on the content of accompanying documentation. Furthermore, IEC 62304 further extends the requirements for technical documentation to include details of the documents that must be produced during different development lifecycle activities and tasks. For example, IEC 62304 requires the manufacturer to create software architecture and detailed design documentation for software units.

The requirements related to Software of Unknown Provenance (SOUP) can be a particularly troublesome area for manufacturers as IEC 62304 requires appropriate management of SOUP items according to its comprehensive rules. SOUP refers to a software or part of the software that is not intended for medical use but is incorporated into a medical device. SOUP also includes parts of software that have been developed before the medical device development processes have been available. The manufacturer must identify and list all SOUP components and specify functional, performance, system, and hardware requirements for the identified components. These documents are part of the product's required technical documentation.

Table 2. Requirements partially implemented in the pipeline.

62304 req.	**Qualifying remarks**
5.1.12	The planning activities are performed before the pipeline, but the pipeline can support the requirement by utilizing automated Linter-tools etc.
5.3.1 - 5.3.4	Even if the design activities are performed before the pipeline, the pipeline can support the implementation of the requirement by automating the creation of architecture documentation.
5.4.1	Even if the design activities are performed before the pipeline, the pipeline can support the implementation of the requirement by an automated creation of architecture documentation. In practice, the automated generation of documentation is assisted by using annotations to document the software structure.
5.5.2	Even if process establishing activities are performed before the pipeline, the pipeline is a tool to implement the requirement.
5.5.3	Even if the design and specification activities are performed before the pipeline, the pipeline can support the verification of the software unit implementation.
5.7.1	Even if test establishing activities are performed before the pipeline, the pipeline is a tool to implement the requirement, i.e. performing the tests.
5.7.3	Retesting is performed in pipeline implicitly, whereas risk management activities must be performed before pipeline.
5.8.2	Anomaly management can be automated to a certain degree, and the documentation generation and verification can be done within the pipeline.
5.8.5	Planning tasks, incl. the software development plan and management of the development process are performed before the pipeline. However, infrastructure code can be part of the documentation, and pipeline participates in the creation of the documentation.
5.8.8	Technical practices can be implemented. However, practices such as user access management for the pipeline infrastructure, are managed outside of the pipeline.
6.3.2	See details from 5.8 requirements.
7.1.3	Even if the evaluation activities are performed before the pipeline, the pipeline can support the implementation of the requirement by checking that evaluation exists and offering a convenient tool to update the evaluation if it is missing.
7.4.3	See details from 7.1, 7.2, and 7.3 requirements.
8.2.3	See details from 5.7.3, and 9.7.
82304 req.	**Qualifying remarks**
4.4, 4.7	Even if requirements management is done mainly before pipeline, use requirement may need to be updated as a result of verification and validation activities.
7.2	Contents of documentation is created before the pipeline, but the pipeline can support the requirement by automating the compilation of the documentation.

5 RegOps Pipeline for Medical Software

This section presents our proposed pipeline for the medical device software, the RegOps pipeline, which builds on the reference pipeline illustrated in Figure 1, and the results of Section 4. To ensure compliance against the regulatory requirements, the RegOps pipeline contains both automated and manual activities. The DevOps stages are modeled as Gates, with acceptance criteria that must be met before software release activities can proceed to the next stage. When the software release has passed all Gates, the regulatory requirements implemented within the RegOps pipeline have been fulfilled for that specific version of the software product. The RegOps pipeline is illustrated in Figure 2 and presented in more detail later in this section.

It is worth noting that the RegOps pipeline relies on specific technical infrastructure details. For instance, we assume that the manufacturer manages product-related user requirements, software requirements, risks, anomalies, and change requests in electronic systems that can be integrated with the Version Control System (VCS). The RegOps pipeline itself can be implemented with a modern DevOps tool-set that is extended with customized improvements to support regulatory compliance.

5.1 Gate 1: Continuous Integration

The CI is the first stage of the RegOps pipeline. When new code is checked into the source code repository, the software build is triggered. In addition, automated verification activities and static code analysis are performed, and the product documentation is generated. Finally, the software artifact is published in the software artifact repository. The associated documentation is published into the dedicated documentation storage and made available for review.

In the *Integration testing* step, the covered regulatory requirements involve software unit verification, software integration testing, and the documentation of the results (IEC 62304 clauses 5.6.3, 5.6.5, 5.6.7, 5.7.1), which the CI stage can perform depending on the test automation coverage. However, the verification activities that are not covered by the test automation must be tested later manually, increasing the process lead time. In addition, IEC 62304 requires identifying and avoiding common software defects (clause 5.1.12), and the implementation of this requirement can be partly automated by performing a *Static code analysis* against the source code. The coding conventions, errors not detected by compilers, possible control flow defects, and usage of variables that have not been assigned are audited during the analysis [15].

The major source of concern related to SOUP management, as discussed previously, is significantly reduced by automatically tracking SOUP components, which can also be done using static code analysis [17]. In practice, the pipeline performs *SOUP analysis* by identifying the SOUP items from the software (IEC 62304 clause 8.1.2). However, the IEC 62304 requirements for the manufacturer to specify functional, performance, and hardware specifications for SOUP components must still be implemented appropriately (clauses 5.3.3, 5.3.4). The re-

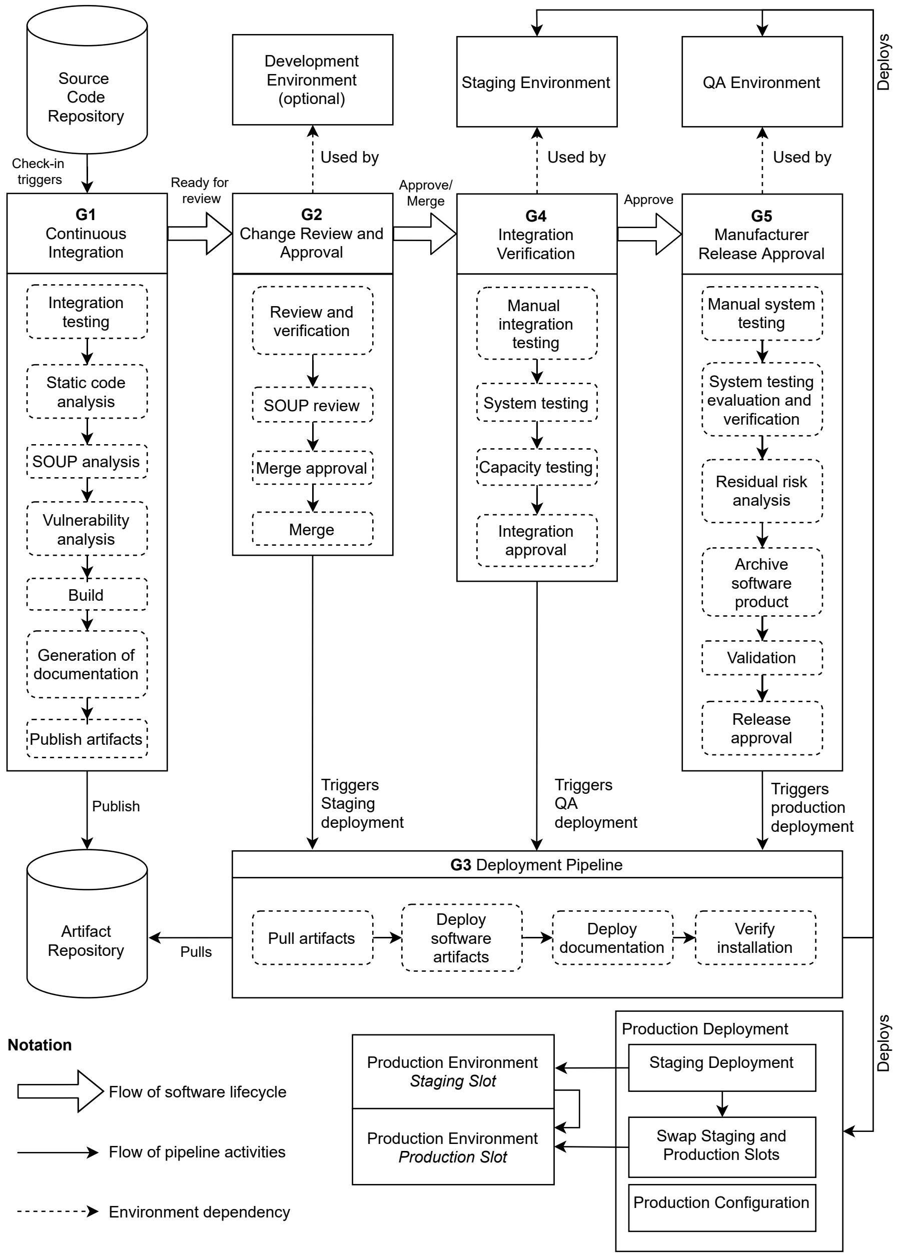

Fig. 2. Our proposed RegOps Pipeline with the regulatory activities emerging from the requirements of the standards IEC 62304 and IEC 82304-1 applied.

quired specifications are documented either before the commit or in Gate 2, at the latest. Finally, various vulnerability analyses could be performed to find any vulnerability risks arising from the SOUP items. As an example, OWASP dependency-check tool [16] is an efficient security utility to find vulnerabilities from third-party components.

In the *Generation of documentation* step, the software documentation is automatically compiled, to the extent possible. For example, the required architecture documentation (IEC 62304 clauses 5.3.1, 5.3.2) can be generated by using an augmented C4 software architecture model [17] by appropriately annotating the source code packages. The generated decomposition diagram represents the actual state of the software structure. In general, the source data for documentation content can be pulled from different data sources, and the generated documentation is stored in the VCS with the software source code. Automatic document creation also enables implementing other IEC 62304 requirements, such as refining the system into software units (clause 5.4.1) and documenting traceability (clause 7.3.3). In addition, the step contributes to implementing requirements related to software system testing verification (clause 5.7.4), system configuration documentation (clause 8.1.3), and test documentation (clause 9.8). Finally, IEC 82304-1 requirements for accompanying documents (clause 7.2), as discussed earlier, can be implemented.

To summarise, the CI stage does most of the heavy lifting in the RegOps pipeline. It performs the automated part of the software integration verification and prepares the software artifacts and the documentation for the following stages of the pipeline. Aside from all the activities, CI should be a repeatable and quick process to give feedback for the developers [18].

5.2 Gate 2: Change review and approval

Not all regulatory requirements can be implemented by automation, and certain specific required tasks need manual verification. Such verification activities are often characterized by the fact that they are related to the outcomes of the previous steps. The second stage of the RegOps pipeline, the Change Review and Approval stage, is the first manual phase of the pipeline. Its purpose is to ensure systematic analysis and endorsement of the change made, both in source code and documentation. The stage builds on the pull-based development model [19]. A pull request is the developer's way of announcing that their work has been finished and is ready for further actions [20]. In practice, after the code is committed and pushed into the source code repository, the pull request is created automatically by the pipeline.

The first step in the stage is *Review and verification.* IEC 62304 requires the manufacturer to verify the software units (clause 5.5.5) against the software unit acceptance criteria (clause 5.5.3) defined in the software unit verification process (clause 5.5.2). The verification can be done partially by automated testing in Gate 1, but the requirement can only be fully met with a code review by another developer entitled to approve the change. Other required review and verification activities include detailed design verification (clause 5.4.4) and software architecture verification (clause 5.3.6). Even if the planning of these activities is done before the RegOps pipeline initiates, the outcomes can only be verified after the commit. The step also contributes to implementing requirements related to the evaluation of verification strategies (clause 5.7.4) and system configuration documentation (clause 8.1.3). In addition, IEC 82304 requires updating the health software use and system requirements following verification, as appropriate (clauses 4.4, 4.7). Therefore, in the verification review, if any contradictions are found from the requirements, they may be updated to reflect reality.

The SOUP items were identified and analyzed in the previous stage, but there needs to be a formal *SOUP review*. Only a human can approve or reject the new SOUP components to be included in the medical software. Also, the additional documentation contents must be verified (IEC 62304 clauses 5.3.3, 5.3.4). Additionally, the published SOUP anomaly lists must be evaluated (clause 7.1.3).

Finally, in the *Merge approval* step, when the verification tasks are completed, approved, and recorded, the change-set - that exists in the form of a pull request - can be approved by a competent person. After the approval, the software integration is performed automatically [21]. However, if any merge conflicts appear, changes are withdrawn, and the software developer is notified to fix the problems. In a successful *Merge*, the new code and documentation are integrated into the mainline of the source code repository. As a by-product, the regulatory requirement to integrate the software units (clause 5.6.1) is fulfilled.

5.3 Gate 3: Deployment Pipeline

The concept of Continuous Deployment, when considered as an automatic software release for the end-user and as illustrated in Figure 1, is problematic in medical device software development [2,4]. For instance, regulatory requirements related to product validation can be seen as an obstacle for deploying the software directly to use, without human-made actions. As a result, the release of the software to the end-users must be gated by a human decision to ensure regulatory compliance. Furthermore, the final decision to release is human-made by the manufacturer's authorized employee. Therefore, the RegOps pipeline contains a Deployment Pipeline stage, which deploys the software into a specific computational environment after approval, and only with appropriate approvals, the software can be released to the end-users. The same stage is utilized for all deployments; only the destination computational environment differs. As the software change advances through the pipeline gates, it is deployed into inspection of other developers, QA experts, and finally, to be used in real life.

Technically, the Deployment Pipeline pulls the software artifact generated by the CI stage. The software artifact is then deployed into the destination computational environment. The documentation accompanied by the software is also published in a dedicated location for further review. Finally, the software deployment is verified automatically by performing a set of smoke tests, which verify that the software is up and running. If the software does not start or the smoke tests fail, the developer is notified immediately to fix the problem.

Our research did not identify any specific requirements from the addressed standards to be implemented in the Deployment Pipeline stage. However, as the deployment is a crucial part of the system, the software change cannot be approved unless the deployment is performed successfully. Also, the deployment must be gated by a human decision. For instance, Gate 4 and Gate 5 contain specific approval activities, which essentially trigger the software deployment.

5.4 Gate 4: Integration Verification

As a result of the actions performed in previous stages, the software is deployed into the Staging Environment, which is as accurate a copy as possible of the final use environment [22]. Hence, the software product can be reviewed as a whole. In addition, the software artifacts deployed to the Staging Environment are already reviewed by another person and automatically tested for possible defects. However, there may be a need to perform *Manual integration testing*: not all test cases can be automated, and test automation cannot be used for exploratory testing. Any additional tests can be performed in the Staging Environment to test and verify that the software integration has been performed successfully (IEC 62304 clause 5.6.2). The software integration and regression testing (clauses 5.6.3 - 5.6.7) are finished at this stage.

The software system's functionality is verified through *System testing*, which ensures that the software system meets its intended requirements and performs as designed. The system testing must be carried out in a computational environment that closely corresponds to the actual use environment to ensure reliable test results. In Gate 4, the part of the system testing that is automated is carried out (clauses 5.7.1, 5.7.3, 5.7.5, 7.3.3, 8.2.3, 9.8), and, again, the stage contributes to implementing requirements related to software system testing verification (clause 5.7.4) and system configuration documentation (clause 8.1.3). A high degree of automation coverage enables efficiency by reducing the burden of manual work during the later stages. Furthermore, as automated system testing can take a considerable amount of time, it is only carried out in Gate 4, allowing fast feedback for the developers from earlier stages in the pipeline [23]. In a scenario where any anomalies are found during the system testing, the pipeline will not proceed. The identified anomaly is escalated to the problem resolution process (clause 5.7.2). Finally, the development team is notified with an automatic problem report (clause 9.1).

The performance of the system is tested by running a set of relevant tests in the *Capacity testing* step. The capacity testing provides a way for the manufacturer to analyze the behavior of the system under stress. For example, any change in the software could introduce performance issues, which can be detected early by running performance tests against the system.

The last step in Gate 4 is *Integration approval* after all other integration verification activities are completed. Technically, the approval triggers the Deployment Pipeline, which then deploys the software into the QA Environment. The QA Environment is the final environment for testing and verification before the software can be released.

5.5 Gate 5: Manufacturer Release Approval

In Gate 5, the software is system tested, the system testing is evaluated and verified, and the software product is validated before the final release. The activities performed in this stage are primarily manual or require human inspection.

The test cases that could not be automated are performed in the QA environment (clauses 5.6.6, 5.7.5, 7.3.3, 9.8) within the activity of *Manual system testing.* Depending on the test automation coverage, this stage may require significant amounts of resources. However, even with comprehensive test automation coverage, exploratory testing is recommended [24].

After the system testing has been completed, the system testing activities must be evaluated and verified within the formal *System testing evaluation and verification* step. In practice, the system test results are evaluated and verified as stated in IEC 62304 (clauses 5.7.4, 5.8.1). Essentially, all relevant test cases are verified to have been performed properly. Furthermore, according to IEC 62304, any anomalies found from the product must be documented and evaluated (clause 5.8.2, 5.8.3). In addition, the risk control measures are to be verified (clause 7.3.1). Finally, the residual risk level of the medical device product must be reduced to or remain at an acceptable level before the release to the end users can happen. These requirements are implemented in the *Residual risk analysis* step.

Before the final release, the manufacturer must ensure that all activities mentioned in the software development plan are completed (clause 5.8.6). The software and documentation artifacts created by the CI are labeled with a release version tag (clause 5.8.4). However, IEC 82304-1 extends this requirement to require a Unique Device Identifier (UDI) (clause 7.2). The software artifacts are transferred into a permanent archive (IEC 62304 clause 5.8.7). The archived software artifacts and documentation are used to install the product into the computational environment where it will be used. From a technical perspective, the medical device software product is ready to be released at this point. First, however, the manufacturer must perform the *Validation* according to the validation plan (IEC 82304-1 clauses 6.2, 6.3, 8.3). Essentially, the manufacturer must obtain reliable evidence of the software to fulfill its intended purpose.

Finally, when the software product is technically intact and verified to conform to the regulatory requirements, the software can be released and deployed into the customer environment by formal *Release approval.* In the pipeline, we utilized the Blue-green deployment, which practically means deployment into the staging slot of the environment, as discussed earlier. This practice allows customer organizations to familiarise themselves with the product as it is common that they have their validation processes. Then, when it is time to release the software to the end-users, the staging slot can be swapped with the production slot, making the software available for real-world use.

It should be highlighted that the pipeline implements some of the regulatory requirements implicitly, such as documenting how the release was made, the repeatability of the release, and re-releasing the modified system (clauses 5.8.5, 5.8.8, 6.3.2). These are the core principles of the pipeline.

6 Discussion and Conclusions

In this paper, we have collected the most central regulatory requirements related to medical software, that is, requirements from the standards IEC 62304 and IEC 82304-1, and integrated the aligned requirements into our proposed software delivery pipeline. The resulting RegOps pipeline aims to reduce the lead time of the software delivery while at the same time maintaining compliance with regulatory requirements. We identified 110 requirements from the standards, of which 26 are fully implemented, and 20 are partially implemented within the pipeline. The remaining 64 requirements, shown in Table 3, were scoped out.

Table 3. Requirements scoped out of the pipeline.

62304 req.	**Explanation**
4.1	Applies to the whole organization and all its functions.
4.2	Applies to the entire product development process.
4.3	The software process rigor level is decided outside of the pipeline. However, it affects how many requirements are applicable for the product.
4.4	The model is intended to support software development done in compliance with the standard.
5.1.1 - 5.1.9 5.1.11, 6.1, 8.1.1	Planning activities are performed before the verification activities.
5.1.10	Applies to the entire product development process. However, the pipeline needs to be addressed in the supporting items management.
5.2.1 - 5.2.6, 5.4, 5.3.5, 5.4.2 5.4.3, 5.5.4, 7.1.1 7.1.2, 7.1.4, 7.2.1 7.4.1, 7.4.2, 8.2.1	The software requirements management and technical design activities are performed before the verification activities.
5.5.1, 6.3.1	The implementation activities are performed before verification activities.
5.6.8, 5.7.2	However, anomalies detected in the pipeline can be automatically forwarded to the software problem resolution process.
6.2.1 - 6.2.6	Problem and modification analysis activities are performed before pipeline.
7.2.2, 8.2.2	The software requirements management, technical design, and implementations activities are performed before verification activities.
8.2.4	Change requests, change request approvals, and problem reports are managed outside of the pipeline.
8.3	Configuration item history is stored in VCS.
9.1 - 9.7	Software problem resolution is managed outside of the pipeline.
82304 req.	**Explanation**
4.1	Applies to general product documentation.
4.2, 4.3, 4.5, 4.6	The requirements management and technical design activities are performed before the pipeline.
6.1	Planning activities are performed before the pipeline.
8.4, 8.5	Post-market activities are performed after the pipeline.

Ideally, the use of the RegOps pipeline could enable early customer feedback from healthcare practitioners by allowing them to test the software in the staging slot of their computational environment before the software is released into medical use. In addition to collecting feedback, this allocated environment could be used to perform customer-specific acceptance testing and validation activities. First, however, it must be ensured that the unreleased software is not used for patient treatment in any circumstances.

The paper's primary contribution for medical device software industry professionals is to provide a conventional and pragmatic approach to deliver software for real-world use. As the medical regulations introduce rather unique requirements in the software industry, such as a demand for extensive and traceable documentation, the automation capabilities of the pipeline are precious in the automatic generation of documentation. Furthermore, when the whole delivery process is implemented within the pipeline, it is implicitly deterministic and in control by nature in the form of version-controlled infrastructure code - a feature that regulatory professionals appreciate. For researchers, our consolidated regulatory requirements can act as a baseline for future extensions in the use of DevOps practices in the medical domain. Finally, for those interested in only DevOps, our pipeline could offer a new perspective and ideas for their work.

As a limitation in our proposed approach, we acknowledge that the RegOps pipeline does not alone fulfill every applicable regulatory requirement associated with a medical software product. Therefore, it is only a part of the overall solution for regulatory compliance. As discussed previously, in addition to these standards, other regulatory requirements must also be taken into account when designing and implementing the medical device manufacturing process.

In the future, we intend to perform a case study to validate our proposed approach's applicability in real-world use and to develop the concept further. Then, the pipeline could be expanded to apply to the entire manufacturing process while retaining the mindset for process automation. Finally, the applicability of the proposed pipeline could be explored in the field of embedded medical devices, in which case additional regulatory requirements related to electrical equipment need to be taken into account.

Acknowledgements The authors would like to thank Business Finland and the members of AHMED (Agile and Holistic MEdical software Development) consortium for supporting this work.

References

1. Forsgren, N., Smith, D., Humble, J., Frazelle, J.: 2019 Accelerate State of DevOps Report. DevOps Research and Assessment & Google Cloud (2019)
2. Granlund, T., Mikkonen, T., Stirbu, V.: On Medical Device Software CE Compliance and Conformity Assessment. In: IEEE International Conference on Software Architecture Companion (ICSA-C), pp. 185-191. IEEE (2020)
3. Granlund, T., Stirbu, V., Mikkonen, T.: Towards Regulatory-Compliant MLOps: Oravizio's Journey from a Machine Learning Experiment to a Deployed Certified Medical Product. SN COMPUT. SCI. 2, 342 (2021)
4. Laukkarinen, T., Kuusinen, K., Mikkonen, T.: DevOps in Regulated Software Development: Case Medical Devices. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE-NIER), pp. 15-18. IEEE (2017)
5. A standardisation request regarding medical devices to support Regulation (EU) 2017/745 and (EU) 2017/746, https://ec.europa.eu/growth/tools-databases/mandates/index.cfm?fuseaction=search.detail&id=599. Last accessed 2 Jul 2021

6. Kim, G., Humble, J., Debois, P., Willis, J.: The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. IT Revolution Press, Portland (2016)
7. Wettinger, J., Breitenbücher, U., Leymann, F.: DevOpSlang – Bridging the Gap between Development and Operations. In: Service-Oriented and Cloud Computing, pp. 108–122, Springer Berlin Heidelberg, Berlin (2014)
8. Laukkarinen, T., Kuusinen, K., Mikkonen, T.: Regulated software meets DevOps. Information and Software Technology, vol. 97, pp. 176—178. (2018)
9. Humble, J., Farley, D.: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley (2010)
10. Google Cloud Architecture Center, https://cloud.google.com/architecture/addressing-continuous-delivery-challenges-in-a-kubernetes-world. Last accessed 21 May 2021
11. Fowler, M.: Continuous Integration https://www.martinfowler.com/articles/continuousIntegration.html. Last accessed 1 June 2021
12. Microsoft Documentation: DevTest and DevOps for microservice solutions, https://docs.microsoft.com/en-us/azure/architecture/solution-ideas/articles/dev-test-microservice. Last accessed 8 Jul 2021
13. IEC/EN 62304:2006/A1:2015. Medical device software - Software life-cycle processes, 2015.
14. IEC 82304-1:2016. Health software — Part 1: General requirements for product safety, 2016.
15. Wichmann, B., Canning, A., Marsh D., Clutterbock D., Winsborrow L., Ward N.: Industrial perspective on static analysis. Software Engineering Journal, vol.10 (2), 69–75 (1995)
16. OWASP Dependency-Check Project, https://owasp.org/www-project-dependency-check/. Last accessed 12 Jul 2021
17. Stirbu, V., Mikkonen, T.: CompliancePal: A Tool for Supporting Practical Agile and Regulatory-Compliant Development of Medical Software. In: 2020 IEEE International Conference on Software Architecture Companion (ICSA-C), pp. 151-158. IEEE (2020)
18. Duvall, P., Glover, A, Matyas, S.: Continuous Integration : Improving Software Quality and Reducing Risk. Addison Wesley, 2007.
19. Sadowski, C., Söderberg, E., Church, L., Sipko, M., Bacchelli, A.: Modern code review: a case study at Google. In: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, pp. 181-190. Association for Computing Machinery, New York (2018)
20. Fowler, M.: Pull Request, https://martinfowler.com/bliki/PullRequest.html. Last accessed 1 June 2021
21. Ståhl, D., Bosch, J.: Automated Software Integration Flows in Industry: A Multiple-Case Study. In: Companion Proceedings of the 36th International Conference on software engineering, pp. 54-63. ACM (2014)
22. Morales, J., Yasar, H., Volkman, A.: Implementing DevOps practices in highly regulated environments. In: Proceedings of the 19th International Conference on Agile Software Development: Companion, Article 4, pp. 1-9. ACM (2018)
23. Laukkanen, E., Mäntylä, M.: Build Waiting Time in Continuous Integration – An Initial Interdisciplinary Literature Review. IEEE/ACM 2nd International Workshop on Rapid Continuous Software Engineering (2015)
24. Shah, S., Cigdem, G., Sattar, A, Petersen, K.: Towards a Hybrid Testing Process Unifying Exploratory Testing and Scripted Testing. Journal of Software: Evolution and Process, vol.26 (2), pp. 220-–250. (2014)