Tampere University

Henrik Toivakka

Integration of EU medical device regulatory requirements into a CI/CD pipeline

Faculty of Information Technology and Communication Sciences (ITC)
Master's thesis
September 2021

Abstract

Henrik Toivakka: Integration of EU medical device regulatory requirements into a CI/CD pipeline
Master's thesis
Tampere University
Master's Degree Programme in Computer Sciences
September 2021

Manufacture of medical device software is strictly controlled by the law in European Union region. The medical device regulations could be seen as cumbersome and difficult to incorporate into the manufacturing process of a software. The regulations introduce a rigid product life cycle process to medical device software products, which could be seen as counter-intuitive by the software industry. Meanwhile, the software industry has found DevOps culture and practices, such as process automation and delivery pipelines, which are utilized to improve the competitiveness and reliability of the software business.

This thesis tries to find a common ground between the European Union medical device regulatory requirements and modern software development practices. The European Union medical device regulatory framework is introduced, followed by a systematic document analysis of the standards IEC 62304 and IEC 82304-1 to identify applicable requirements for a CI/CD pipeline. As a result, a reference model for regulated CI/CD pipeline is introduced.

For practitioners, tools and activities were found, that could help to overcome some of the burdensome activities, required by the medical device regulations. Additionally, the benefits of integrating CI/CD pipeline into the medical device manufacturing process is examined, such as the possibility for early customer feedback.

For researchers, a document analysis based on the standards IEC 62304 and IEC 82304-1 is provided. Moreover, the proposed regulated CI/CD pipeline model can be expanded to cover more regulatory requirements. Finally, research topics related to medical device software manufacturing are discussed.

Keywords: regulatory compliance, medical device software, medical device regulation, in vitro diagnostic regulation, CI/CD pipeline, DevOps, IEC 62304, IEC 82304-1.

Contents

1 Introduction

Medical device is a device that is used for medical purposes, introducing benefits and potential risks to a person's health. The patient's health is a great concern for the regulators, resulting in heavy regulations on the healthcare sector. Traditionally, a medical device is seen as a physical device or equipment. As the technology has evolved, and software has become mainstream, even a medical device may contain software or be a standalone software, leading to the identification of Medical Device Software (MDSW) separately in the EU regulation. The same rules apply to the manufacturers of medical device software, as the manufacturers of traditional medical device equipment. As a result, if the intended use of the software is for medical purposes, the manufacturing must be done under the European Union Regulatory Framework. If the software fulfills the criteria of a medical device, but does not conform to the regulations, it may not be used or sold for medical purposes, even free of a charge.

1.1 Regulations and medical device standards

For a long period, EU legislation consisted of the following directives; Active Implantable Medical Device Directive (AIMDD) [European Commission, 1990], Medical Device Directive (MDD) [European Commission, 1993] and In Vitro Diagnostic Directive (IVDD) [European Commission, 1998]. However, the aforementioned directives are repealed and replaced with Medical Device Regulation (MDR) [European Commission, 2017] and In Vitro Diagnostic Regulation (IVDR) [European Commission, 2017]. After the MDR and the IVDR are fully applicable, the MDD and the IVDD are repealed. The regulations introduce new requirements for medical device software manufacturers, which the medical device manufacturers must implement into their manufacturing processes. MDR came fully into effect 26th of May in 2017 and had a transition phase until 26th of May in 2020. However, the transition phase was prolonged until 26th of May in 2021, because of the pandemic caused by COVID-19 [Fimea, 2020]. The In Vitro Diagnostics Regulation (IVDR) [European Commission, 2017] came into effect 26th of May in 2017, and has a transition phase until 26th of May in 2022. After the transition phase, medical device manufacturers must conform to the regulations, to continue selling medical device products in the EU region. The manufacturer of medical devices in the European Union (EU) region must follow different directives, regulations, national legislation and standards regarding medical devices, to ensure that the device is safe to use for medical purposes. The regulations and standards introduce regulatory activities for

manufacturing a medical device.

For software manufacturers, it could be seen difficult, or even counter-intuitive, to integrate the regulatory requirements into the modern software development practices. The regulatory requirements introduce rigid and burdensome activities, when compared to the unregulated software development. In this thesis, it is assumed that the software already qualifies as a medical device, and the EU regulation is applied to the manufacturing process.

1.2 DevOps and pipelines

DevOps is an operational model for manufacturing digital services, which tries to build a bridge between software development and IT operations. DevOps-influenced software industry has adopted tools and processes to supercharge the development, delivery and maintenance processes of software. Instead of building large monolithic systems, the current industry mainstream is focusing on modular software built on cloud architecture. Additionally, the software is developed rapidly with agile software development models such as Extreme Programming, Kanban or Scrum [Stellman et al., 2014]. Recent trend of DevOps has brought the culture of automation into the processes, which has enabled software verification, delivery and even deployment into the production automatically [Laster, 2020]. The automated verification and deployment activities in DevOps are performed in pipelines, which can perform any predefined tasks repeatedly. In this thesis, the definition of DevOps leans towards automation and pipelines, although it is not the only definitive characteristic of DevOps.

1.3 Research objectives and methods

The purpose of the research is to find ways for manufacture of medical device software to gain more agility and competitiveness, while maintaining compliance with the medical device regulations.

The research questions are following:

1. Is it possible to integrate CI/CD pipeline into the medical device software development process?

2. Is it possible to utilize Continuous Deployment in medical device software delivery process?

3. What advantages could a CI/CD pipeline introduce to a medical device manufacturer?

In this thesis, a systematic document analysis of IEC 62304 [IEC, 2015] and IEC 82304-1 [IEC, 2007] is performed, and the applicable regulatory requirements

to a CI/CD pipeline are identified. IEC 62304 and IEC 82304-1 were chosen as the main standards for this study, as they introduce requirements for software life cycle and product safety, and impact a CI/CD pipeline the most. The regulatory requirements are integrated into a CI/CD pipeline model, and as a result, a regulated CI/CD pipeline model is established. The model is further analysed and reviewed, and the possible follow-up studies, based on the model are discussed.

The study does not include every possible regulatory requirements, and activities, such as interaction with Notified Bodies (NB), risk management, configuration management, clinical evaluation, performance evaluation, product validation, usability validation and details on Quality Management Systems, are excluded out of the scope. The activities are mentioned while discussing the main topic, but not included in the pipeline model. Also, there could be other applicable standards for a medical device software product, depending on the intended use of the product, which are not mentioned in this thesis.

The rest of this thesis is organized as follows. In Chapter 2, the medical device regulations and related international standards are introduced. In Chapter 3, the key characteristics of DevOps and CI/CD pipelines are introduced. In Chapter 4, the requirements from medical device standards are identified and collected for further analysis. In the Chapter 5, the identified requirements from Chapter 4 are integrated into a CI/CD pipeline. Finally, in Chapter 6 the conclusions and further research topics of this thesis are discussed.

2 Regulation of medical devices in EU region

In this chapter, the EU medical device legislation, and the related medical device standards are introduced. The European Union Regulatory Framework for medical devices may be interpreted to consist of four layers, illustrated in Figure 1. The layers are EU legislation, national legislation, guidance documents and international standards [Granlund et al., 2021]. In reality, the EU legislation consists of directives and regulations, and the difference between regulation and directive is significant. Directives are optional, and must be adopted into the national legislation. According to European Commission [European Commission, 2021], an EU regulation is a directly applicable set of legal acts, that EU member countries must conform to, without a national interpretation in their legislation. Furthermore, compliance with the regulations is mandatory. Essentially, the regulatory framework sets the baseline rules for every member country, but leaves room for national legislation.

However, MDR and IVDR introduce even more requirements than MDD and IVDD, which already make medical software development slightly different from non-medical software development. Additionally, some of the requirements from MDR and IVDR are completely new for the software industry. The major changes are following; software manufacturers need to perform global impact assessment, and rationalize the product portfolios. Most legacy devices are addressed with new rules, and existing products need to be re-certified with a CE mark. Devices, that were previously not considered medical devices, could now fulfill the criteria of a medical device, and the existing medical devices could be re-classified into another product class of the MDR. Furthermore, in vitro devices are classified into four different product classes. Finally, the requirements such as clinical evaluation and clinical investigation for MDR, performance evaluation for IVDR, and assessment with a Notified Body (NB) still apply.

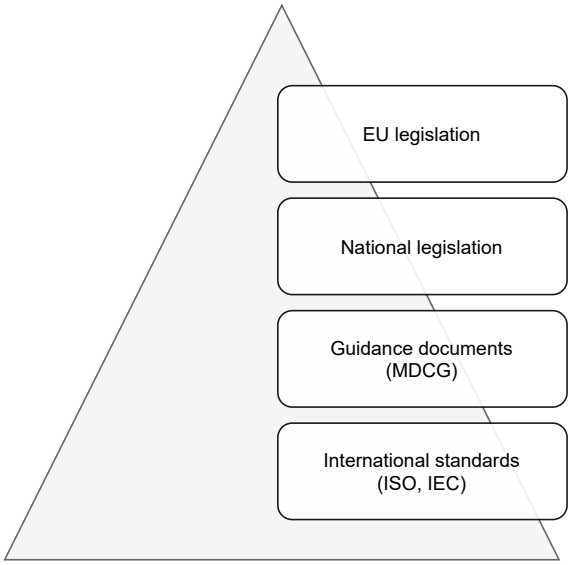

Figure 1 *Different layers of EU regulatory framework [Granlund et al., 2021].*

The EU regulations, as they are, are difficult to bring into the practice, leading to implementation of guidance documents, which clarify the intentions behind the regulatory aspect regarding general safety, performance and standardisation [European Commission, 2017]. Currently, the guidance documents are released by the Medical Device Coordination Group (MDCG), and help with effective and harmonised implementation of the regulations. However, the most practical and convenient way to understand the regulatory aspect, is to follow the applicable set of standards, established by the International Electrotechnical Commission (IEC) and International Organization for Standardization (ISO), and to use a development process which implements the standards IEC 62304 [IEC, 2015], ISO 14971 [ISO, 2007], ISO 13485 [ISO, 2018], IEC 62366-1 [IEC, 2015] and IEC 82304-1 [IEC, 2016].

To meet all the safety and performance requirements of the EU Regulatory Framework, the medical device must implement the regulatory requirements correctly and the manufacturer must implement a Quality Management System (QMS) [Pitkänen et al., 2020].

2.1 Medical Device Regulation

The European Medical Device Regulation (MDR) [European Commission, 2017] aims to harmonize the EU region's legislation, and to remove the national interpretation from the law, when manufacturing any kind of medical device. Moreover, MDR covers the clinical investigation and sale of medical devices for human use. The objective of MDR is to improve the quality, safety and reliability of medical devices, applying to every member country of the European Union.

A medical device implies any instrument, apparatus, appliance, software, implant, reagent, material or other article intended by the manufacturer to be used, alone or in combination for medical purposes [European Commission, 2017]. A medical device software is software that has intended use, either alone or in combination for medical or in vitro purposes [European Commission, 2017].

Classification for medical devices is introduced by MDR, reflecting the potential harm, that could be caused by a medical device, in a scenario where the risk is realized. The product classification is determined by the intended use of the medical device, rather than the composition of the medical device. Moreover, the same product classification is present, regardless, if the device is a software, a physical device, a chemistry product or something else [Ståhlberg, 2015]. For instance, a blood sugar monitoring software and a physical item, such as a syringe, are classified by the same rules. The same product classification rules apply for all medical devices under MDR, and even if the software only controls or influences a medical device somehow, it falls into the same classification as the medical device in the question.

Class I is the lowest risk-level class. All other software, that is not classified as II or III, is classified into class I. Additionally, medical devices classified into Class I, have lesser requirements for the Notified Body (NB) assessment [European Commission, 2017]. Class II is divided into two sub-classes, IIa and IIb, with IIb being the higher risk-level class. Software intended to provide information for diagnosis, therapeutic purposes, or monitoring physiological processes, is classified as a class IIa device. However, if the software could cause harm or danger to a person's health, the device is classified as a class IIb device instead. The distinction between the classes of IIb and III is following; devices which could cause serious, but not irreversible injury to a person's health are classified into class IIb, instead of Class III. Class III is the highest risk-level class for medical devices. A device that could cause a death, or irreversible injury to a person's health is classified into Class III.

To summarise, the MDR introduces regulatory requirements for manufacturing medical devices in the European Union. Moreover, compliance with the regulatory requirements is necessary. Otherwise, the manufactured medical devices can not be sold or distributed in the EU region, even free of a charge.

2.2 In Vitro Diagnostics Regulation

The European In Vitro Diagnostics Regulation (IVDR) [European Commission, 1998] is the medical device regulation for in vitro diagnostic devices, repealing the IVDD [European Commission, 1998], which was not detailed enough for high-risk devices, leading to problems with interpretation and practical application of the directive. The objective of IVDR is to improve quality, safety and reliability of in vitro devices (IVD).

In vitro medical device is any kind of medical device, which is a reagent, reagent product, calibration, control material, kit, instrument, apparatus, piece of equipment, software or a system intended to be used in vitro for examination of specimens, including blood and tissue, derived from human body [European Commission, 2017]. The device may be used alone, or in combination with other devices. IVDR covers monitoring devices and diagnostics, that are specifically designed or used for monitoring human functions. Regulation applies to every device, chemical product, or a software, if the product qualifies for the regulation by the intended purpose of the product. However, because the healthcare software industry is rapidly evolving, only a non-exhaustive list of examples can be made. Systems, that are included by the regulation, support the process from the patient sample to the test result in healthcare, and contain either pre-analytical, analytical or post-analytical functionality, related sample processing. However, the guidance explicitly states that some systems are not medical devices by themselves, but they may be used with additional modules, which can be considered medical devices [Medical Device Coordination Group, 2019].

The different classifications of IVDR are following; Class A is the lowest risk-level class, which contains general purpose equipment and instruments intended to be used in vitro diagnostics. Class B contains devices for self-testing of pregnancy, fertility and cholesterol level. Devices, that do not fit other classifications are classified as class B. Class C contains devices for self-testing glucose, erythrocytes, leukocytes or bacteria in urine. Class D contains devices that detect human organic materials, such as blood, tissues, and agents, that cause life-threatening diseases [European Commission, 2017].

To summarise, IVDR introduces regulatory requirements for manufacturing in vitro devices in EU region, and exactly like in MDR, the manufactured in vitro devices can not be sold or distributed in the EU region, even for free, unless the regulatory requirements are conformed.

2.3 Product safety and clinical effectiveness

Before the medical device software product can be placed on the market, the device must be evaluated and validated properly [Pitkänen et al., 2020]. The evaluation is done in the planning phase of the development life cycle, and should be demonstrated and documented before the product can be released. The validation activities are performed after the development, and before the product can enter the market.

MDR requires a clinical evaluation, which is a systematic and planned process to continuously generate, analyse, collect and assess the clinical data and verify the safety and performance, including clinical benefits, when the device is used as intended [Pitkänen et al., 2020]. It is mandatory for all medical devices under MDR,

regardless of the product class.

On the other hand, IVDR requires a performance evaluation. The conclusion of performance evaluation demonstrates scientific evidence, that the intended clinical benefits are achieved, and that the device is safe to use. The evidence is used to prove, that general safety and performance requirements of the MDR are fulfilled under normal use [European Commission, 2017].

The purpose behind clinical evaluation and performance evaluation is to provide evidence, that the medical device product performs as intended, and it can be safely used for patient treatment.

2.4 Product placement on the market

To be able to sell medical device products in the EU market, the manufacturer must prove that the product complies with the regulatory requirements. The CE mark is European Union's mandatory marking for regulating goods sold in the EU countries [European Commission, 2021]. The CE marking is a proof from the manufacturer, that the medical device product meets the mandatory regulatory requirements. Furthermore, the declaration of conformity is obtained from a Notified Body (NB), which is an organization responsible for verifying the conformity of the product. After the conformity has been verified, the manufacturer can label the product with the CE mark, which is mandatory when selling medical devices in the EU region [European Commission, 2017].

Besides of the CE marking, there are other activities for the medical device manufacturer, before the product can be placed on the market. All applicable regulatory requirements must have been identified, and applied to the device. Moreover, all required technical documentation, and the medical device technical files are verified to exist and properly implemented [Granlund et al., 2020]. Additionally, the manufacturer must have a Quality Management System (QMS) established [Pitkänen et al., 2020]. The manufacturer must ensure, that intended use of the device is defined, and prove that the device fulfills the definition. Finally, the device should be identifiable with Unique Device Identifier (UDI).

The product classification must be done correctly, and the correct conformity assessment procedure must be selected for the device. The NB should be involved in the assessment procedure, depending on the applicable regulatory requirements. Before the release, the device should be registered for the national authority, or the EUDAMED database [European Commission, 2021] in case of MDR, and a declaration of conformity should be signed [Granlund et al., 2020].

2.5 Overview of medical device standards

A medical device software manufacturer must implement the regulatory framework requirements in the software development process correctly, to conform with the regulations. The standards guide the manufacturer to use the best practises in the industry, but do not require any specific product development model. The relationship of the medical device standards and implementation of the medical device software, is illustrated in Figure 2. However, as a reminder, depending on the nature of the medical device software, there could be other standards that apply.

Medical devices manufacturers, that follow harmonised standards, are presumed to conform to the regulation [European Commission, 2016]. Essentially, harmonisation means that the European Union has incorporated the standard into the European Union legislation. By following the harmonised standards, it is easier to implement the mandatory activities to achieve regulatory compliance. However, the legislation is evolving over time. Therefore, it is suggested to apply the current state of the art of the medical device standards, meaning the latest development of the standardisation industry, regarding the development life cycle, risk management, including information security, risk control measures, verification and validation [European Commission, 2017]. The application of the medical device standards into software manufacturing process is optional, but highly recommended.

Currently, there are harmonised standards against MDD and IVDD, but the harmonisation process with MDR and IVDR is being delayed. Pitkänen and others [Pitkänen et al., 2020] suggest using standards that are currently harmonised against MDD, IVDD and AIMDD, until there are harmonised standards for MDR and IVDR. Newer versions of the standards that have been harmonised, exist already. However, European Commission has adopted a new request for harmonisation of medical device standards against MDR and IVDR [European Commission, 2021]. Hence, the request represents the upcoming state of standardisation of MDR and IVDR, and gives an idea of what standards should be followed in the future.

IEC 62304 and IEC 82304-1 were chosen to be the main standards under discussion for this thesis, as they introduce requirements for the software life cycle processes and the product safety, and impact the software development and delivery the most. IEC 82304-1 applies to medical device software and software with other health uses, while IEC 62304 includes standalone medical device software and software as part of a medical device. However, the standards overlap at software-only medical device, as illustrated in Figure 3, presenting the relationship between IEC 62304 and IEC 82304-1.

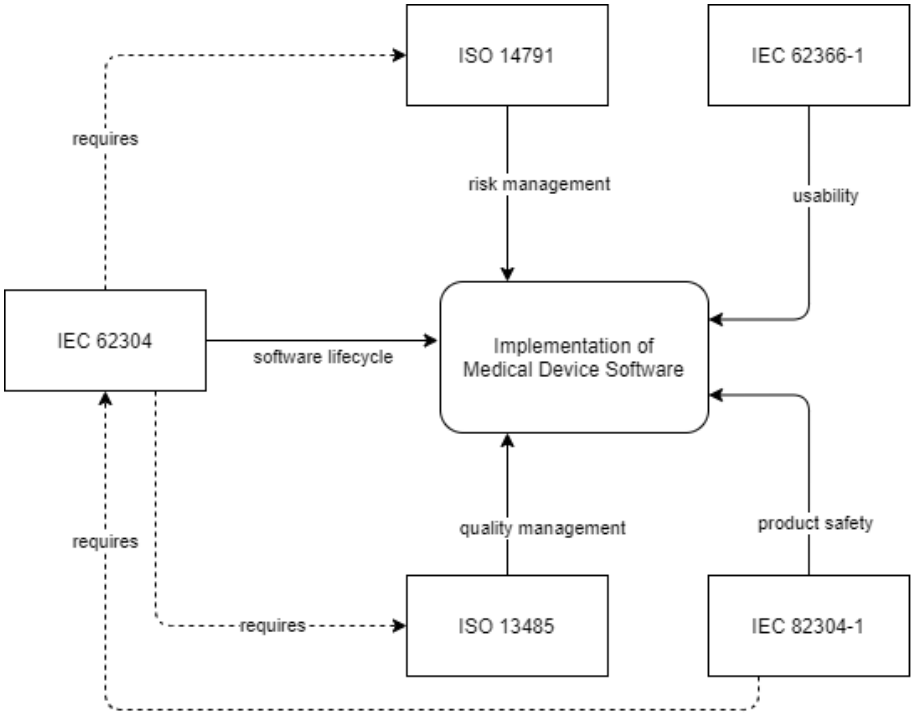

Figure 2 *Relationship of medical device software and the related standards.*

2.5.1 IEC 82304-1: Health software — Part 1: General requirements for product safety

The international standard IEC 82304-1 applies to safety and security of health software products. A health software is independent of dedicated hardware and may be run on a general computing platform [IEC, 2017]. The primary purpose of the standard, is to introduce product safety-related requirements to medical device software manufacturers [IEC, 2016]. The standard contains direct reference to use software life cycle processes from IEC 62304, but it introduces more safety-related requirements for the manufactured product, which are not covered by IEC 62304.

The standard introduces additional layer of software requirements, the health software use requirements, which are transformed into system requirements, and can be utilized by the development process, which follows IEC 62304. The health software use requirements are used as an input for the development process, and are the implementation of the requirements is validated after the product development has been finished and before the product can be delivered to the customer. Validation of the health software is introduced in the standard as an independent activity. Essentially the standard requires the medical device software manufacturer to establish a validation plan and validate the product accordingly.

Product identification and accompanying documents are required by the stan-

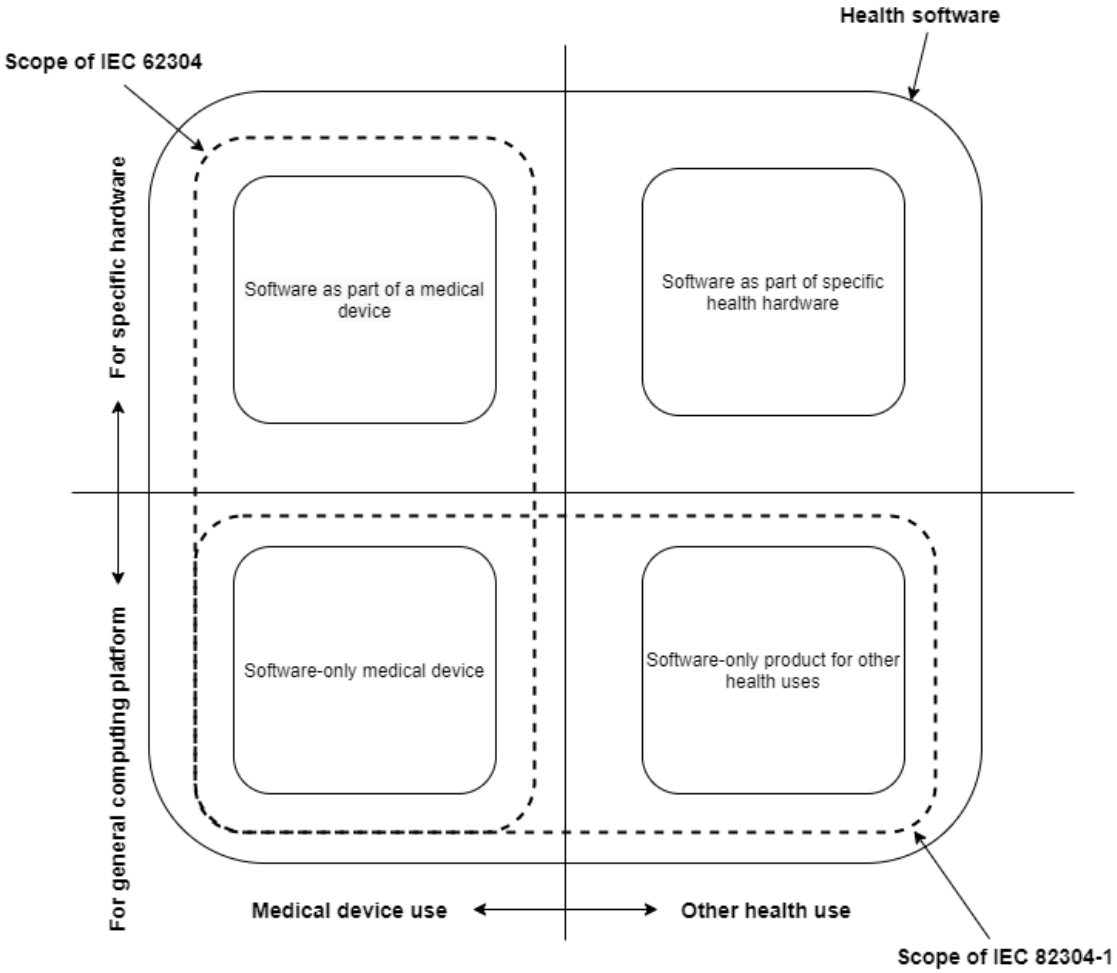

Figure 3 *Application of the standards IEC 82304-1 and IEC 62304 to the health software domain [IEC, 2016].*

dard. Essentially, the product identification means marking the device with a Unique Device Identifier (UDI) in European Union region. The accompanying documents introduce general documentation of the product, instructions for use, and a technical description. Finally, the standards covers activities which apply to the product, after is has been deployed into real world use. The post-market activities include maintenance and re-validation of the product, decommissioning and disposal, and post-market communication.

2.5.2 IEC 62304: Medical device software — Software life cycle processes

IEC 62304 is an international standard, defining the requirements for the whole life cycle of a medical device software [IEC, 2015]. The standard introduces regulatory activities for requirements management, design, development, testing, releas-

ing, documentation and maintenance of the software. However, the standard only defines the processes and activities, but does not state exactly how to implement them, leaving the responsibility for the manufacturer, to apply the standards into the manufacturing process correctly. The manufacturer is responsible for selecting a development model, which can be integrated into the regulatory activities. However, this allows the integration of the standards, and the modern software development practices, as long as the activities and tasks from the standard are implemented.

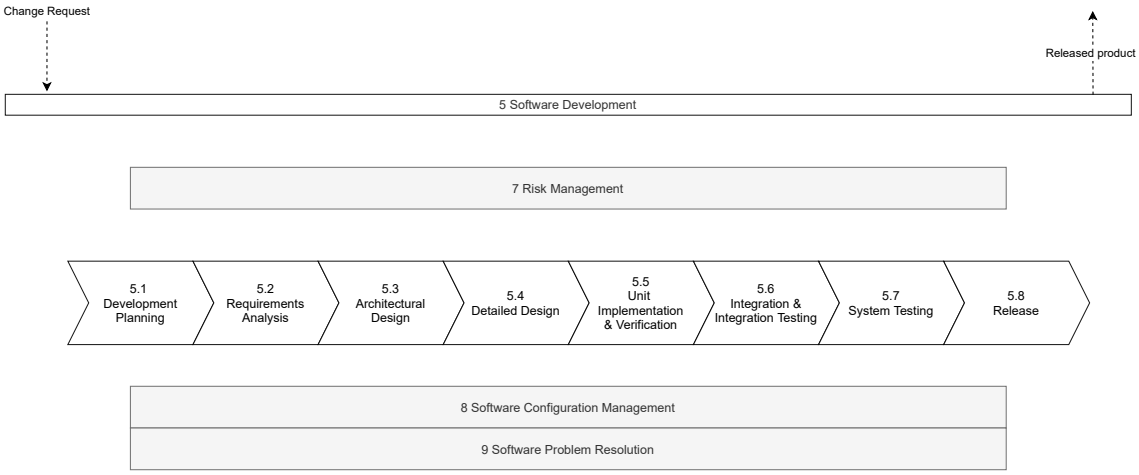

Figure 4 *Sequential order of the activities from the standard IEC 62304 [IEC, 2015].*

The development process defined by IEC 62304, is illustrated in Figure 4. The standard defines a set of software development activities, and tasks that are described sequentially in the standard. Some of the activities use output of the previous step as an input, which must be noticed in the overall process. For example, there can't be a software detailed design document, before the software requirements are composed for that specific scenario.

Table 1 *Safety classes of IEC 62304 [IEC, 2015].*

Class	**Risk severity qualification**
Class A	No injury or damage to health is possible
Class B	Non-serious injury is possible
Class C	Death or serious injury is possible

The manufacturer must classify the software into a software safety class, defined by IEC 62304 [IEC, 2015], and listed in Table 1. The software safety classification must be done to every software item/component separately, to segregate the medical device components by the risk severity, enabling the implementation risk management requirements for the system, and the risk control activities to prevent any dangerous outcomes. Every part of the software system must be implemented

according to the safety class. The safety class is inherited from the top-level component, when implementing new features into the software, unless the manufacturer can rationalize the segregation of the software modules [Pitkänen et al., 2020].

However, the software safety classification of IEC 62304 does not directly map into product classifications, introduced by MDR or IVDR [Pitkänen et al., 2020]. The intended purpose, as planned by the manufacturer, is one of the main factors in the decision of the product classification [Pitkänen et al., 2020]. Additionally, the interpretation is left for the manufacturer. The manufacturer holds full responsibility of the product, and the compliance with the regulatory requirements.

Software risk management is introduced in the standard as a core activity for software development. Essentially, the whole manufacturing process of the medical device, focuses on managing and mitigating risks. The standard requires the manufacturer to analyse hazardous situations, that the software could contribute into.

Software configuration management is identified in the standard as a process for identifying the configuration items, and controlling the configuration changes of the software. However, configuration in the perspective of the standard includes any piece of information, that is created during the manufacturing process of the software, including the product documentation.

Finally, any modifications to a released version should use a maintenance process, and developing a new version should use the software development process. The maintenance process may use lighter processes to fix any defects found from the released product.

2.5.3 Other relevant standards

Other relevant standards to the development of medical device software are ISO 14971, ISO 13485 and IEC 62366-1. However, these standards are ignored in the identification of the requirements for a CI/CD pipeline, because they do not directly introduce activities for the software development/delivery. It should be mentioned, that the standards introduced in this chapter are still recommended for implementing a medical device software, if not necessary.

ISO 14971 is a standard for medical device risk management [ISO, 2007]. A medical device software manufacturer must conform to ISO 14971 as stated in IEC 62304. The manufacturer must have an active risk management process, and the residual risk of the medical device must be reduced to an acceptable level, before the product can enter the market. The standard assumes, anything that can impact the medical device, could be a risk, while every identified risk should be reduced to an acceptable risk-level to prevent damage to person's health. Requirement management is associated with risk management, as the identified risks might alter existing

requirements, or introduce completely new requirements for the system. Finally, mitigating a risk could also introduce new risks.

ISO 13485 introduces requirements of quality management systems for medical device manufacturers [ISO, 2018]. Essentially, a medical device manufacturer is required to have a Quality Management System (QMS) [Pitkänen et al., 2020]. Implementing a QMS suggests, that the manufacturer has established processes, that conform to the regulatory requirements. It is presumed by the regulators, that the medical device manufacturer has a QMS, that complies with the requirements of ISO 13485 [Pitkänen et al., 2020].

IEC 62366-1 introduces requirements for usability engineering for medical devices. The purpose of the standard is to negate use-related risks from the product by applying risk management procedures to the user interface of the medical device. The manufacturer must address the unpredictable use scenarios and minimize any use errors that could occur [IEC, 2015]. Additionally, the primary operation functions of the software must be usability tested properly.

2.6 Agile software development in regulated environment

There is no direct instruction to use any specific software development model in IVDR, MDR or any of the medical device software standards, or guidance documents. IEC 62304's requirements for software development can be confused with the waterfall-oriented model, and it can be relatively difficult to implement in a modern software development environment. Moreover, IEC 62304 recognizes the existence of waterfall, incremental and evolutionary development strategies and the manufacturer is encouraged to choose the appropriate strategy for the project, as long as the following core principles of the standard are satisfied [AAMI, 2012], [IEC, 2015]:

- All the process outputs should be maintained in consistent state, meaning every document and software item should be updated, when a related item changes.
- All process outputs should be available as input when needed, meaning that some parts of the process can't be started before others are finished.
- Before the release, every process output should be consistent with each other and all dependencies, explicitly or implicitly, stated by the standard should be observed.

In the document AAMI:TIR45 [AAMI, 2012], agile practices for software development are utilized, while maintaining compliance with the standard, as illustrated

in the Figure 5. Medical device software standards require extensive planning and technical documentation of the implementation, whereas in agile-oriented development the planning is done in an iterative manner, and there is no requirement for extensive documentation. However, the requirement for extensive technical documentation suits for mindset of high-quality software products, as it makes maintenance and further customization of the software less complicated. The medical device standards consider the documentation as a part of the product.

IEC 62304's activities and tasks are mapped into an iterative software development process, as illustrated in the Figure 5. In this particular scenario, most of the activities are done for every task, such as requirements analysis, architectural design, detailed design, unit implementation and verification, software integration, integration testing and system testing. For each increment and release following activities are performed; software integration, integration testing, system testing and regression testing. Moreover, for each release, the release procedures are performed as well. On the project level, there are emerging documents such as software development plan, requirement analysis document, architectural design document. The documents evolve through iterations, and should be versioned for each release.

In this thesis, the agile workflow is not the main focus, but idea of the the agile workflow in the regulated environment, illustrated in Figure 5, has been used as one of the sources, for incubating ideas.

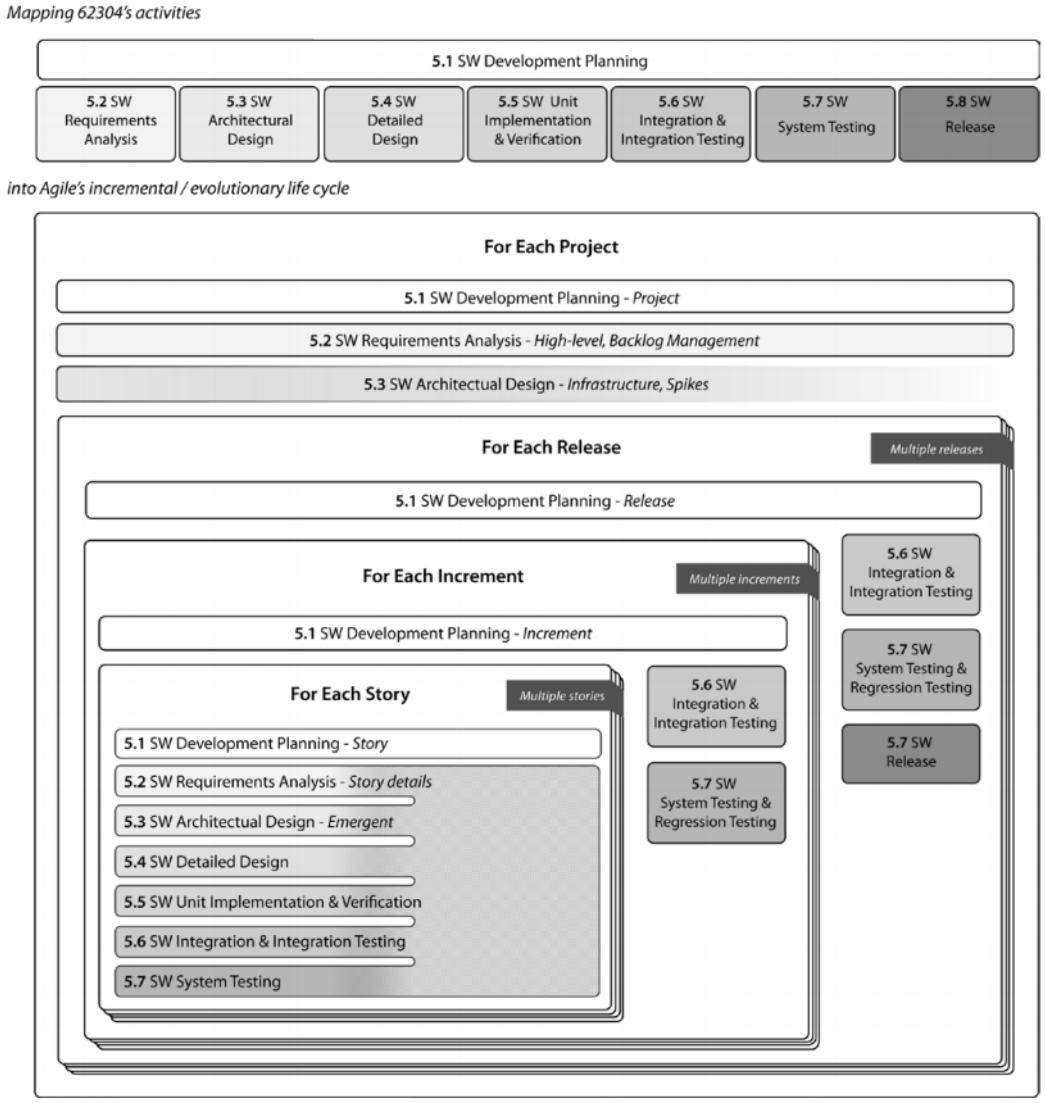

Figure 5 *Mapping 62304's activities into Agile's incremental / evolutionary life cycle [AAMI, 2012].*

3 Key characteristics of CI/CD pipelines

In this chapter, the key characteristics related to DevOps and CI/CD pipelines are discussed. While there is not an exact definition for DevOps, the purpose of DevOps has been described as in "bridging the gap between development and operations" [Wettinger et al., 2014]. This essentially means a set of practices that software developers and IT operators have agreed upon, to reduce the complexity in software delivery. Another aspect of DevOps is defined as "practices that reduce and join repetitive tasks with automation in development, integration and deployment" [Laukkarinen et al., 2018]. It is enabled by three main characteristics; capabilities, cultural enablers and technological enablers [Smeds et al., 2015]. DevOps could be described as a mindset of systems thinking, process automation, agile principles, open communication, data-driven decision making, continuous iteration and people-first culture. The main promise of DevOps is to enable fast and efficient releases. By releasing often, it is possible to create a feedback loop, which can be utilized to quickly adapt into changing customer requirements. This can be crucial to establish a business advantage for a software company, when the customer requirements are a moving target [Wettinger et al., 2014].

DevOps helps software teams by automating tasks, that are performed repeatedly during the development cycle, through a combination of different tools/practices. The automated activities are performed in so-called pipelines, which are triggered on-demand. In this thesis, process automation and software delivery aspects of DevOps are studied more closely.

3.1 Continuous Integration

The definition of Continuous Integration (CI) is the practice of integrating new code frequently, preferably as soon as possible [Duvall et al., 2007]. The integration is done by running automated build and automated tests on the software, verifying the changed software in the pipeline. The purpose of the CI is to prevent the code from diverting too much between the developers and to keep the code constantly intact, ready for release.

Continuous Integration is enabled by having the code in a code repository, which can be offered by any modern distributed version control system (VCS), such as Git [Git, 2021] or Mercurial [Mercurial, 2021]. The build process of the software should be automated into a point, where a single command can build the whole software, which enables triggering a build for every change in the software [Duvall et al., 2007].

Continuous Integration pipeline performs automated tests and other verification

methods to new and already existing code [Duvall et al., 2007]. The automated verification process is run for every committed change in the software. If the software has high unit test coverage, the test automation can find defects that would otherwise be left unnoticed [Spinellis, 2017]. This leads to CI acting as a guard for developers for maintaining code quality and integrity. If the build fails, the developer must either fix the defect or revert the changes to fix the build.

The software integration should be done in a normalised environment, preferably a clone of the production environment to ensure the software will work in production as intended [Duvall et al., 2007]. There are often nuances to the integration environment, depending on the complexity of the overall system. In an optimal scenario, correct implementation of CI reduces software risks by early discovery of defects, improving project visibility, improving code quality and testing the software in an environment equivalent to production [Duvall et al., 2007]. The application of CI into the development process of a software has been found to increase the focus on the quality of the product and the test applications [Amrit et al., 2018].

3.2 Continuous Delivery

In this study, the definition of Continuous Delivery (CD) is the automation of the software release to a point where the software could be released at any given moment. The purpose of the Continuous Delivery is to ensure that new features of the software can be delivered to the point of use on-demand. To clarify terminology, in this study, Continuous Deployment means the automated deployment of software into a computational production environment, while making it instantly available to end-users [Laster, 2020], without human intervention, in contrast to Continuous Delivery, where the release to the end users requires a human approval.

Before Continuous Delivery can be applied effectively into the software development and delivery, the software must meet set of architecturally significant requirements such as testability, deployability and modifiability [Rossberg, 2014]. These requirements can be difficult to achieve for certain systems. Also, it is necessary to implement a deployment pipeline, which enables the automated deployment of the software into a target software environment [Humble et al., 2011]. The deployment pipeline utilizes packaged software binaries, software artifacts, which are installed along with the version-specific software configuration. This whole process can be automated into the point, where a single command can execute the whole software deployment, and even verify, that the software is started after the deployment.

Benefits of the automated deployment of changes are significant for the development teams, as there is no personnel needed to deploy the software. While the pipeline handles the deployment, thus the development environment is constantly updated with fresh features, the quality assurance personnel may verify the changes

constantly, without waiting for the deployment to happen.

Additionally, when the whole deployment process is automated, and documented in the code, it is transparent for everyone, including every detail to get the whole software system up and running. This can help the whole manufacturing organisation to understand, what is happening within the product, and can reduce time needed for the product delivery [Humble et al., 2011]. Moreover, removing human from the deployment process also removes the possibility of the human-made error, which could be a reason for a failure in the software deployment [Humble et al., 2011].

Finally, if the product can be delivered to the point of use more efficiently, the customer's feature wishes could be implemented and delivered much more frequently, which could increase the customer satisfaction, making the software more valuable for the end users.

3.3 Pull-based development

Pull-based development has become popular in the open-source communities of software development, after the introduction of distributed version control systems. Essentially, when multiple developers work with same project, they fork the main repository and push their changes into the original repository, where the developers responsible of the project may decide whether the changes are integrated [Gousios et al., 2016]. The developers, responsible of the feature integration, are notified with the means of *pull requests* [GitHub, 2021] or *merge requests* [GitLab, 2021], which are essentially the platform-specific terms of a notification for a integration-ready feature in the software.

The software change is assessed in the pull request, and it offers a solid basis for distributed collaboration. Results of any verification tasks, whether automated or manual, can be incorporated into pull request with additional tooling, making the pull request a valuable tool for software developers. Ideally, a pull request can be merged automatically, after all verification activities have been completed, and the change is approved.

3.4 *-as-Code

In this section, the *Infrastructure-as-Code* (IaC) and *Configuration-as-Code* (CoC) are introduced. Essentially, anything related to software product could be treated as code. Ideally, to enable reliable ways to construct software environments, not only the software source code is treated as code, but also the whole infrastructure of the computational environment where the software is executed, including the software configuration for different environments, and the product documentation. Storing

software, and it's whole infrastructure as code could be seen as a prerequisite for CI/CD pipelines [Humble et al., 2011]. By treating the whole software infrastructure as code, it is possible to automate more tasks, and remove the possibility of human-error from the process.

Infrastructure-as-Code is the practice of storing the software infrastructure related items into the VCS [Morris et al., 2016]. Essentially, the software infrastructure is treated as code, and the infrastructure is always built from the same scripts. Meanwhile, the scripts also documents how the system is created.

Configuration-as-Code is the software configuration stored in a version control system. Configuration files are stored in a version control system, and changes can be revisited from commit history. Even if the configuration is stored into the version control system, there should still be a document describing what configuration options are available and how to configure the system. The practice offers possibility for easier maintenance of the configuration, as the software configuration is in one place, and the manual documentation needs are less significant.

When the whole software product is treated as code, it enables version control for any change in the product, as well as the automation possibilities for any repeatable part of the software life cycle processes [Humble et al., 2011]. Essentially, if the whole software product is within the same VCS, it can be developed, reviewed and verified more efficiently.

3.5 Blue-green deployment

Blue-green deployment is the practice of establishing two identical copies of the computational production environment for the software. One of the environments acts as the *production slot* and the other one is the *staging slot*. The software can be deployed into the staging slot at any given time, but the software release is performed by swapping the production and the staging slot over. The swapping could be done by directing the traffic through a router component, which handles the traffic forwarding to the correct environment [Humble et al., 2011].

The advantages of Blue-green deployment are zero-downtime deployments and the ability to rollback the software update, if anything goes wrong. In the ideal situation, a software deployment for the customer does not disturb the production or cause serious downtime, making the update seamless.

Managing databases, while utilizing Blue-green deployment, can be difficult, as the production database might require data migrations, or the database needs to be switched between the environment. However, this can be accounted in the architectural design of the software system and deployment practices.

3.6 Reference CI/CD pipeline

Together, CI and CD enables the composition of CI/CD pipelines, which can perform automated verification activities on the software, and deploy the software into a specific computational environment. A reference model for a CI/CD pipeline is illustrated in Figure 6 [Khan, 2020]. The idea behind the pipeline is to enable rapid software delivery to development, quality assurance (QA) and production environments, while performing series of verification tasks in the form of approvals and gates. A software developer checks the software change into the upstream of the version control system, triggering the CI pipeline, which performs the automated verification activities against the check-in and verifies the software integrity. After the CI pipeline has passed successfully, the CD pipeline deploys the software into a computational software environment. However, the deployment is gated by a human approval. At first, the new version is deployed into the development environment, where the software developer can test the changed code in a production-like environment. After the development is finished, and the change is approved, the software is deployed into QA the environment to test and verify the new functionality. Essentially, the software is tested in more details with either automated or manual tests. After software is tested and verified, it can be staged into production environment's staging slot. The production environment has two slots; one for staging and one for production. The code is always first staged into the production's staging slot. After a release approval, the software can be put into possession of the end-user by swapping the staging slot with the production slot, making the software available to the end users. Additionally, if any problems are noticed after the swap, the production environment can be restored by swapping the staging and the production slots over again.

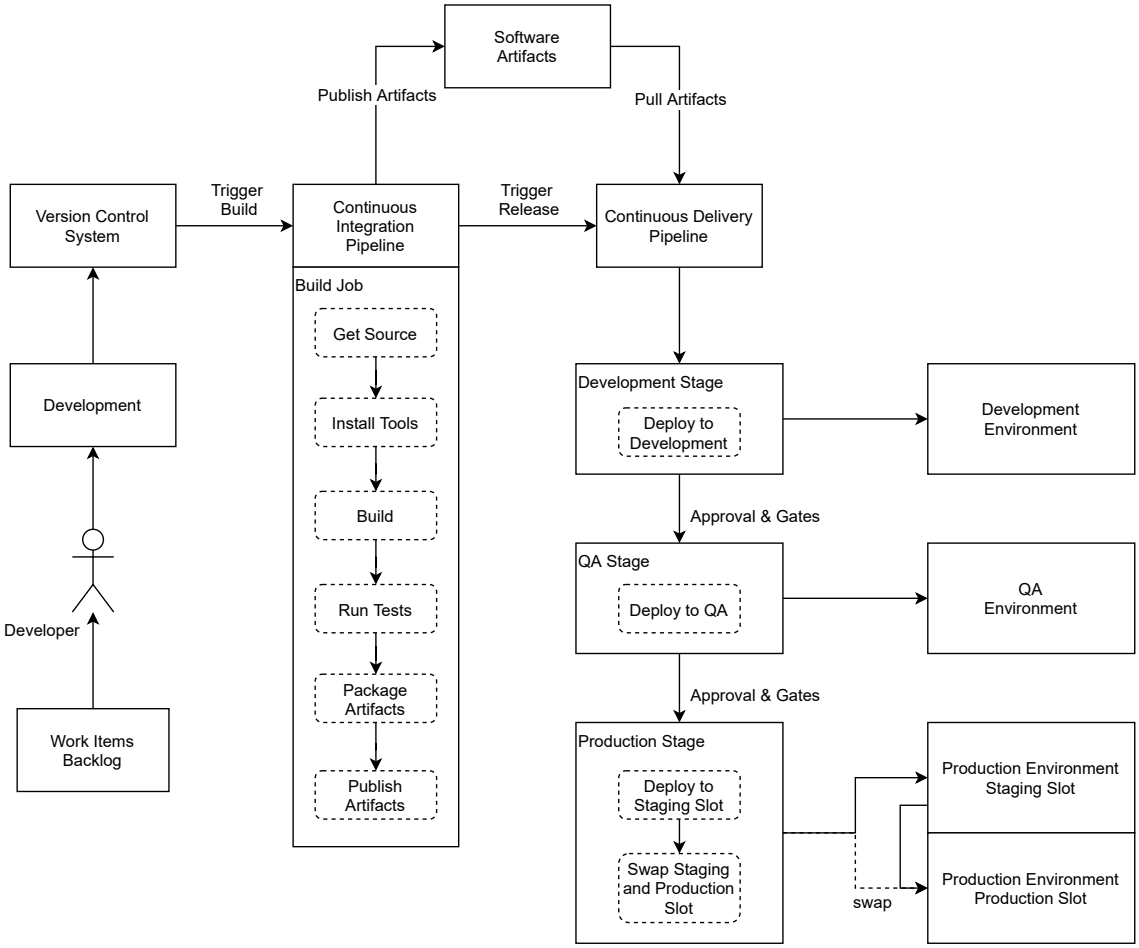

Figure 6 *A reference illustration of a CI/CD pipeline, performing the automated verification, and delivery of the software into real world use [Khan, 2020].*

4 Identifying requirements for CI/CD pipelines in the regulated environment

In this chapter, the requirements from the standards IEC 62304 and IEC 82304-1, that can be applied to a CI/CD pipeline, such as the pipeline that was illustrated in Figure 6, are identified and collected into tables. The aforementioned standards were selected, because from all generally applicable medical device standards, IEC 62304 and IEC 82304-1 impact the software development and delivery process the most. However, it should be mentioned, that these standards do not cover every regulatory requirement, and depending of the product, other standards may apply. The standards were analysed in an iterative process, with an objective to produce a systematic analysis, to extract every regulatory requirement that could be applied to a CI/CD pipeline. The identified requirements were divided into the following categories;

1. The requirement can be fully implemented in a CI/CD pipeline.
2. The requirement can be partially implemented in a CI/CD pipeline.
3. The requirement can not be implemented in a CI/CD pipeline.

There were total of 108 requirements identified from the standards IEC 62304 and IEC 82304-1, and a total of 26 requirements could be fully implemented in a CI/CD pipeline. Additionally, total of 20 requirements were identified, that could be partially implemented in a CI/CD pipeline. Finally, total of 62 requirements were scoped out of the pipeline, that can not be implemented in a CI/CD pipeline.

4.1 Issues arising from the regulatory requirements

In this section, the difficulties related to regulatory requirements are introduced. From the regulatory compliance perspective, it is presumed that every applicable requirement is implemented correctly. However, the application of the requirements into the software development life cycle processes, creates an additional layer of complexity for the software manufacturer. Furthermore, the software manufacturer might not be able to adapt the best practices of the software development industry into the medical device software development standards, leading to slower and more burdensome software development and delivery.

4.1.1 Rigidness of the manufacturing process

The standard IEC 62304 [IEC, 2015] introduces many activities to software development, which could make the manufacturing process of the software more challenging. While the software industry has the mindset for rapid development, and delivery of a software to maximize the customer value, the requirements from medical device regulations could slow down the manufacturing process significantly. For instance, to meet with the regulatory compliance, the manufacturer must establish and maintain comprehensive product documentation. For every software change, the documentation must be updated, leading to laborious and cumbersome tasks, if done manually. By having more work with every individual change in the software, the process lead time, essentially meaning the time from the start to finishing a process, could be increased drastically. If on average, more time is needed to change the software, it would take more time to deliver value to the customer.

Additionally, it is expected that all the relevant activities are performed during the development process, of which most of are tasks, that would not be performed in an unregulated software development process. However, some of these activities can be automated, at least partially, to some extent. It is worth noting, that depending on the software safety class, some of the regulatory activities can be ignored completely. However, if the software safety class of the software is adjusted to require more rigid manufacturing process, for example a transition from Class B to Class C, the missing documentation could be hard to produce afterwards.

4.1.2 Burdensome testing and verification requirements

In the regulated environment, the safety of the patient comes first. If a medical device software does not work as intended, it might introduce risks for the safety of the patients. A medical device software manufacturer must possess evidence, that the software system is working as intended, leading to various verification and testing activities, performed against the system. As a result, a change in the software must be tested comprehensively to ensure, that the change does not introduce unwanted side-effects, that could inflict patient risks. If the verification is done manually, performing a comprehensive set of tests against every software change can be laborious and expensive. Furthermore, the tests must be planned and documented in the first place, leading to more rigid manufacturing process.

As a solution, test automation can help to reduce the burdensome verification activities from the development process, at least to some extent. However, the implementation and maintenance of the test automation requires skilled personnel, which increases the total cost of manufacturing a medical device software. Additionally, it might not be possible to automate every possible test scenario, and there

could still be a need for manual testing, leading to a situation, where the verification and testing activities slow down the product delivery process.

4.1.3 Software of Unknown Provenance

Software of Unknown Provenance (SOUP) refers to a software, or part of a software, that is not intended for medical use, but is incorporated into a medical device. It is worth nothing, that SOUP includes parts of software, that have been developed before the medical device development processes have been available, if the development has started before the establishment of the medical device development processes. The manufacturer of medical devices must evaluate every SOUP component, incorporated into the medical device. In practice, the functional, performance, software and hardware requirements, necessary for its intended purpose, are to be identified and documented [IEC, 2015]. Any change to a SOUP component means, that the SOUP analysis must be performed again, identifying the risks and hazards, that the change could cause. Additionally, there must be a plan for software configuration and change management for SOUP components [IEC, 2015].

SOUP is a troublesome area for medical device manufacturers, that have developed software systems, before MDR or IVDR were applicable into them. The systems could be decades old, and the documentation of the software creation could be missing, thus most of the system would be considered SOUP. Additionally, if the documentation is missing, it is difficult to determine later, how the system is supposed to work. As a result, from the regulatory perspective, any change on the system could be laborious to perform.

Another challenge SOUP introduces, is that the modern software industry depends on general-purpose libraries, usually developed outside the manufacturing organization. It would be very time-consuming and hard for the manufacturer to implement and sustain every generally used library themselves, and it would push them outside of the best-practices of software development industry, utilized by everyone else.

Running the software on general computing platforms introduces additional layer(s) of complexity and difficulties to demonstrate the conformity of the product [Granlund et al., 2020], leading to a scenario, where a major part of the software could be considered SOUP. The responsibilities between the medical device software manufacturer and computing platform provider should be clarified in the documentation. However, to conform with the regulations, the utilization of general cloud computing platforms could introduce more documentation work, making the manufacturing of the medical devices more laborious.

The SOUP requirements could discourage the utilization of general-purpose libraries and cloud computing platforms, which makes the software development more

volatile and laborious, as most generally used solutions are already in a very mature stage, and verified to be able to perform well in production. In the worst case scenario, in the regulated environment, a medical device software manufacturer might have to reinvent the wheel, to solve a problem.

4.2 Fully implemented regulatory requirements

In this section, the fully implemented requirements from the standards IEC 62304 and IEC 82304-1 are presented (see Table 2). There were total of 26 requirements identified, that could be fully implemented in a CI/CD pipeline. The criteria for the requirements were, that activity must be fulfilled after the code has been checked in to the version control system, and before the software is released to the real world use. The identified requirements contain activities, which can be performed in a CI/CD pipeline, either automatically or manually. Some of the identified regulatory activities require manual inspection on the software product, thus must be done manually. However, automation could be possible for some of the documentation, verification or testing activities, with additional tools in the pipeline. Therefore, the identified requirements are fully implementable, but could require specific tooling or manual work in the pipeline. Further analysis of the fully implementable requirements is in Chapter 5, where the identified requirements are integrated into a CI/CD pipeline model.

Table 2 *Requirements of the standard IEC 62304 and IEC 82304-1 that can be fully implemented in a CI/CD pipeline.*

Standard	**Item**	**Safety class**	**Title**
IEC 62304	5.3.6	B, C	Verify SW architecture
IEC 62304	5.4.4	C	Verify detailed design
IEC 62304	5.5.5	B, C	SW unit verification
IEC 62304	5.6.1	B, C	Integrate SW units
IEC 62304	5.6.2	B, C	Verify SW integration
IEC 62304	5.6.3	B, C	Test integrated SW
IEC 62304	5.6.4	B, C	Integration testing content
IEC 62304	5.6.5	B, C	Verify integration test procedures
IEC 62304	5.6.6	B, C	Conduct regression tests
IEC 62304	5.6.7	B, C	Integration test record contents
IEC 62304	5.7.4	A, B, C	Verify SW system testing
IEC 62304	5.7.5	A, B, C	SW system test record contents
IEC 62304	5.8.1	A, B, C	Ensure SW verification is complete
IEC 62304	5.8.3	B, C	Evaluate known residual anomalies
IEC 62304	5.8.4	A, B, C	Document released versions
IEC 62304	5.8.6	B, C	Ensure activities and tasks are complete
IEC 62304	5.8.7	A, B, C	Archive SW
IEC 62304	7.3.1	B, C	Verify risk control measures
IEC 62304	7.3.3	B, C	Document traceability
IEC 62304	8.1.2	A, B, C	Identify SOUP
IEC 62304	8.1.3	A, B, C	Identify system configuration documentation
IEC 62304	9.8	A, B, C	Test documentation contents
IEC 82304-1	6.2	n/a	Performing validation
IEC 82304-1	6.3	n/a	Validation report
IEC 82304-1	7.1	n/a	* Identification
IEC 82304-1	8.3	n/a	Re-validation

4.3 Partially implemented regulatory requirements

In this section, the partially implemented requirements from the standards IEC 62304 and IEC 82304-1 are presented, illustrated in Table 3. The partially implemented requirements are partially fulfilled either before or after the activities of the CI/CD pipeline are performed. However, a CI/CD pipeline can support the implementation of the regulatory requirements, but not fulfill the requirement completely. There were total of 20 requirements identified that could be partially implemented

in a CI/CD pipeline.

The planning activities for *Identification and avoidance of common SW defects* (IEC 62304 clause 5.1.12) must be done before the software is developed, but a CI/CD pipeline could support the requirement by running automated code analysis tools for every new check-in [Wichmann et al., 1995].

Many of the regulatory requirements involve design activities, which are performed before the pipeline is triggered. However, the pipeline the pipeline can support the aforementioned activities by generating the resulting documentation. For instance, the pipeline can generate documentation (IEC 62304 clauses 5.3.1, 5.3.2, 5.3.3, 5.3.3, 5.4.1) for software architecture, software units and SOUP items. Additionally, the process for software unit verification process (IEC 62034 clause 5.5.2), and the software unit acceptance criteria (IEC 62034 clause 5.5.3) must be established before the pipeline, but the pipeline can be utilized as a tool to verify the software unit. Ultimately, the pipeline could compile all documentation, required by the regulatory requirements, including accompanying documents (IEC 82304-1 clause 7.2).

The software use requirement management is mostly done outside of the pipeline, but the requirements may need to be updated during the development (IEC 82304-1 clauses 4.4, 4.7). Moreover, tests for software requirements (IEC 62304 clause 5.7.1, 8.2.3) are established before the pipeline, but the testing activities are performed within the pipeline, verifying the changes made into the software. Additionally, the requirement to retest after changes (IEC 62304 clause 5.7.3) is partially implemented, as the pipeline can be utilized for the testing activities, but the risk management activities are performed outside of the pipeline. Finally, risk management could be automated to certain degree, but the verification is done within the pipeline (IEC 62304 clauses 5.8.2, 7.4.3).

Some of the requirements are implicitly built into a CI/CD pipeline, such as the repeatability of software release (IEC 62304 clause 5.8.8), re-releasing of the modified software system (IEC 62304 clause 6.3.2), and documenting how the software was created (IEC 62304 clause 5.8.5). Furthermore, the pipeline could offer tools for checking the existence of SOUP anomaly lists, and to provide a convenient way to update the missing SOUP components (IEC 62304 clause 7.1.3).

Table 3 *Requirements of the standard IEC 62304 and IEC 82304-1 that can be partially implemented in a CI/CD pipeline.*

Standard	Item	Safety class	Title
IEC 62304	5.1.12	B, C	Identification and avoidance of common SW defects
IEC 62304	5.3.1	B, C	Transform SW requirements into an architecture
IEC 62304	5.3.2	B, C	Develop an architecture for the interfaces of SW items
IEC 62304	5.3.3	B, C	Specify system functional and performance required by SOUP items
IEC 62304	5.3.4	B, C	Specify system hardware and SW required by SOUP items
IEC 62304	5.4.1	B, C	Refine SW architecture into SW units
IEC 62304	5.5.2	B, C	Establish SW unit verification process
IEC 62304	5.5.3	B, C	SW unit acceptance criteria
IEC 62304	5.7.1	A, B, C	Establish tests for SW requirements
IEC 62304	5.7.3	A, B, C	Retest after changes
IEC 62304	5.8.2	A, B, C	Document known residual anomalies
IEC 62304	5.8.5	B, C	Document how released SW was created
IEC 62304	5.8.8	A, B, C	Assure repeatability of SW release
IEC 62304	6.3.2	A, B, C	Re-release modified SW system
IEC 62304	7.1.3	B, C	Evaluate published SOUP anomaly lists
IEC 62304	7.4.3	B, C	Perform risk management activities based on analyses
IEC 62304	8.2.3	A, B, C	Verify changes
IEC 82304-1	4.4	n/a	Updating health SW product use requirements
IEC 82304-1	4.7	n/a	Updating health SW product system requirements
IEC 82304-1	7.2	n/a	Accompanying documents

4.4 Not implemented regulatory requirements

In this section, the scoped out requirements from the standards IEC 62304 and IEC 82304-1 are presented, illustrated in Table 4. The requirements in this section are fulfilled either before or after the pipeline activities. There were total of 62 requirements scoped out of the pipeline.

The standards contain clauses, which are not activities, but more of presumptions of things, which can't be formed into a repeatable task. For example, a quality management system (IEC 62304 clause 4.1) and risk management (IEC 62304 clause 4.2) affect the whole manufacturing organization and all it's functions. The general requirements and initial risk assessment (IEC 82304-1 clause 4.1) affects the product documentation, but not directly the pipeline. Software safety classification is decided

outside of the pipeline, but it affects the manufacturing process rigidity (IEC 62304 clause 4.3), and quantity of the regulatory requirements from the IEC 62304, which apply to the pipeline. A CI/CD pipeline could support legacy software, but it does not directly affect the implementation of the pipeline (IEC 62304 clause 4.4).

The scoped out requirements are mostly planning or design activities. Ultimately, all planning activities were scoped out, such as software development plan, requirements management, technical design and validation planning activities (IEC 62304 clauses 5.1.1 - 5.1.9, 5.1.11, 6.1, 8.1.1, 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, 7.2.2, 7.4.1, 7.4.2. 8.2.1, 8.2.2) (IEC 82304-1 clauses 4.2, 4.3, 4.5, 4.6, 6.1). Also, the pipeline needs to be addressed in the supporting items management (IEC 62304 clause 5.1.10), but not vice versa.

The software implementation activities were also scoped out, as a CI/CD pipeline is triggered after the implementation of the software, including the associated documentation, has been performed (IEC 62304 clauses 5.5.1, 6.3.1). Moreover, the software problems resolution process is managed outside of the CI/CD pipeline, but may feed input into the pipeline (IEC 62304 clauses 9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7). Furthermore, the problem and modification analyses are performed before the pipeline (IEC 62304 clauses 6.2.1, 6.2.2, 6.2.3. 6.2.4, 6.2.5, 6.2.6), thus scoped out.

The responsibility of the manufacturer does not end after the software has been delivered into the point of use. For instance, the change requests, associated approvals, and problems reports are managed outside of the pipeline (IEC 62304 clause 8.2.4). Finally, the post-market activities are performed after the pipeline (IEC 62304 clause 8.4, 8.5).

Even if all of the regulatory requirements can not implemented within the CI/CD pipeline, the medical device software manufacturer must apply the requirements into the overall software development process.

Table 4 *Requirements of the standard IEC 62304 that can not be implemented in a CI/CD pipeline.*

Standard	**Item(s)**	**Title**
IEC 62304	4.1	Quality Management System
IEC 62304	4.2	Risk management
IEC 62304	4.3	SW safety classification
IEC 62304	4.4	Legacy software
IEC 62304	5.1.1 - 5.1.11	SW development plan
IEC 62304	5.2.1 - 5.2.6	SW requirements analysis
IEC 62304	5.3.5	Identity segregation necessary for risk control
IEC 62304	5.4.2	Develop detailed design for each SW unit
IEC 62304	5.4.3	Develop detailed design for interfaces
IEC 62304	5.5.3	Additional SW acceptance criteria
IEC 62304	5.5.1	Implement each SW unit
IEC 62304	6.1	Establish SW maintenance plan
IEC 62304	6.3.1	Use established process to implement modification
IEC 62304	7.1.1 7.1.2 7.1.4	SW risk management process
IEC 62304	7.2.1 7.2.2	Risk control measures
IEC 62304	7.4.1	Analyse changes to medical device SW with respect to safety
IEC 62304	7.4.2	Analyse impact of software changes on existing risk control measures
IEC 62304	8.1.1	Establish means to identify configuration items
IEC 62304	8.2.1	Approve change requests
IEC 62304	5.6.8 5.7.2	Use SW problem resolution process
IEC 62304	6.2.1 - 6.2.6	Problem and modification analysis
IEC 62304	8.2.2	Implement changes
IEC 62304	8.2.4	Provide means for traceability of change
IEC 62304	8.3	Configuration status accounting
IEC 62304	9.1 - 9.7	SW problem resolution process
IEC 82304-1	4.1	General requirements and initial risk assessment
IEC 82304-1	4.2	Health SW product use requirements
IEC 82304-1	4.3	Verification of health SW product use requirements
IEC 82304-1	4.5	Updating health SW product use requirements
IEC 82304-1	4.6	Verification of system requirements
IEC 82304-1	6.1	Validation plan
IEC 82304-1	8.4	Post-market communication on the health SW product
IEC 82304-1	8.5	Decommissioning and disposal of the health SW product

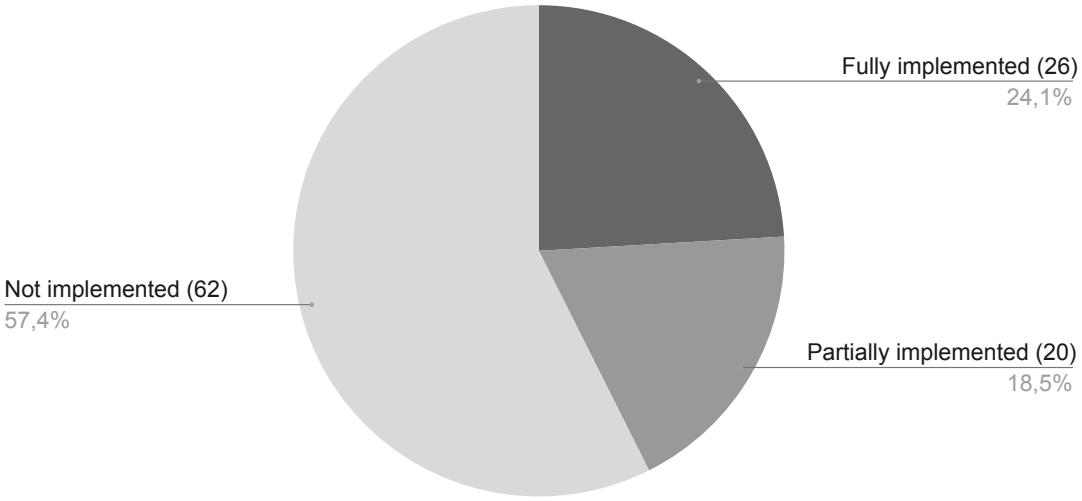

Figure 7 *Chart of the regulatory requirements. Out of 108 identified regulatory requirements, 26 can be fully implemented in a CI/CD pipeline, 20 can be partially implemented in a CI/CD pipeline, and 62 must be implemented outside of a CI/CD pipeline.*

5 Integration of the identified regulatory requirements into a CI/CD pipeline

There are both gaps and similarities between mindsets of regulators and software developers. Both groups desire high-quality software, but the approach is different. The DevOps culture tries to maximize customer value by delivering new features, as quickly as possible. On the other hand, the regulators value product safety and performance. In an ideal situation, the product achieves goals of both parties. However, modern software development and medical software development could be challenging to integrate. Pitkänen and others [Pitkänen et al., 2020] consider, that the winning medical device software manufacturers will apply regulations and standards efficiently into their processes. While it may not be completely apparent, the regulations and standards act as enablers for more safe, secure and better performing software. Process traceability and transparency are core principles in DevOps-culture, which is also an essential requirement for compliance with the medical device regulations.

In this Chapter, the regulatory requirements, identified in Chapter 4, are integrated into a CI/CD pipeline, introduced in Section 3.2. As a result, a regulated CI/CD Pipeline, illustrated in Figure 8, is established. The illustrated pipeline can be applied to any software, that falls into the IEC 62304's software safety classification Class C. However, any medical device software with less rigid manufacturing process can also be applied, as the Class C is the most rigid of the software safety classifications, and includes the requirements of the Class A and Class B.

There are specific infrastructure-related assumptions in the model. The pipeline utilizes a modern CI/CD pipeline toolkit, which enables creation of hardware and software environments directly from code. The whole infrastructure is specified in code, and can be built with scripts, that are stored in VCS, and accessed from the pipeline, to launch the script at demand. This is extended by customized tools, to implement activities required by regulatory compliance. It is assumed, that the manufacturer manages product-related user requirements, software requirements, risks, anomalies, and change requests. Moreover, it is assumed, that the resulting documentation is stored in such a way, that the result can be integrated with modern version control systems, and can be utilized in the pipeline for different purposes, such as generation of documentation. Additionally, the development process, nor the activities, are not included in the regulated CI/CD pipeline model. However, the agile development model that was introduced in Chapter 2, and illustrated in Figure 5, could be applied to the model as a substitute.

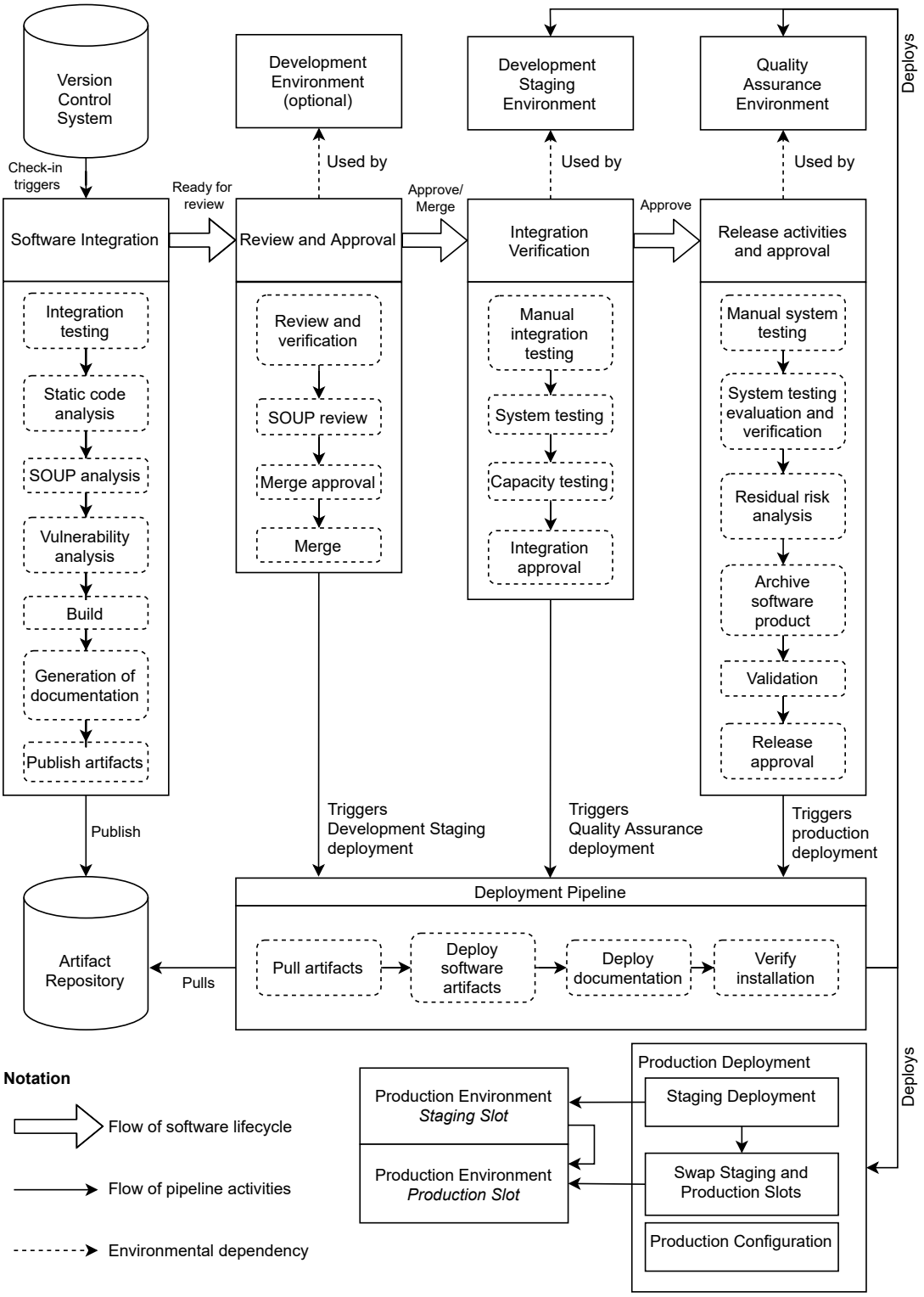

Figure 8 *Regulated CI/CD pipeline. The stages of the pipeline are illustrated from left to right. A new check-in into the VCS triggers the pipeline. The software change is reviewed and approved by human in the pipeline stages, which deploy the software into different environment, and finally into real world use.*

5.1 Stages of the pipeline

In this section, the stages of the regulated CI/CD pipeline are presented in more detail. The pipeline consists of five different stages, which are following; software integration, review and verification, deployment pipeline, integration verification and release activities and approval. A change made into the software must pass all the stages, before the change can be deployed into the real world use. Moreover, the software must also meet with the medical device regulations, rendering the activities in the pipeline mandatory.

5.1.1 Software integration

In the software integration stage, a Continuous Integration pipeline is triggered by a new check-in, pushed to the upstream of the version control system. The check-in contains changes to either software, documentation or associated items stored in the VCS. After the CI pipeline has been triggered, the code is tested against automated unit and integration tests (IEC 62304 clauses 5.6.3, 5.6.5, 5.6.7, 5.7.1). A comprehensive set of unit tests can be used to find any defects from the software during the development, as early as possible. Fixing bugs early in the development stage, has been found to reduce the software development costs [Vitharana, 2017].

In the next step, the static code analysis, the code is verified against formal rules and coding conventions (IEC 62304 clause 5.1.12). In the same stage, the SOUP components of the software may be checked, performing a SOUP analysis to find any components that need to be listed to fulfill the regulatory requirements (IEC 62304 clause 8.1.2). The results of the SOUP analysis are reviewed later in the pipeline. The IEC 62304 requires the manufacturer to divide the software into *software units* by the standard (IEC 62304 clause 5.4.1). The division of the software into the software units can be done programmatically, by using the augmented C4 architecture model [Stirbu et al., 2020]. The software components are annotated in the code package structure, and can be generated into diagram of software units in the pipeline. The architecture diagram is associated as the part of the product documentation, fulfilling the architectural documentation requirements (clauses 5.3.1, 5.3.2), software unit division requirement (clause 5.4.1) and documenting traceability requirement (clause 7.3.3) of IEC 62304.

After the static code analysis, a vulnerability analysis is performed against the software, to find any vulnerabilities from the software dependencies. If the software contains many SOUP components, it is possible, that the external components expose vulnerabilities in the software, which could be exploited to gain unprivileged access to the software. The vulnerability analysis can be performed with tools such as OWASP Dependency Check tool [OWASP, 2021].

The software artifacts are created after the automated testing and other verification activities have been performed. The software artifact is essentially a binary file of the software, which can be used later to install the software into different computational environments. It is recommended to build the software artifacts only once during the pipeline [Humble et al., 2011]. Moreover, the associated software documentation is generated in the pipeline. The software documentation is either pulled from an external source, or stored in the VCS, and generated from source files. This step generates the system configuration documentation (IEC 62304 clause 8.1.3), software system testing verification (IEC 62304 clause 5.7.4) and test documentation (IEC 62304 clause 9.8). Finally, the software and the associated documentation can be stored in a dedicated location, to be accessed later in the pipeline.

If any defects or problems are found during the software integration stage, the software developer is notified immediately to fix the problem and build will not proceed. However, if the build is successful and the feature is ready, it may move forward in the pipeline. Additionally, the software is not deployed into a dedicated computational software environment in this stage, but the changes can be tested in a development environment, if needed.

5.1.2 Review and verification

The regulatory requirements define a need for verification of various tasks and activities, which can be considered mostly manual task in the pipeline, as most of the verification activities require human-made inspections and decision-making.

When the software change is ready to be reviewed, the pipeline has created automatically a review request for the change. For instance, a review request could be a pull request [GitHub, 2021] or a merge request [GitLab, 2021], depending of the platform. A review request is a way for the software developer to acknowledge other parties, such as reviewers and testers, that their feature has been finished and is ready for further actions.

In the review, the software change is reviewed by one or more competent developers, or other qualified personnel. A code review is performed to verify that the new code is implemented correctly. The software units are verified (IEC 62304 clause 5.5.5) against the software unit acceptance criteria (IEC 62304 clause 5.5.3), defined in the software unit verification process (IEC 62304 clause 5.5.2). Moreover, the generated product documentation is verified to be up-to-date, and to match the implementation of the software, such as detailed design documents (IEC 62304 clause 5.4.4) and architecture verification (IEC 62304 clause 5.3.6). Furthermore, the result of the SOUP analysis, performed in the CI pipeline, is also inspected by a human. Any unlisted SOUP items are documented in this stage, at the latest. It

is also required to document functional, performance, hardware and system requirements for every SOUP item in the software (IEC 62304 clause 5.3.3, 5.3.4), and to evaluate the published SOUP lists (IEC 62304 clause 7.1.3).

After the software change is completed, verified and reviewed, a merge approval is performed and the change may be merged into the mainline of the product in the VCS. The mainline of the product is merged into the new code change, before the review is performed. However, if any merge conflicts happen, the developer is notified to fix the problems and the merge will not proceed.

5.1.3 Deployment pipeline

A check-in to the mainline of the version control system triggers the deployment pipeline, which happens after a new change has been merged. When the deployment is triggered, the pipeline pulls the artifacts, generated by the CI pipeline. The software artifacts are installed into the target environment, and the associated documentation is made available. Finally, the installation is verified by a running smoke tests into the deployed environment. With smoke tests, it is possible to quickly verify that the installation was successful and the software is working as intended [Memon et al., 2005]. Additionally, the installation can be verified with secure hash check [Hamilton, 1995], to prevent any unauthorized binaries from being installed.

Continuous Deployment, when defined as a process, which deploys and makes the software available to the end-users, is problematic in the context of medical device software. Many of the regulatory activities, introduced by EU Regulatory Framework, require a human-made decision, before the software can be placed in the possession of the end-user. For instance, Design Change Approval could act as a barrier to Continuous Deployment [Granlund et al., 2020]. Any substantial changes, such as new features that require clinical or performance evaluation must be approved by a Notified Body before deployment. Only small scale changes such as bugs with negligible risk could be applied into this model. The risk level of the change is an important factor. A high-risk change in the system or a change in critical system component requires more attention and careful planning before production deployment. As a result, fully automated deployment without human approval is not possible. Leading to the fact, there must be a human-decision factor deciding when the software is deployed, although it could only be a single confirmation step in the process. Thus, it is difficult to incorporate Continuous Deployment pipeline into the delivery process of the medical device software. It can be even argued if the customers want DevOps-style frequent updates [Smeds et al., 2015]. The nature of the medical device software system could be critical and any possible maintenance break must be planned in advance to prevent patient risks. The manufacturer could be ready to deploy the software into the end-user

environment anytime, but the customer can't approve the update, because of the involved risks, if any problems occur in the production.

In the regulated CI/CD pipeline model, the stages require a human-made approval, before any deployment is made. Any regulatory requirements that directly affect the deployment pipeline were not identified during the analysis of the standards. However, the deployment pipeline is a part of the pipeline, and the software can not be reliably installed without it. The same deployment pipeline handles every deployment of the software into different environments, including the final end user's environment. The deployment process is repeatable and deterministic by nature. Furthermore, installing the software with the same scripts multiple times before the final installation into the production environment, can make the installation process more likely to succeed [Humble et al., 2011].

5.1.4 Integration verification

After the review request has been approved, and the new version of the software has been deployed into the development staging environment, the software change can further verified. In an ideal situation, the development staging environment is a copy of the real world customer environment, as close as possible. In the development staging environment, the software may be tested in a full context. Ultimately, every test case can not be automated, thus leading to need for manual testing in real environment. Any remaining manual integration tests are performed in the staging environment, which fulfills the requirement to verify software integration (IEC 62304 clauses 5.6.2, 5.6.3, 5.6.4, 5.6.5, 5.6.6, 5.6.7).

Moreover, automated system tests are also performed against the software in the staging environment (IEC 62304 clauses 5.7.1, 5.7.3, 5.7.5, 7.3.3, 8.2.3, 9.8). The system testing ensures that the system's functionality remains as it is required, as well as, to verify the functionality of the new features or other changes in the system. The system must meet it's intended requirements, and perform as designed, to be able to be released for the end-users. The system testing process is repeated for every iteration of the software, delivered to the real use. However, performing the full system testing manually can be burdensome. Hence, a high level of automation for the system testing tasks could reduce the burden, caused by the comprehensive testing. However, even while automated, running the system test automation could take a considerable amount of time, which is the reason it is performed in the staging environment, and not the integration pipeline.

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.

Integration approval can be made, after all automated and manual tests have passed successfully, completing the software integration (IEC 62304 clause 5.6.1). A deployment to the quality assurance environment is triggered by the approval. The quality assurance environment is the final destination for software testing and verification, before the software is deployed into the real world use.

5.1.5 Release activities and approval

Before the software can be released to the use, it must be system tested according to the system testing plan. Furthermore, the system testing must be evaluated and verified, and the software must be validated to fulfill it's purpose.

First of all, the system testing must be completed by running all test scenarios that could not be automated (IEC 62304 clauses 5.6.6, 5.7.5, 7.3.3, 9.8). This also includes the regression testing for the system, which essentially requires the manufacturer to produce evidence, that the changes in the software did not cause any unwanted side-effects. The testing is done in a separate QA environment, which contains an unreleased version of the software, called a release candidate. However, depending of the test automation coverage and the magnitude of the overall system context, this process could require significant amount of resources. As a side note, even with comprehensive test coverage, exploratory testing is still recommended, to find any defects that could be undetected by test cases [Shah et al., 2014].

The system testing activities must be evaluated and verified by formal practices after completion of system testing. In practice, the system test results are evaluated and verified as stated in IEC 62304 (clauses 5.7.4, 5.8.1) to be performed properly. Furthermore, anomalies found from the product must be documented and evaluated (clause 5.8.2, 5.8.3).

In the next step, the risk control measures are to be verified (IEC 62304 clause 7.3.1) and the residual risk level of the medical device product must be either reduced to or remain at an acceptable level before the software is ready for release. Any unmitigated risks, that could possible cause risk to person's health, act as barriers for the final release.

Finally, before the final release, the manufacturer must ensure that all activities mentioned in the software development plan are completed (IEC 62304 clause 5.8.6). The software and documentation artifacts created by the CI are labeled with a release version tag (IEC 62304 clause 5.8.4), and a Unique Device Identifier, required by IEC 82304-1 (IEC 82304-1 clause 7.2). The software artifacts and the associated documentation are permanently archived (IEC 62304 clause 5.8.7) for later access. At this point, the software is ready for release in the technical perspective, but regulatory compliance requires the manufacturer to perform a validation according

to the validation plan (IEC 82304-1 clauses 6.2, 6.3, 8.3). Essentially, validation is the process for obtaining reliable evidence, that proves in formal way, the software fulfills its intended purpose as defined.

When the software is technically ready for release, and the regulatory compliance activities have been performed properly, the manufacturer can perform a formal release approval, and the software can be deployed to the customer environment. However, deployment is not equivalent to releasing the software into real world use in this scenario. The model utilizes Blue-green deployment model, which enables the manufacturer to release the system on demand to the customer's use. Essentially, the software can be released by switching the current customer environment with the secondary environment with a new version deployed, making the software available to the end users instantly.

5.1.6 Implicitly implemented regulatory requirements

Not all of the regulatory requirements are activities/tasks, that can be performed directly in the pipeline. Instead, the pipeline implements some of the regulatory requirements implicitly as in-built features, such as documenting how the release was made, the repeatability of the release, and re-releasing the modified system (IEC 62304 clauses 5.8.5, 5.8.8, 6.3.2).

The pipeline itself stores information of every built software version and software artifact (clause 5.8.5). Additionally, the scripts and configurations used to build the software, are stored in the VCS. In fact, every build should be reproducible from the version control [Humble et al., 2011]. By having the possibility to return to an old version at any point, the releases are always repeatable (clause 5.8.8). Moreover, every version of the system, that ends up in to real world use, must be deployed through the CI/CD pipeline (clause 6.3.2).

The regulated CI/CD pipeline enforces the medical device manufacturer to incorporate quality aspects into the software development processes, while performing the most burdensome tasks and activities automatically. However, the amount of the process automation is crucial, because the requirements for regulatory compliance are burdensome and strict. Finally, not all of the activities can be fully automated, while some of the activities require a human approval to proceed.

5.2 Regulatory compliance by design

In this section, practices which could help with designing a system with regulatory compliance in mind, are discussed. A medical device software can meet the regulatory requirements with much less effort, when the system is designed to comply with the regulations from the beginning of the software life cycle. If the support

for regulatory compliance is ignored in the beginning of the development phase, it requires lots of effort to change the system to support the regulatory requirements.

5.2.1 Version control and traceability

Version control could be seen as enabler for CI/CD pipelines [Humble et al., 2011]. To take this viewpoint slightly further, a modern VCS, such as Git, could be seen as a enabler for reliable and deterministic way for regulatory compliance, in the technical perspective. Ultimately, if every item of the technical product and the associated documentation are stored in VCS, and a specific version of the product can be retrieved at any point from the history, the traceability requirements of the regulatory compliance are much easier to implement. However, it might not be sensible to store every piece of documentation into VCS, such as verification reports, which are generated every time, when the CI/CD pipeline runs automated verification activities. However, the manufacturer should be able to generate the documents from the specific revision of the product, at any time. Therefore, if the whole software product, including the system infrastructure and the product documentation are reproducible from the version control system, problems related to the comprehensive documentation and traceability requirements, introduced by the medical device standards, could be solved.

5.2.2 Behavior-Driven Development

Behaviour-Driven Development (BDD) is a software development method [Smart, 2014], which encourages collaboration between requirements analysts, software developers and software quality assurance specialists. In BDD, the business goals of the software are analysed, turned into software requirements, and further defined into acceptance criteria/tests for the software. As a result, when the software has been changed, the verification can be done against the resulting acceptance criteria. Basic principles of BDD include defining tests first, making the tests fail, then implementing the unit, and finally verifying the unit with the passing tests. This could support the manufacture of medical devices, as the manufacturing process is reminiscent with the behavior-driven development process.

Specifically, the IEC 82304-1 requires the medical device manufacturer to establish the use requirements for a health software product, which are used to verify and validate the product, before the software can enter the market. The use requirements of the software could be used as an input for the BDD process, which could provide practical evidence, that the software implementation is verified. In both, BDD and IEC 82034-1 activities, the requirements of the software are first collected

and analysed. The software requirements are used as an input for the software implementation and testing/verification. Moreover, in BDD, the information related to the test cases is documented in the process. Furthermore, the test cases could be automated in the process, as far as possible, to improve the test coverage, which can be performed in a CI/CD pipeline.

The application of BDD into the manufacturing of medical device software could help the medical device software manufacturers to implement regulatory requirements into their software life cycle processes with an industry best-practice solution [Giorgi et al., 2019]. Additionally, BDD could assist the software manufacturer with the composition of test documentation, and to provide better evidence for the software verification, while establishing more comprehensive test automation. As a result, better performing and more reliable software could be manufactured.

5.2.3 Segregation of medical device software components

Sometimes, medical device software may contain some components that are not considered as medical devices. Nevertheless, medical device components must meet with the medical device regulations, while the non-medical device components are not subject to the regulatory requirements [Pitkänen et al., 2020]. The medical device components are identified by their intended purpose, and are classified by the software safety classification, introduced by IEC 62304 and the applicable product classification from MDR/IVDR.

The software architecture must be implemented in a way, that enables risk control for safety critical components. The risk control could be possible by applying a modular software architecture, to isolate the medical device components from the software. Moreover, the manufacturer is required to explain how risk control is ensured, and to rationalize the effectiveness of the segregation. Therefore, by utilizing software component segregation, it could be possible to use a less rigid development process on some parts of the system, while maintaining the more rigid process on higher risk level components. This could help to reduce the laborious activities, introduced by the regulatory requirements. However, if the regulatory landscape around the software product changes for some reason, and the software safety classification is replaced with more rigid class, it could be beneficial to have the software developed with the most rigid development process from the beginning, as producing the missing product documentation could be burdensome afterwards. Additionally, in some cases, the additional efforts in the more rigid development process could be useful. For instance, in large software systems, the complexity could grow over time, and the additional documentation could help with the maintenance of the system.

It could be possible to divide the system into appropriate pieces, and to segregate the high-risk medical components by utilizing Service-Oriented Architecture (SOA)

[Sprott et al., 2004], as it allows establishment of independent services for different business functionalities. The idea behind SOA, is to segregate the functionalities of the software into independent software components, which could be utilized to adjust the software into the regulated environment by incorporating software safety classifications, software units, or other regulatory requirements into the software architecture.

5.3 Software risk management

Ultimately, the manufacture of medical devices is culminated into risk management and mitigation. Therefore, the purpose of the rigid process requirements is to reduce the risks related to a person's health that could be caused by a medical device. The risk management follows the whole life cycle process, and the residual risk should always be mitigated into an acceptable level, before the medical device can be deployed into use.

By applying reliable and deterministic practices to software verification and delivery, the regulated CI/CD pipeline could reduce software risks by mitigating the factor of human-made error from the process. In practice, by removing manual work and activities that may depend on human memory from the verification and delivery process, the risk of human-made error could be reduced significantly [Humble et al., 2011]. The process is implicitly documents itself, as every activity on the verification and delivery process is scripted and automated.

5.4 Software problem resolution process

The regulated CI/CD pipeline model does not include *software problem resolution process*. Nevertheless, the pipeline could feed input to the problem resolution process, such as bugs or problem reports. As a result, a fixed problem could appear as an input for the CI/CD pipeline, as it could change the software. Software problems resolution process applies to the whole software life cycle process, and it must be possible to fix a problem at any point of the process. However, the form of the problem report can be different depending on the process part the problem was identified.

5.5 Software maintenance

The software life cycle does not end after production deployment, and the release of the product into the real-world use. The manufacturer must establish a software maintenance process, which is used to modify an existing software [IEC, 2015]. The software maintenance plan is utilized after the initial development has ended, and the software needs upgrades or any problems are detected. The manufacturer is

permitted to use a lighter process for maintenance, to fix problem reports related to a released product, or to implement rapid changes in response to urgent problems [IEC, 2015]. The process might correspond to the development process with some modifications. Finally, the manufacturer is responsible for the product and it's safety, until it is removed from use and disposed properly, which is when the responsibility ends.

6 Conclusions

The purpose of this thesis was to find ways to integrate regulatory activities, introduced by EU regulations, such as MDR and IVDR into CI/CD pipelines. The most practical and convenient way to implement the regulatory requirements, is to follow medical device standards. IEC 62304 and IEC 82304-1 were chosen as the main standards for this study because they introduce requirements for software life cycle and product safety, which mostly affect the implementation of CI/CD pipeline. DevOps practices were studied by studying the literature and adopting a reference model for the study. Total of 108 requirements were identified from the standards IEC 62304 and IEC 82304-1, of which 26 were identified as fully implementable in a CI/CD pipeline, 20 were identified as partially implementable requirements in a CI/CD pipeline, and 62 requirements were identified to be excluded from a CI/CD pipeline. As a result of this research, a model for regulated CI/CD pipeline was created, and the implemented regulatory requirements were rationalized.

The objective of this thesis was to produce practical ideas for the medical device software industry to establish a software development process that utilizes DevOps, while maintaining compliance with the medical device regulations. In the real world, software development can sometimes resemble a chaos, but even then it is important to assure the safety and performance of the end product. A change in the software could be iterated multiple times, which could cause a great amount of additional verification work and refactoring of documentation.

The answer to *Research Question 1. Is it possible to integrate CI/CD pipeline into the medical device software development process?* Yes. As long as the required activities and tasks can be automated, there is no problem in the utilization of Continuous Integration or Continuous Delivery. It is highly recommended to utilize Continuous Integration. It is also possible to utilize Continuous Delivery, while maintaining conformity with the requirements from IEC 62304, keeping the software in release-ready state all the time. However, any change that has not completed the development process, should not be deployed to real world use, and the final deployment should be gated by a human-decision.

The answer to the *Research Question 2. Is it possible to utilize Continuous Deployment in medical device software delivery process?* is No. Even if the system architecture supports Continuous Deployment, risk control measures must be applied to the deployment process, as only features that do not impact clinical and/or performance evaluation can be deployed without a permission. The pipeline can otherwise be utilized, but there must be a human-made decision incorporated, which limits the applicability of Continuous Deployment. However, the Continuous

Deployment process could be applied to other environments outside of customer production environment, because there is no patient risk involved and only production environments should be used for patient treatment. A testing environment identical to the production environment could exist for testing purposes, and the changed software could be deployed continuously into the testing environment.

The answer to the *Research Question 3. What advantages could a CI/CD pipeline introduce to a medical device manufacturer?* The automated pipeline reduces the amount of manual work needed for the verification, release and installation activities. The deployment process is normalized, reducing the possible user errors from the software deployment [Humble et al., 2011]. The software changes could be deployed into a testing environment rapidly, after the development and verification of the new feature. The customer organisation could try and experiment the changes in the testing environment, and provide feedback based on the testing, creating a feedback loop for the manufacturer. The manufacturer receives feedback from the customer, and any possible flaws or defects in the software can be detected, before the changed software product is put into real world use, reducing the possibility of discovering unknown risks in the production. Downside to this solution could be the cost-efficiency, because it requires more resources, such as server capability, to maintain. Upsides would be the possibility to inspect the technical integrity of the product even during development phases, and to take an instant action when a problem/defect is identified. However, using a testing environment on real patient treatment is prohibited, and the risk of doing that should be mitigated before any testing or validation activities can happen.

6.1 Limitations and applicability of the research

The regulated CI/CD pipeline does not include every regulatory requirement that could apply to a medical device software. The set of applicable regulatory requirements can differ, depending on the intended purpose and the type of the software system. However, in general, NB assessment, CE marking, clinical and performance evaluation, most of the software risk management and software configuration management, software usability validation, and some other medical device standards were excluded from the scope intentionally. Some regulatory requirements, left outside of the scope of this study, such as the Notified Body assessment and usability validation could cause some delays for the actual release to the end users. More work is needed to implement a fully comprehensive medical device software development model, to include every generally applicable regulatory requirement.

The applicability of the proposed pipeline model itself depends on many factors, such as the architecture of the system, size and scope of the project, coverage of test automation, the development tools and technologies used. The model is more

suitable for new projects, as legacy software may contain issues with many of the aforementioned factors. In the beginning of new project, it is possible to choose the correct tools for the implementation, and establish the development and delivery processes from the scratch, whereas for legacy projects it might require lots of resources to alter the software and the surrounding infrastructure. Additionally, it could be difficult to utilize DevOps practices on a large software systems with a monolithic architecture as the time needed for build and execution of automated tests might take a significant time. A comprehensive test automation coverage is difficult to attain, if it has been neglected in the early stages of the software project, as the catching up with the development could take considerable amount of work. Moreover, without the comprehensive test automation, the manufacturer must resort to manual testing, which can become a burdensome, slow and expensive task, when the system reaches maturity. Furthermore, it is required to perform verification of the product on demand, when it is needed. Test automation does not necessarily find new defects in the product, but it can be utilized for regression testing, to ensure that existent features do not break, when the software is changed. Test automation reduces the amount of manual testing needed by a substantial amount, making the software changes easier to implement.

6.2 Discussion and further research

The research found some key aspects and proposed a model to implement a regulated CI/CD pipeline. To further validate the idea behind the proposed model, a reference implementation is needed, followed by a case study, which validates the model in practice. The model could also be further developed to include regulatory requirements/activities that are currently scoped out, such as development planning, design and implementation process of the software, interaction with the NB, configuration management, risk management, software usability validation and quality management systems. As a suggestion, the agile development model from AAMI TIR:45 [AAMI, 2012], illustrated in Figure 5, could be integrated into the proposed regulated CI/CD pipeline model, to implement more of the regulatory requirements related to the planning, design and development activities.

Additionally, the medical device software industry could use standard tooling set and shared industry best practices to implement the automation of the the activities, required by the regulations, more efficiently. The regulatory requirements introduce activities, such as the comprehensive and traceable documentation requirements, which are rare outside of regulated industry sectors, creating a specific need for additional tooling to implement efficiency into the manufacturing processes.

From the business perspective, the medical device software manufacturers, that

successfully adopt and integrate the regulatory requirements into their manufacturing processes, could gain significant business advantage in the industry. It could be difficult for smaller companies, such as start-ups, to integrate the regulatory requirements into the manufacturing process, while developing new products efficiently. This could lead to a difficult situation for smaller software companies, where it is hard or impossible to compete against large software vendors.

Finally, an industry paper, based on ideas presented in this thesis, was written, and submitted into PROFES 2021 Conference [Toivakka et al., 2021].

7 Acknowledgements

The author would like to thank Mylab Oy, staff of Tampere University, Business Finland and the members of AHMED (Agile and Holistic MEdical software Development) consortium for supporting this work.

References

AAMI (2012). *AAMI TIR45:2012 Technical Information Report Guidance on the use of AGILE practices in the development of medical device software*.

Amrit, Chintan and Yoni Meijberg (2018). "Effectiveness of Test-Driven Development and Continuous Integration: A Case Study". *IT Professional* 20.1, pp. 27–35.

Duvall, Paul M., Andrew Glover, and Steve Matyas (2007). *Continuous integration : improving software quality and reducing risk*. 1st ed. Addison Wesley.

European Commission (1990). *COUNCIL DIRECTIVE of 20 June 1990 on the approximation of the laws of the Member States relating to active implantable medical devices*.

European Commission (1993). *COUNCIL DIRECTIVE 93/42/EEC of 14 June 1993 concerning medical devices*.

European Commission (1998). *DIRECTIVE 98/79/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 October 1998 on in vitro diagnostic medical devices*.

European Commission (2016). *The 'Blue Guide' on the implementation of EU products rules*. URL: https://ec.europa.eu/growth/content/%E2%80%98blue-guide%E2%80%99-implementation-eu-product-rules_en (visited on 09/09/2021).

European Commission (2017a). *REGULATION (EU) 2017/745 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 5 April 2017 on medical devices, amending Directive 2001/83/EC, Regulation (EC) No 178/2002 and Regulation (EC) No 1223/2009 and repealing Council Directives 90/385/EEC and 93/42/EEC*. URL: http://d-nb.info/1152290274/04.

European Commission (2017b). *REGULATION (EU) 2017/746 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 5 April 2017 on in vitro diagnostic medical devices and repealing Directive 98/79/EC and Commission Decision 2010/227/EU*.

European Commission (2021a). *CE marking - obtaining the certificate, EU requirements*. URL: https://europa.eu/youreurope/business/product-requirements/labels-markings/ce-marking/index_en.htm (visited on 09/09/2021).

European Commission (2021b). *EUDAMED Database*. URL: https://ec.europa.eu/tools/eudamed/#/screen/home (visited on 09/09/2021).

European Commission (2021c). *Standardisation request to the European Committee for Standardization and the European Committee for Electrotechnical Standardization as regards medical devices in support of Regulation (EU) 2017/745 of the*

European Parliament and of the Council and in vitro diagnostic medical devices in support of Regulation (EU) 2017/746 of the European Parliament and of the Council.

European Commission (2021d). *Types of EU law*. URL: https://ec.europa.eu/info/law/law-making-process/types-eu-law_en (visited on 09/09/2021).

Fimea (2020). *Lääkinnällisten laitteiden asetuksen (MDR) siirtymäaika siirtyy vuodella eteenpäin Covid-19:n vuoksi*. URL: http://www.fimea.fi/-/laakinnallisten-laitteiden-asetuksen-mdr-siirtymaaika-siirtyy-vuodella-eteenpain-covid-19-n-vuoksi (visited on 09/09/2021).

Giorgi, Fabio and Frances Paulisch (2019). *Transition Towards Continuous Delivery in the Healthcare Domain*. DOI: 10.1109/ICSE-SEIP.2019.00035.

GitHub (2021). *About pull requests*. URL: https://docs.github.com/en/github/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests (visited on 09/09/2021).

GitLab (2021). *Merge requests*. URL: https://docs.gitlab.com/ee/user/project/merge_requests/ (visited on 09/09/2021).

Gousios, Georgios, Margaret-Anne Storey, and Alberto Bacchelli (2016). "Work practices and challenges in pull-based development". In: ICSE '16. ACM, pp. 285–296. URL: http://dl.acm.org/citation.cfm?id=2884826.

Granlund, Tuomas, Tommi Mikkonen, and Vlad Stirbu (2020). "On Medical Device Software CE Compliance and Conformity Assessment". In: 2020 IEEE International Conference on Software Architecture Companion (ICSA-C). DOI: 10.1109/ICSA-C50368.2020.00040. URL: https://ieeexplore.ieee.org/document/9095660.

Granlund, Tuomas, Vlad Stirbu, and Tommi Mikkonen (2021). *The Need for Calm Compliance*. Unpublished manuscript.

Hamilton, Booz Allen (1995). *SECURE HASH STANDARD*.

Humble, Jez and David Farley (2011). *Continuous delivery*. Upper Saddle River (N.J.): Addison-Wesley.

IEC (2015a). *MEDICAL DEVICE SOFTWARE. SOFTWARE LIFE-CYCLE PROCESSES (IEC 62304:2006/A1:2015)*.

IEC (2015b). *MEDICAL DEVICES - PART 1: APPLICATION OF USABILITY ENGINEERING TO MEDICAL DEVICES (IEC 62366-1:2015/A1:2020)*.

IEC (2016). *IEC 82304-1:2016. HEALTH SOFTWARE — PART 1: GENERAL REQUIREMENTS FOR PRODUCT SAFETY*.

ISO (2007). *ISO 14971:2007. APPLICATION OF RISK MANAGEMENT TO MEDICAL DEVICES*.

ISO (2018). *ISO 13485:2003/2016—Medical Devices—Quality Management Systems—Requirements for Regulatory Purposes*.

Khan, Muhammad Owais (2020). "Fast Delivery, Continuously Build, Testing and Deployment with DevOps Pipeline Techniques on Cloud". *Indian Journal of Science and Technology* 13.5, pp. 552–575.

Laster, Brent (2020). *Continuous Integration vs. Continuous Delivery vs. Continuous Deployment, 2nd Edition*. 1st ed. O'Reilly Media, Inc.

Laukkarinen, Teemu, Kati Kuusinen, and Tommi Mikkonen (2018). "Regulated software meets DevOps". *Information and Software Technology* 97, pp. 176–178. DOI: 10.1016/j.infsof.2018.01.011. URL: https://search.datacite.org/works/10.1016/j.infsof.2018.01.011.

Medical Device Coordination Group (2019). *Guidance on Qualification and Classification of Software in Regulation (EU) 2017/745 – MDR and Regulation (EU) 2017/746 – IVDR*.

Memon, Atif, Adithya Nagarajan, and Qing Xie (2005). "Automating regression testing for evolving GUI software". *Journal of Software Maintenance and Evolution: Research and Practice* 12.1, pp. 27–64.

Morris, Kief, Brian Anderson, Jasmine Kwityn, Karen Montgomery, Rong Tang, and Rebecca Demarest (2016). *Infrastructure as code : managing servers in the cloud*. Sebastopol, California: O'Reilly.

OWASP (2021). *OWASP Dependency-Check Project*. URL: https://owasp.org/www-project-dependency-check/ (visited on 08/30/2021).

Pitkänen, Heikki, Leena Raunio, Santavaara Ilona, and Tom Ståhlberg (2020). *A Guide to Market*. URL: https://www.leanentries.com/wp-content/uploads/european-medical-device-regulations-mdr-ivdr-a-guide-to-market.pdf (visited on 09/09/2021).

Rossberg, Joachim (2014). *Beginning Application Lifecycle Management*. 1st ed. Apress. ISBN: 1-4302-5813-6. DOI: 10.1007/978-1-4302-5813-1.

Shah, Syed Muhammad Ali, Cigdem Gencel, Usman Sattar Alvi, and Kai Petersen (2014). "Towards a hybrid testing process unifying exploratory testing and scripted testing". *Journal of Software : Evolution and Process* 26.2, pp. 220–250. DOI: 10.1002/smr.1621.

Smart, John (2014). *BDD in Action: Behavior-Driven Development for the Whole Software Lifecycle*.

Smeds, Jens, Kristian Nybom, and Ivan Porres (2015). *DevOps: A Definition and Perceived Adoption Impediments*. Cham: Springer International Publishing. DOI: 10.1007/978-3-319-18612-2_14. URL: https://search.datacite.org/works/10.1007/978-3-319-18612-2_14.

Spinellis, Diomidis (2017). *State-of-the-Art Software Testing*.

Sprott, David and Lawrence Wilkes (2004). *Understanding Service-Oriented Architecture*.

Ståhlberg, Tom (2015). *Suomi ja EU fokuksessa*.

Stellman, Andrew and Jennifer Greene (2014). *Learning Agile: Understanding Scrum, XP, Lean, and Kanban*. O'Reilly Media, Incorporated.

Stirbu, Vlad and Tommi Mikkonen (2020). *CompliancePal: A Tool for Supporting Practical Agile and Regulatory-Compliant Development of Medical Software*. DOI: 10.1109/ICSA-C50368.2020.00035.

Toivakka, Henrik, Tuomas Granlund, Zheying Zhang, and Timo Poranen (2021). "Towards RegOps: A DevOps Pipeline for Medical Device Software". Unpublished manuscript. 16 pages. Accepted for publication in International Conference on Product-Focused Software Process Improvement (PROFES 2021).

Vitharana, Padmal (2017). "Defect propagation at the project-level: results and a post-hoc analysis on inspection efficiency". *Empirical Software Engineering* 22.1, pp. 57–79.

Wettinger, Johannes, Uwe Breitenbücher, and Frank Leymann (2014). "DevOpSlang – Bridging the Gap between Development and Operations". In: Lecture Notes in Computer Science. Springer Berlin Heidelberg. DOI: 10.1007/978-3-662-44879-3_8. URL: http://link.springer.com/10.1007/978-3-662-44879-3_8.

Wichmann, B. A., A. A. Canning, D. W. R. Marsh, D. L. Clutterbuck, L. A. Winsborrow, and N. J. Ward (1995). "Industrial perspective on static analysis". *Software Engineering Journal* 10.2, p. 69.