CHALMERS

Regulations for the Development of Medical Device Software

Master of Science Thesis

ANDREAS MAGNUSSON

Department of Signal and Systems
Division of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden, 2011
Report No. EX004/2012

Regulations for the Development of Medical Device Software

ANDREAS MAGNUSSON

Technical report no. EX004/2012
Department of Signal and Systems
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31-7721000

Acknowledgments

I would like to thank Integrum AB for giving me this opportunity to assist them in their expansion into medical software development and for their patience with me throughout this process. I have learned a lot through this project concerning medical software development and the regulatory requirements that a medical device manufacturing and developing company faces.

I would specifically want to thank my supervisor, Max. J. Ortiz C., for his input, support and assistance during this project.

I would also want to thank my examiner, Professor Bo Håkansson (Department of Signal and Systems at Chalmers University of Technology), for having me as his student and for his assistance and input during the project.

Lastly, I would like to thank my family and friends for their continued support during the project.

Abstract

Today, most medical devices contain software to some extent. As the use of software in medical devices has increased, a need to govern the development of the software used in these devices has been identified. In order to ensure that all medical devices that are released does not present any unnecessary risk to the end user, regulatory requirements has been created that govern how these medical devices should be developed.

When developing software that is to be used as part of a medical device, or software that is a medical device on its own, there are specific medical software regulatory requirements that need to be considered.

This thesis is about the adaptation of such regulatory requirements into a software development procedure that is to be implemented into the quality management system at Integrum AB.

The result of the thesis is the development of a set of software development procedures in order to assure compliance with the regulatory requirements of *IEC 62304 Medical device software - Software life cycle processes.*

Abbreviations

CDRH Center for Device and Radiological Health

CFR Code of Federal Regulations

CVS Concurrent Versions System

EC European Commission

FDA Food and Drug Administration

IDE Integrated Development Environment

IEC International Electrotechnical Commission

ISO International Organization for Standardization

MDD Medical Device Directive

NCAL-Soft Natural Control of Artificial Limbs Software

OHMG Osseointegrated Human-Machine Gateway

OPRA Osseointegrated Prostheses for the Rehabilitation of Amputees

OPRA-NCAL Osseointegrated Prostheses for the Rehabilitation of Amputees with Natural Control of Artificial Limbs

PMA Premarket Approval

SOP Standard Operating Procedure

SDE Software Development Environment

SVN Apache Subversive

QMS Quality Management System

Contents

Chapter 1

Introduction

Development of software for medical devices are controlled through international regulatory requirements defined by regulatory standards, such as International Organization for Standardization (ISO), or International Electrotechnical Commission (IEC).

As with other devices intended for medical applications, regulatory requirements for medical software are centered around risk analysis. This is to ensure that the software does not present any risk of harming the user(s). It is also to ensure that identifiable risks are properly controlled to minimize the potential harm, in the event of an accident.

1.1 Purpose

The purpose of this masters thesis is to establish a medical software development procedure that complies with current medical device software regulatory requirements and implement this procedure into the quality management system at Integrum AB.

1.2 Scope

The following activities make up the basis for this thesis

- A study of medical software development activities
- A study of commonly used software development models in order to properly implement development activities into a quality management system
- Determination of medical software safety classification based on a preliminary hazard analysis
- Implementation of regulatory requirements into an existing quality management system through a medical software development procedure

1.3 Integrum AB

Integrum AB is a Swedish company that develops implant systems for bone anchored amputation prostheses and was founded by Dr. Rickard Brånemark in 1998 at Sahlgrenska Science Park in Gothenburg, Sweden. The implant system, developed by Integrum AB, consists of a titanium device that is surgically inserted into the bone, protruding through the skin of the amputation stump. A prosthesis can easily be attached to the anchorage by the patient, giving no direct contact with the skin. This solution improves functionality and comfort to the patient compared to the traditionally used socket prostheses. It implies reduced problems with sores, pain and pressure, as well as easier attachment and detachment, improved stability and better walking ability. Integrum's implant system is called OPRA (Osseointegrated Prostheses for the Rehabilitation of Amputees). The ability of titanium to integrate with bone is called osseointegration. In this process, bone tissue is formed around the implant causing the bone tissue and surface of the implant to anchor together. The concept of osseointegration was presented in 1977 by Per-Ingvar Brånemark, who has developed the technique primarily for prosthetic replacement of teeth, but also for orthopedic applications.

Chapter 2

Medical Device Regulations Background

All devices that are used on the market have gone through some level of testing, depending on the type of device, to verify that the device is safe to use. For medical devices, these tests are stricter as a medical device can be in direct contact with a patient or be located inside a patient, such as a pacemaker. Depending on the type of medical device, there are different regulations which govern what tests are needed to be conducted and the level of documentation needed to prove that the device is safe to use. The regulations also govern that the device's intended use is such that it means an improvement of life quality in order to validate the potential risks inherited with the device.

2.1 Device Regulations in the US

Within the United States (US), the governing agency for medical devices is Food and Drug Administration (FDA) within the Department of Health and Human Services. Within the FDA, the Center for Devices and Radiological Health (CDRH) handles the regulatory concerns for companies who manufacture, repackage, relabel and/or import medical devices which are to be sold within the US. Any regulation for medical devices is published in the Federal Register under the Code of Federal Regulations Title 21 §820.

The FDA has created three regulatory classes which are based on the level of control necessary to assure the safety and effectiveness of the device, which are Class I, Class II or Class III, with Class III being for medical device with the highest risk. A medical device classified as Class I or Class II will require a premarket submission or 510(k) whilst a Class III medical device is required to get a premarket approval unless special circumstances are given, for which only a 510(k) is needed [1]. A 510(k) is a premarket submission which demonstrate that the medical is at least as safe and effective as a medical device already on the market and that is not subject to a Premarket Approval (PMA) [2].

When needing to do a 510(k), the medical device manufacturer is required to submit documentation to the FDA. The submitted documentation must prove that the device is at least as safe and effective as a similar device which is already on the market [2].

2.2 Device Regulations in the EU

In the European Union (EU) the governing organ for legislation's is the European Commission (EC). The EC publishes directives which serves as the regulatory framework, for medical devices this directive is simply called the Medical Device Directive (MDD). The MDD divides medical devices into either Class I, Class IIa, Class IIb or Class III, with Class III being for medical devices with the highest risk. In addition, there are a number of notified bodies which acts as a third party controller for medical devices. These notified bodies ensure that the medical devices that are released on the market complies with the MDD as well as other applicable directives through quality system inspections, certification and classification assessments. The notified bodies also review the technical documentation for the medical devices. It is also the notified bodies that issue the CE-marking, which is required in order to get a product onto the European market [3].

Chapter 3

Medical Software Development Background

As mentioned in the previous section, all medical devices are subjected to regulations intended to ensure safety for the operator and patient. As the use of software in medical devices has increased, the need for specific regulations for medical device software has increased. An indicator of this need was shown by an analysis performed by the FDA between 1992 and 1998 of 3140 medical device recalls. The analysis revealed that 242 (or 7.7%) of the recalls could be attributed to software failures. Out of those recalls, 192 (or 79%) were caused by software defects which had been introduced when changes were made to the software after its initial production and distribution [4]. Another example is from the mid 1980's, where coding errors in a radiation therapy device contributed to the lethal overdose of a number of patients [5].

In 2006, a new international standard was released for medical device software and has been developed by a joint working group from IEC and ISO[12]. This new standard has, since its release, become fully harmonized in both the EU and the US.

3.1 Software Validation and Verification

When developing software, the software validation and software verification are the two main elements to achieve acceptance with. Where *software validation* is the activity of proving that the software product does what it is stated by the intended use. *Software verification* is what is continuously performed throughout the development process by performing testing to ensure that the software code is working properly. At the end of the software development process, given that a sufficient level of software testing (verification) has been conducted, the software product can be viewed as validated through the verification effort.

The process of software validation is a requirement of the quality system regulation, which was published by the FDA in the Federal Register on October 7, 1996 and took effect on June 1, 1997.

The main component of software validation is what is described as the *software validation plan* in [4], or as the *software development plan* in [12], which

specifies areas within the development process such as scope, approach, resources, schedules and types and extent of activities, tasks, and work items.

In the following sections, I will go over some of the main components of the software development plan and what they require from a developer of medical software.

3.1.1 Quality Planning

The software development plan is, as mentioned above, not the only component of the software validation. The software validation is handled through, what is called, *Quality Planning* and this includes plans for risk management, configuration management and problem resolution as well as the development plan.

The risk management for medical software is governed by *ISO 14971 Medical devices - Application of risk management to medical devices* and additional regulatory requirements defined in *IEC 62304 Medical device software - Software life cycle processes.* These standards handles the identification and evaluation of possible hazards which are documented in a risk analysis document along with possible risk control measures.

The configuration management plan should guide and control multiple parallel development activities and ensure proper communications and documentation. There should also be controls in place to ensure that all approved versions of the specifications documents, source code, object code, and test suites that comprise the software system is kept updated and correct.

The problem resolution procedure handles all reporting and evaluation of anomalies found during the validation or after the software product has been released as well as the resolution of identified anomalies. Both the configuration management and the problem resolution is governed by *IEC 62304 Medical device software - Software life cycle processes.*

Finally, the software development plan handles the entire development of the software product and connects to the plans described above at different stages throughout the development process.

3.1.2 Software Safety Classification

As with standard medical devices, software embedded in medical devices or software that are medical devices in their own right are subject to safety classification. The regulatory requirements that a software development plan must comply with is determined by the software safety classification of the software product. The classifications and the supporting definitions are:

- 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

where serious injury is defined as an injury or illness that:

- directly or indirectly is life threatening,
- results in permanent impairment of a body function or permanent damage to a body structure, or

- necessitates medical or surgical intervention to prevent permanent impairment of a body function or permanent damage to a body structure

The hazardous situations are determined through a risk analysis of the software, until a risk analysis has been performed the software is to be considered as a Class C medical software. A medical software, that is composed of multiple parts, is classified based on the part with the highest potential hazard. Each separate part of the medical software, given that sufficient segregation can be shown, can be classified individually so that only the high risk software parts will require the extra verification effort.

The software safety classification should be performed at the start of the development and repeated as the software design and requirements change and/or are implemented.

There are ways for a developer to reduce the safety classification through risk control measures, but these must be performed through hardware modifications or additions. Risk control measures that are done through additional software functions or user instructions are not sufficient to reduce the safety classification as they are deemed less effective and secure.

3.1.3 Software Requirements

At the start of software development, there is a need to define system requirements based on the intended use of the software. These initial system requirements may change during the development process, but they will help in creating a good framework for which the software is to be kept within. Based on the system requirements, sets of software requirements needs to be defined that defines how the system requirements is to be implemented into source code. The software requirements will develop as new performance or system requirements are identified, analysis of the software requirements are conducted, and as the intended use for the software becomes more detailed. All system and software requirements is to be documented in a software requirements specification document.

Software requirements specification should include all the software system inputs and outputs as well as all the functions that the software system will perform. Every performance requirement for the software, such as reliability, data throughput, timing etc), should also be included. Within the software requirements, it should also be mentioned how users will interact with the system, what is classified as an error and how errors should be handled. In extension, the intended operating environment for the software system, such as hardware platform or operating system, should be stated if this is a design constraint.

A software requirements traceability analysis should also be conducted to trace software requirements to (and from) system requirements and to risk analysis results.

Before proceeding, the software requirements needs to be evaluated to verify that:

- There are no internal inconsistencies among the requirements;
- All of the performance requirements for the system have been spelled out;
- Fault tolerance, safety and security requirements are complete and correct;

- Software requirements are appropriate for the system hazards; and
- All of the requirements are expressed in terms that are measurable or objectively verifiable.

3.1.4 Software Design

Once the software requirement specification has been created, the next step is to transform the requirements into an architectural design of the software. The purpose of the design specification is to describe what the software should do and how it should do it. The design specification should aim to reduce the probability of use errors due to the software design being to complex or act contrary to the user's intuitive expectations.

The software requirements as well as coding guidelines and the development procedure should be included in the software design specification. It should also include information concerning any supporting software, such as operating system, drivers or other application software as well as which hardware is to be used. Systems documentation describing the context in which the program is intended to function should also be included along with a risk analysis for the software system. Additionally, information concerning error, alarm, and warning messages should be mentioned as well as what security measures (physical and logical) that have been implemented. If the software does not include alarm or warning messages, it is recommended that this be mentioned as well.

Once a software design specification has been created, an evaluation is needed to verify that the design is complete, correct and possible to be maintained. A traceability analysis should also be conducted as part of the evaluation to verify that all of the software requirements have been implemented in the software design.

As the software design specification is based on the software requirements specification, the design specification is subject to change during the software development as the requirements are updated.

3.1.5 Software Coding

The software coding activity is where the software programmers translate the design specifications into source code, or assemble previously coded software components from code libraries or from off-the-shelf software (described further in section 3.3) for use in the new software. Prior to starting the coding activity, decisions are to be made concerning the selection of which programming language to be used as well as which software development tools, such as assemblers and compilers, should be used. When selecting either which language to use or which tools to use, consideration needs be taken to the impact on the future testing and evaluation activities that will follow. The coding also needs to follow a pre-defined coding standard to ensure that all developed source code is uniform.

Before implementing developed source code into the software system, a thorough level of error checking is needed to be performed to minimize the amount of residual errors. For every encountered error that is left unresolved a rationale must be documented with a risk evaluation of the error.

To aid in the error checking and tracing down errors, code comments need to provide useful and descriptive information such as expected inputs and outputs, variables referenced, expected data types and operations performed. The code comments should also be used to aid with the source code traceability analysis. The source code traceability analysis verifies that all requirements has been implemented in the code and that all the modules and functions implemented in the code can be traced back to a requirement as well as to the risk analysis. The same traceability analysis should also be documented for the tests conducted to verify that the modules and functions work as intended.

3.1.6 Software Testing

As touched upon in the previous section, in order to verify that the software program functions according to its intended use and design specifications, testing is needed to be performed. The extent of the tests is determined largely by the software classification, as software which could cause death or serious injury if malfunctioning has to have a higher integrity than software found in an electronic thermometer. While in development, the software program will go through an iterative process which includes testing of the actual code to verify that the software behaves according to the predetermined design requirements. The testing itself consists of running the software under known conditions with documented inputs and outcomes that can be compared to their predefined expectations.

It is important to realize however, that software cannot be tested for every possible input as it is a complex system. In order to achieve full coverage of every possible input/output combination, the time requirement would be much to long for a sustainable development from a financial aspect. Instead priority should be placed on ensuring that all the critical functionalities and safety related functionalities are sufficiently tested. Further, as mentioned in *General Principals of Software Validation* [4], testing of all program functionality does not mean all of the program has been tested, neither does testing all program functionality and all program code mean that the program is 100% tested. It just means that no errors were found using that set of tests, which could be due to that the testing in itself was superficial.

As a software developer, it is important to create a software test plan that include the particular testing activities that are to be performed during each stage of the development process. These selected testing activities should be directed to verify specified cases based on the software product's internal structure and external specification. They should also provide a thorough examination of the software product's compliance with its functional, performance and interface definitions and requirements. The software test plan should also include justification for the level of effort represented by the selected testing activities and the corresponding acceptance criteria.

Following the software testing performed during development, the software needs to be tested in the actual environment where it is intended to be used. This can be done either through simulation or actual use of the software by the intended end users.

When conducting user site testing, there should be a pre-defined plan containing a summary of testing and a record of acceptance. Documentation of testing procedures, test results and test input data should also be retained

along with documentation proving that the software and supporting hardware is installed and configured as specified in accompanying documents of the software. It should also be ensured that all software system components are tested and that these are of the correct version. The user site testing plan should specify the full range of operating conditions for which the software is to be tested under as well as specify a duration sufficiently long as to allow possible conditions or events to occur that could trigger detection of previously undetected bugs. There should also be an evaluation performed on the end users ability to understand and correctly use the software.

3.1.7 Software Maintenance and Software Changes

Following the software testing activity, changes are commonly needed to be made to the software due to identified errors. As mentioned previously in chapter 3.1.1, the activity of software changes is handled by the configuration management plan. As part of the configuration management, all potential software changes needs to be evaluated to determine if the change is necessary and how the change will affect the software as well as the organization. Once a decision has been made, the software change is either implemented into the software code or the rationale for why the change was not implemented is documented as part of the risk management documentation.

Software maintenance, which handles the post-release activity of software development, is also part of the configuration management. Software maintenance primarily handles any received feedback, either from end users or internal, and performing evaluation efforts to analyze the feedback to determine the response to the feedback. If required, changes to the released software will be performed and depending on the nature of the changes, it is classified as either corrective, perfective or adaptive maintenance. Changes made to correct errors and faults in the software are *corrective* maintenance. Changes made to the software to improve the performance, maintainability, or other attributes of the software system are *perfective* maintenance. Software changes to make the software system usable in a changed environment are *adaptive* maintenance.

When changes are made to a software system, either during initial development or during post release maintenance, sufficient regression analysis and testing should be conducted to demonstrate that portions of the software not involved in the change were not adversely impacted.

3.2 Software Life Cycle Development Models

The activities that make up the software development plan can be used differently depending on the approach of the developer. These different approaches are commonly expressed as development models or life cycle models.

This section will briefly cover some of the more common software development models though there are several more development models available, such as *Evolutionary Systems Development* and *Spiral Development*, but these are not the focus of this report.

3.2.1 Waterfall Development Model

The Waterfall development model is credited to Dr. Winston W. Royce due to his article *Managing the Development of Large Software Systems [7]* and originates from an earlier development model called the Stagewise model. The Waterfall development model uses a linear approach, where each activity is only performed once and a project may not proceed into the next activity until the current activity has been completed.

Figure 3.1 shows a graphical illustration of a basic Waterfall development model. Although it does not represent the finished development model proposed by Dr. Royce, it gives a good indication to how the Waterfall model is being implemented.

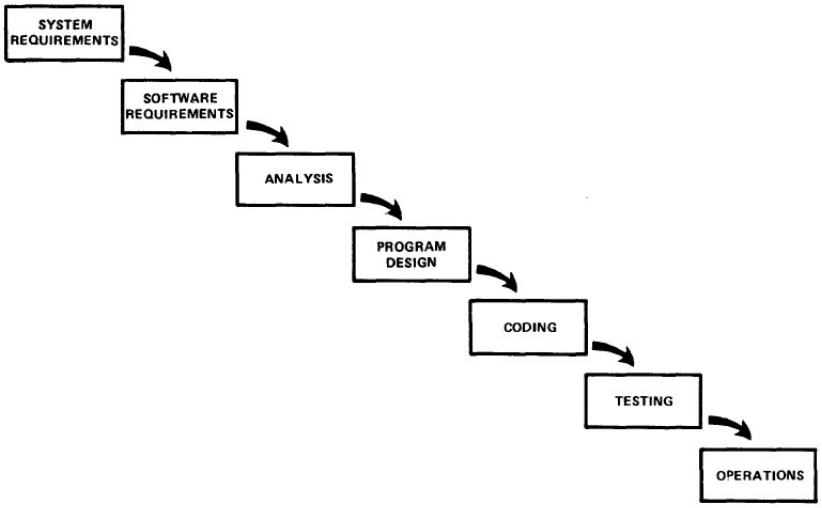


Figure 3.1: Graphical illustration of the Waterfall model [7]

For a software development, this kind of model is not optimal, as the requirements (both system and software) can change during the development process. The software design can also change which means that there is a direct need for a more iterative approach to ensure that both the requirements fully describe the software design but also that the software design fully describes the requirements.

The finished model described in *Managing the Development of Large Software Systems* [7], includes feedback loops between the different phases to correct initial errors as well as a "Do it twice" methodology.

3.2.2 Iterative and Incremental Development Model

The development model described by Dr. Royce is closer to how the Iterative and Incremental Development (IID) model is designed[8].

At the start of an IID project there is a need to subdivide the project into binary deliverables. This is done by performing a high level analysis of the project with the output of "slices", or increments. Each slice should meet a number of criterias, such as;

- implement a use case, or part of a use case,
- represent features,
- be executable and demonstrable, and
- have completion and acceptance criteria.

as described in *Iterative and Incremental Development* [8]. Once the project has been subdivided into slices, each slice is developed in isolation from the other slices until all slices have been completed.

Figure 3.2 shows a graphical illustration of the Iterative and Incremental development model, where the project is iteratively improved through testing, re-planning, re-analyzes and evaluations until the criterias have been met for the program to be deployed. The "Initial Planning" shown in the figure represents the initial subdivision of the project and other, company specific, activities (such as financial analyses). The iterative process represents the development of each slice.

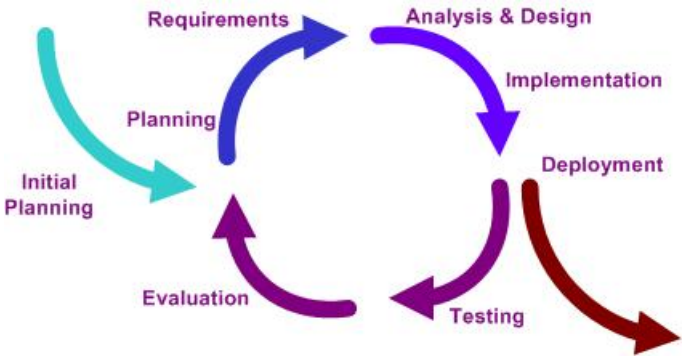


Figure 3.2: Graphical illustration of the Iterative and Incremental model [14]

However, this development model does invite the developer to stay too long in the iterative process to reach a perfect program. But releasing software without any bugs is near impossible which would lead to the software staying in development for too long and delivery dates being missed. As such, it is important to recognize that not all bugs has to be fixed for the software to be released and not to set the completion criterias to high. A well conducted risk analysis during the development will help identify which bugs are required to be fixed and which that can be left for a later software update.

3.2.3 Agile Development Model

The Agile development model is a continuation of the IID model and the most commonly used Agile development model is called SCRUM [9]. Due to this, I will here describe SCRUM as an example of Agile development.

Similar to how the IID divided the project into slices, the Scrum model uses "Sprints" which commonly last for 2-4 weeks. During these sprints, the development is carried out like the model shown in figure 3.2. Each day within

these sprints also follow this model, leading to nestled iterative development, which is depicted in figure 3.3.

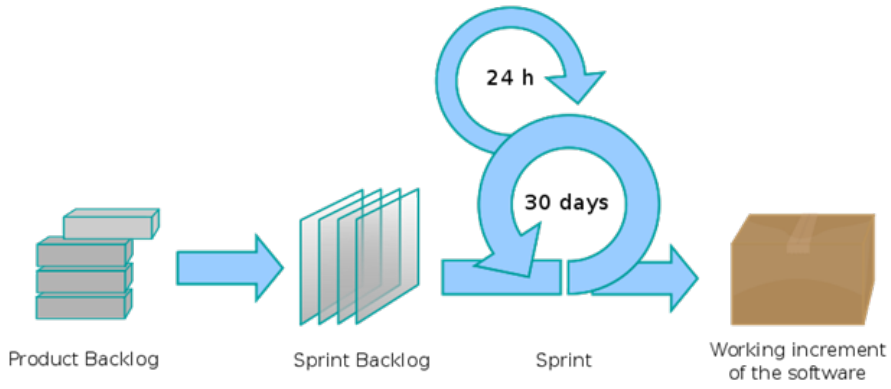


Figure 3.3: Graphical illustration of SCRUM [13]

The daily iteration starts with a SCRUM meeting where the team of developers go over what each of them will be doing during the day and bring up if there are any blockers that prevent them from performing their assigned tasks. This ensures that everyone within the developing team knows what the rest are doing and resources can be allocated to quickly deal with blockers to minimize possible delays in development.

The product backlog contains the requirements that should be implemented into the final product whilst the sprint backlog contain the different functionalities or tasks that should be completed during that sprint to achieve a working increment of the final software product.

This gives the project leader a complete overview of the development and makes it easier to get a good sense of the progress.

3.3 Off-the-Shelf Software

Manufacturers of medical devices that use software can choose to buy commercially available or open-source software instead of developing the needed software themselves. The software, which is not developed by the manufacturer and for which the manufacturer can not provide a complete software life cycle control, is then classified as "Off-the-shelf software" or OTS software. Using OTS software allows the medical device manufacturer to focus on developing their device without having to spend resources on software already existing on the market. However, the manufacturer needs to ensure that the OTS software will not introduce any risks towards the patient. It is also important to state that although the manufacturer has not developed the software used, the responsibility to ensure that the software's performance is maintained safe and effective remains with the developer [11].

As a general guidance on how much documentation the manufacturer needs to provide when using OTS software the schematic shown in figure 3.4 can be

used, which takes into account the level of concern presented by the OTS software. Level of concern is a measurement of the severity of potential hazardous situation that the OTS software is implemented to prevent, or if the OTS software in itself can cause a potential hazardous situation.

3.3.1 Software of Unknown Pedigree

As an extension to the OTS software, Software of Unknown Pedigree (sometimes Software of Unknown Provenance), or SOUP, is used to referred to software obtained from a third party and for which documentation is difficult to obtain. As documentation is essential when submitting a PMA to the FDA or to the EC, this would make all devices using third party software illegible for market distribution. Instead, by classifying a third party software as an SOUP, the documentation requirements decreases to enclose motivation to the use of the software, explaining the origin as well as presenting what documentation is available. Further, the risk analysis of the software being developed should include the risks associated with the use of the SOUP in regards to lack of documentation prior to testing the SOUP within the software device. As mentioned in *Guidance for the Content of Pre-market Submissions for Software Contained in Medical Devices*[6], the responsibility for adequate testing of the device and for providing appropriate documentation of software test plans and results remains with the developer issuing the PMA.

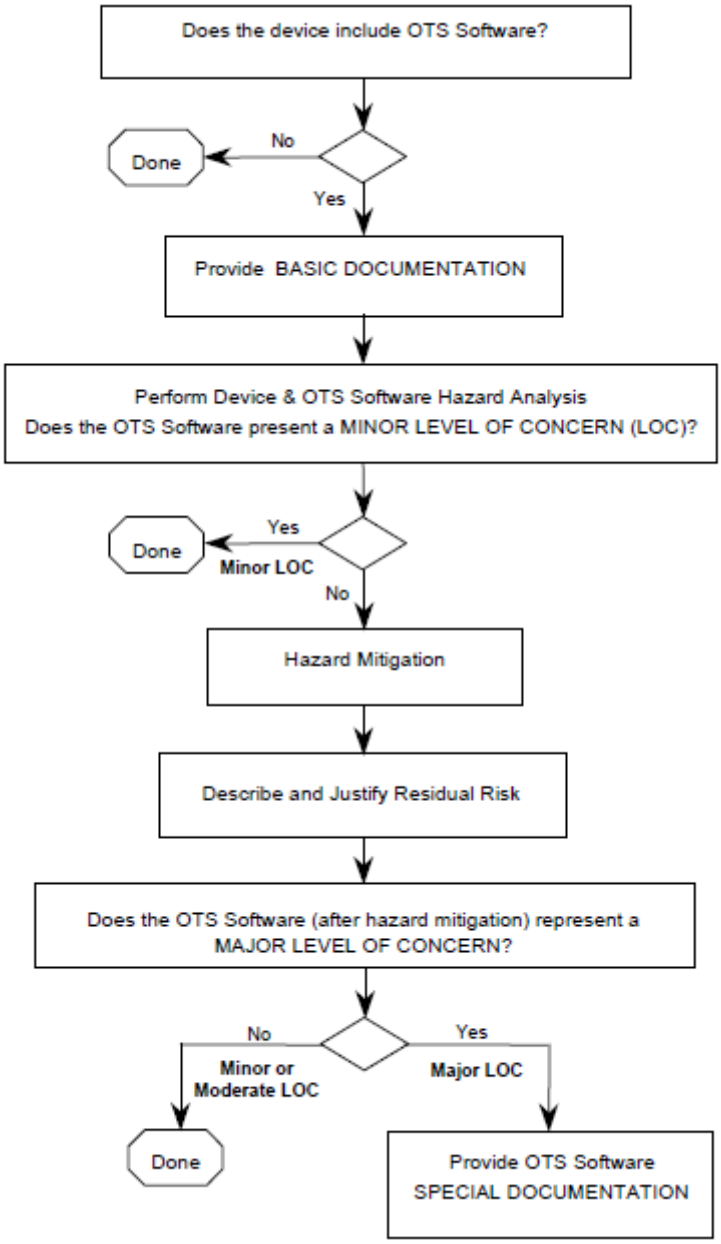


Figure 3.4: OTS Software Decision Schematic [11]

Chapter 4

Methods

This chapter describes the methods used during the master thesis project.

4.1 Literature Studies

First a literature study was conducted to gain a better understanding for software development in general and medical software development specifically. A focus was placed on obtaining guidance documents from the FDA and the EC as acceptance from these organizations is one of the main goals. The literature study also included gaining a basic understanding of the software development models described in chapter 3.2. Lastly, the literature study involved identifying current medical software regulatory standards to which compliance would be required.

4.2 Requirements Identification

Through the literature study, needed activities were identified that would make up the medical software development procedure.

The identified regulatory standard, IEC 62304 Medical device software, were used identify the different processes required to be established to provide a framework for the procedure that were to be created. The FDA guidance document to software validation were used to provide a foundation of software development practices to the framework.

Lastly, existing procedures in Integrum's quality management system were reviewed to determine if these could used for the medical software development procedure as well as to identify if there were any restrictions within the quality management system that needed to be taken into consideration.

4.3 Verification

Verification of the medical software development procedure was needed to ensure that the procedure meet the regulatory requirements. The procedure also needed to be understandable to the software development team and that the activities described in the procedure were free of redundancies.

Chapter 5

Problem Description

This chapter describes the task assigned to be performed and the different considerations that were taken into account.

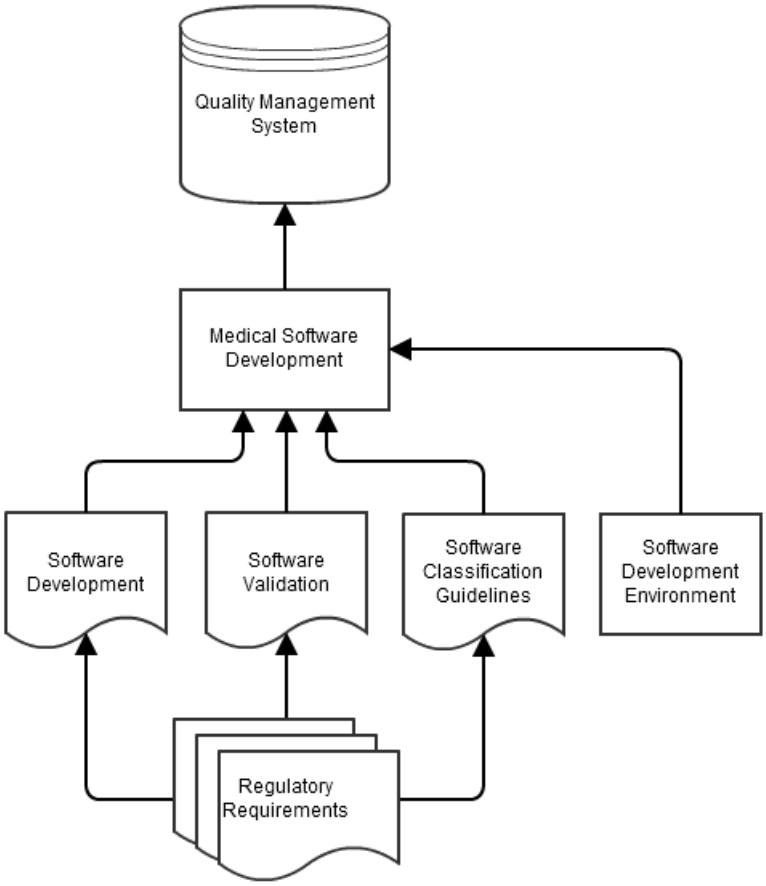
5.1 Problem Description

The task was to establish a medical software development procedure to be implemented into the quality management system at Integrum AB.

Prior to this master thesis project, there were no procedures or experience on medical software development at Integrum AB. As such, the master thesis project also came to involve the selection of supportive software development environment, safety classification of the medical device software and classification of the different medical device components connected to the medical software.

This thesis is concerning the development of the Software Development, Software Validation and Software Classification Guidelines as well as the selection of the software development environment as shown in figure 5.1.

Figure 5.1: Problem Description



Chapter 6

Design and Implementation

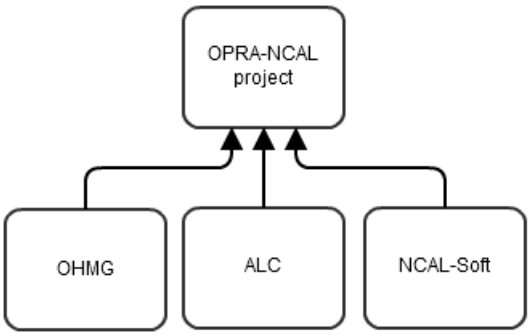
This chapter describes the design and implementation process performed in order to complete the assigned task. In the chapter, the different components that make up the project for which the medical software development procedure is to be used are considered and described in how they affect the overall project at Integrum AB.

6.1 The OPRA-NCAL Project

The OPRA-NCAL project at Integrum AB concerns the development of a robotic prosthesis that use the muscle and nerve signals of the patient to control the artificial limb. For this purpose, a medical software is needed to be developed which requires the establishment of quality management procedures for handling the development process in order to pass existing regulatory requirements.

Figure 6.1 shows how the OPRA-NCAL project is divided into its different parts. The Osseointegrated Human-Machine Gateway, OHMG, is the mechanical fixture that is implanted into the patient's bone, the Artificial Limb Controller, ALC, is the electrical component that controls the robotic prosthetic and lastly the NCAL-Soft which is the software that is used both by the ALC

Figure 6.1: OPRA-NCAL Breakdown



as well as used on a PC for training and fitting purposes.

As the OPRA-NCAL project consists of different parts, one of the initial steps that needed to be performed was to investigate the regulatory classification of these parts according to current EU and US directives.

It had already been previously been established that the complete OPRA-NCAL system is a medical device but, at the same time, the OPRA-NCAL is comprised by three parts which potentially can be classified separately with only the main part being a medical device and the other two as accessories. The benefits for doing this would be a decrease in documentation, verification and overall resources required to drive the OPRA-NCAL project through development, the regulatory classifications and onto the market.

The medical software classification for the NCAL-Soft project was also investigated in order to determine the optimal safety classification for the NCAL-Soft. This was due to a higher safety classification would require more resources from Integrum AB.

6.1.1 OPRA-NCAL Classification According to European Union

6.1.1.1 Definition of medical device

Within the EU a medical device is defined as follows from the medical device directive (MDD) 93/42/EEC:

> "a 'medical device' means any instrument, apparatus, appliance, software, material or other article, whether used alone or in combination, together with any accessories, including the software intended by its manufacturer to be used specifically for diagnostic and/or therapeutic purposes and necessary for its proper application, intended by the manufacturer to be used for human beings for the purpose of:
>
> - diagnosis, prevention, monitoring, treatment or alleviation of disease,
> - diagnosis, monitoring, treatment, alleviation of or compensation for an injury or handicap,
> - investigation, replacement or modification of the anatomy or of a physiological process,
> - control of conception,
>
> and which does not achieve its principal intended action in or on the human body by pharmacological, immunological or metabolic means, but which may be assisted in its function by such means;"

6.1.1.2 Definition of accessory

> The MDD defines a medical device accessory as:
>
> "an 'accessory' means an article which whilst not being a device is intended specifically by its manufacturer to be used together with a device to enable it to be used in accordance with the use of the device intended by the manufacturer of the device."

6.1.1.3 Medical software

In the definition of active medical device, the following amendment was implemented in 2010 through the directive 2007/47/EC:

> "Stand alone software is considered to be an active medical device."

Further amendments concerning medical software was in regards to requirements for medical devices connected to or equipped with an energy source, where the following addition was made:

> "For devices which incorporate software or which are medical software in themselves, the software must be validated according to the state of the art taking into account the principles of development lifecycle, risk management, validation and verification."

6.1.1.4 Conclusion

In accordance with the MDD definitions above, it would be possible to label the main part of medical device product as 'medical device' while labeling any attachment parts as 'accessory' to that medical device as long as the attachment on its own does not fulfill the requirements for a 'medical device'. Table 5.1 shows the devices and the different cases which would classify a device as a 'medical device'. It should also be noted that only the device on its own is taken into concideration and the functional capabilities of the device.

Device name	Case 1	Case 2	Case 3	Case 4
OHMG	No	No	Yes	No
ALC	No	No	No	No

Table 6.1: Medical device classification comparison. Case 1: Diagnosis, prevention, monitoring, treatment or alleviation of disease; Case 2: Diagnosis, monitoring, treatment, alleviation of or compensation for an injury or handicap; Case 3: Investigation, replacement or modification of the anatomy or of a physical process; Case 4: Control of conception

As seen, the OHMG fulfills one of the medical device classifications due to being implanted and thus modifying the anatomy of the patient. The ALC however does not fulfill any of the cases on its own, and can be classified as an 'accessory' to the medical device, which would be the OHMG. This means that the OPRA-NCAL medical device can be broken up into a medical device component (the OHMG) and an accessory (the ALC). NCAL-Soft is, in accordance with the 2007/47/EC directive, an active medical device and is therefore not included in table 6.1.

It should finally be mentioned that, while the OHMG, ALC and NCAL-Soft all treat a handicap when combined, they do not fulfill this case when isolated.

6.1.2 OPRA-NCAL Classification According to the United States of America

6.1.2.1 Definition of medical device

The FDA's definition of a medical device is as follows:

> "an instrument, apparatus, implement, machine, contrivance, implant, in vitro reagent, or other similar or related article, including a component part, or accessory which is:
>
> - recognized in the official National Formulary, or the United States Pharmacopoeia, or any supplement to them,
>
> - intended for use in the diagnosis of disease or other conditions, or in the cure, mitigation, treatment, or prevention of disease, in man or other animals, or
>
> - intended to affect the structure or any function of the body of man or other animals, and which does not achieve any of its primary intended purposes through chemical action within or on the body of man or other animals and which is not dependent upon being metabolized for the achievement of any of its primary intended purposes."

6.1.2.2 Conclusion

This would mean that if a medical device consists of multiple parts, all those parts would be labeled as medical devices due to that the product they form is a medical device. For the OPRA-NCAL product, both the OHMG and the ALC would be labeled as medical devices as they together form the actual medical device product.

6.1.3 NCAL-Soft

The NCAL-Soft is the software component to the OPRA-NCAL project and is separated into two parts. One part to be installed on a PC for training and fitting purposes and another intended to be installed into the ALC component of the OPRA-NCAL medical device. NCAL-Soft is described further in chapter 6.2.

In accordance with *IEC 62304 Medical device software - Software life cycle processes*, all medical software being developed for use in a medical device or to be used as a medical device in its own right must be given a safety classification based on the potential hazardous situation that could arise from a software failure.

6.1.3.1 Classification of NCAL-Soft

As the two parts of NCAL-Soft are installed in separate devices and does not interface with each other, they can be given an individual safety classification as mentioned in chapter 3.1.2.

ALC Following a preliminary hazard analysis of the ALC component for the NCAL-Soft, the worst potential risk associated with a hazardous situation would result in bruising. Other identified potential risks would result in a delay of prosthetic movement or no movement at all.

However, there are possibilities to reduce the safety classification through hardware implemented risk control measures. The effects of these hardware changes would need to be analyzed to determine the impact it would have on the

performance of the medical device before a decision can be made to implement them or not.

PC For the PC component of the NCAL-Soft the only concern would be during the sessions where the patient is training the ALC software through the PC, but this should not give rise to any risks unless the patient deliberately act against the user instructions.

6.1.3.2 Conclusion

Based on the preliminary risk analysis it is determined that the ALC component of the NCAL-Soft will have a Class B safety classification while the PC component of the NCAL-Soft will have a Class A safety classification.

6.1.3.3 Evaluation of Safety Classification Reduction

The gain of a reduced safety classification for Integrum AB would be the reduction in documentation and verification effort. This reduction could save resources and possibly result in a faster project closure.

In order to reduce the safety classification of a medical device software, as described in chapter 3.1.2, risk control measures have to be implemented through hardware changes or adding additional hardware to the medical device. So when considering a reduction of safety classification, it is needed to take the impact of hardware changes on the medical device performance into consideration.

One of the identified risks would be accidental bruising due to the prosthesis performing an unintentional action, such as closing of prosthetic hand while holding another person's hand. A possible hardware solution to this problem would be to change the prosthetic motor to a less powerful motor, but it would need to be verified that the intended use can still be met.

6.2 NCAL-Soft

The NCAL-Soft is a training and fitting software designed for the OPRA-NCAL medical device currently being developed by Integrum AB. The project leader is Max J. Ortiz C., who also is the project leader for the OPRA-NCAL project. Here, there different tasks concerning the development of NCAL-Soft are described.

6.2.1 Intended Use

Defining a medical device's intended use is a critical part and is one of core aspects taken into consideration when performing the risk assessment. The intended use should clearly define what the product is designed to do as well as any restrictions in how it is to be used.

One of the first tasks was to define the intended use for NCAL-Soft and the ALC. Due to confidentiality, these definitions can not be presented in this thesis.

6.2.2 Risk Assessment

The risk assessment of medical software must, like any other medical device, comply with the ISO 14971:2007 standard for risk management for medical devices. The major difference, however, between performing a risk assessment on a software medical device contra that of a classic medical device, is that the probability of a risk or hazard occurring can not be estimated in the same statistical way due to software's unpredictability. As such, any foreseeable risk or hazard has a 100% chance of occurring.

As previously stated, the outcome of the risk assessment of NCAL-Soft was a worst case risk of bruising.

6.2.3 Software Safety Classification

As covered previous in chapter 3.1.2, software safety classification needs to be performed at the start of the software development. With NCAL-Soft marking the first software development project at Integrum AB, there were no existing procedures for how the software safety classification should be performed. As such, a classification strategy document for how the software classification should be performed in compliance with the IEC 62304 standard needed to be created. The safety classification for the NCAL-Soft medical software was generated at the same time and documented as NCAL-Soft Classification.

The safety classification strategy includes the identification of the different software entities that comprise the software system and how they interface. It also includes performing risk analysis on the identified entities and the specification of risk control measures.

The software safety classification strategy is documented as Software Classification Guidelines procedure which is to be implemented into the quality management system at Integrum AB and used to determine the software safety classification of any medical software project that Integrum AB initiates.

6.2.4 Software Development Environment

When developing a software, the choice of a suitable software development environment (SDE) needs to be considered. For Integrum, the requirements on a SDE was based on the compliance requirements for the IEC 62304, features such as traceability (risk assessment to code and design requirements to code), revision control, activity planning and document sharing between the developers.

Different SDE's that could be of use to Integrum in the development of NCAL-Soft needed to be identified and evaluated.

The first possible solution was from Mortice Kerns Systems which solution, MKS Integrity, offers full SDE solutions including integrated risk management and quality management but also includes features that were not desirable for the NCAL-Soft project. The cost for the license was another factor which played a part in looking for a different solution.

A second solution was found in IBM Rational SDE which, like MKS Integrity, includes integrated risk management, quality management as well as the possibility of creating a full traceability between implemented software requirements, risk analysis and actual source code. The biggest perk for IBM

SDE	Trace-ability	Revision control	Integrated RM	Integrated QM	License
MKS Integrity	Yes	Yes	Yes	Yes	-
IBM Rational	Yes	Yes	Yes	Yes	Free
rt:collabs	Yes	Yes	No	No	-

Table 6.2: SDE Feature Comparison

Rational is that it is free as long as there are 10 or less developers using it, making it a very attractive choice for small companies, the downside of that is that there is no IBM support included in the free license.

A third solution found is the rt:collabs SDE which includes the possibility of having traceability between source code and risk analysis or software requirements through the use of tags and comments, increasing the manual responsibility. It is designed to use a SVN client for revision control and has a web based user-interface that is built on TracWiki [15].

Table 6.2 shows a breakdown comparison of the three possible solutions described above.

The decision of SDE ended up being between IBM Rational and rt:collabs. Both could offer similar features, with IBM having some additional features over rt:collabs as well as being free. The trade-off would be that the rt:collabs license would include support whilst Integrum would be without support with the IBM solution.

The rt:collabs platform was ultimately selected due to its high level of configuration, that it can be used as a platform for more than just software development and that support would be available.

Following the choice of rt:collabs came the need of identifying a suitable SVN client. A number of different SVN clients were identified, such as Visual Studio SVN and Tortoise SVN. Tortoise SVN was selected as it works well with any file type and does not require a specific programming environment.

6.3 Software Development Procedure

As the NCAL-Soft marks the first software product being developed by Integrum AB, there was a need to create a software development procedure to be implemented into Integrum's quality system which would ensure that the development is conducted in accordance with current regulatory requirements for medical software.

The first aspect to consider was which of the software development models described in chapter 3.2 would fit Integrum AB as an organization, its development team and their quality system. The development model would have to work with a very small team and require as little additional resources from Integrum AB as an organization also if it has a similar structure to the existing development procedure in place at Integrum AB for their non-software products would make it easier to implement but that is of lesser importance.

The main task of this thesis project was the development of a set of software development procedures to assure compliance with the regulatory requirements of *IEC 62304 Medical device software - Software life cycle processes.*

This was done by initially performing a quick review of the quality management system in place and compare the implemented standard operating procedures, SOP, present in it, with the requirements within the standard. From there, it could determined if new SOP's were needed to be created, or if changes to present SOP's would be sufficient.

6.3.1 Proposed Software Development Procedure

The software development procedure described in this section is that of an iterative development model. This is due to that software development is an iterative process but also due to that it is the basis for the proposed software development procedure for Integrum AB. It is also easy to verify that this development procedure follows the general principals of software validation by comparing the activities described in Chapter 3.1 with the processes described below.

The software development would start with an initial planning process, as shown in figure 3.2 on page 15, before the actual development starts. The development outline is shown below with a brief description of each of the processes.

- Initial Planning
- Requirements
- Implementation
- Testing
- Evaluation
- Release

Initial Planning

The initial planning deals with all the different tasks that are needed to be performed before starting a project. It covers initial risk analysis, initial software designs and establishing software system requirements. But also financial estimates and other factors that affect the company.

Requirements

As mentioned in Chapter 3.1.3, the requirements activity should identify and define the various software and system requirements and ensure that any additional requirements that are identified during the development process is properly defined for a successful implementation.

Implementation

This activity is where all the requirements (system and software), as well as any identified risk control measures, are implemented into actual source code.

Testing

The testing activity is the verification effort to ensure that the source code fulfills its intended function, and that any new code or functions implemented into the software does not cause conflicts. It is through this verification effort that the software will be validated for completeness at the end of the project.

Evaluation

The activity of evaluation is where the test results are analyzed and decisions are made on how the software architectural design shall be refined, if additional risk control measures shall be implemented and if an implemented functionality has reached the acceptance criteria. During this activity, the software development plan is updated to include the results of the iteration ending as well as add appropriate documentation concerning the iteration that is to start.

Release

Once the evaluation activity shows that all requirements and risk control measures have successfully been implemented and that the software has reached the acceptance criteria, it will enter the release activity. Documentation shall be created identifying any known software anomalies that has not been corrected along with a motivation for why this has not been done. An evaluation effort shall also be performed to ensure that all created documentation supports that the software has been completed.

6.3.2 Creating a Software Development Procedure

After deciding on software development model to use, the next step was to create the necessary operating procedures for software development based on the requirements set by the IEC 62304 standard.

The first step was to identify the procedures required to be established, followed by defining what these procedures should cover in terms of clauses of the IEC 62304 standard. This resulted in the following list of procedures that needed to be defined and developed:

- Software Development Process
- Software Design Process
- Software Integration and Testing Process
- Software Configuration Management Process
- Software Problem Resolution Process
- Software Maintenance Process
- Software Release Process
- Software Risk Management Process

Activity:	Process:
Initial Planning	-
Requirements	Software Development, Software Design
Design	Software Design
Implementation	Software Integration and Testing
Testing	Software Integration and Testing
Evaluation	Software Configuration Management, Software Problem Resolution
Release	Software Release

Table 6.3: Connection between development activity and development processes

Comparing this list of procedures with the Iterative and Incremental development model shown on page on page 15, each process is connected to a corresponding activity. This connection is shown in table 6.3.

As can be seen in this table, the risk management process and the maintenance process does not have a dedicated development model activity but exist outside of the illustrated model shown in 3.2. It is also shown that the initial planning activity is not covered by the IEC 62304 standard, but it is described in the *General Principals of Software Validation* [4].

Once these processes had been identified, the work of creating and refining them to ensure that they were clear in what is required to be done and how this is to be done began. As part of this process, the software design, software maintenance and software release process were merged into one single process labeled software design and release process. Further, the requirements activity was integrated into a separate document labeled *IEC 62304 Implementation* that serves as a guiding document to how the IEC 62304 regulatory requirements should be implemented into the rt:collabs TracWiki platform that is used by Integrum AB.

At this point, there were several documents describing the different processes involved in medical software development but there were no clear step-by-step procedure that described how these processes could easily be used by the programmers at Integrum AB. Bearing that in mind, an effort was made in re-writing the existing processes, listed in table 6.3, into a step-by-step procedure called *Software Development*. The re-write also included the addition of references to the *IEC 62304 Implementation* document in the cases where a more detailed description would be needed, i.e. for the definition of requirements.

As Integrum AB already has an existing risk management procedure, there was a concern of how the software risk management would best be implemented into the quality management system. After going over the process with the quality manager for Integrum AB, it was decided that a new separate procedure should be created which would cover the software risk management as well as the software release and software maintenance. This new procedure was labeled *Software Validation*.

The final software project development process is described by the flowchart shown in figure 6.2.

The *Software Development* procedure is described in detail by figure 6.3. It should be noted that the last step in the flowchart is the same step described in 6.2 and does not indicate a need to repeat the activities described in the

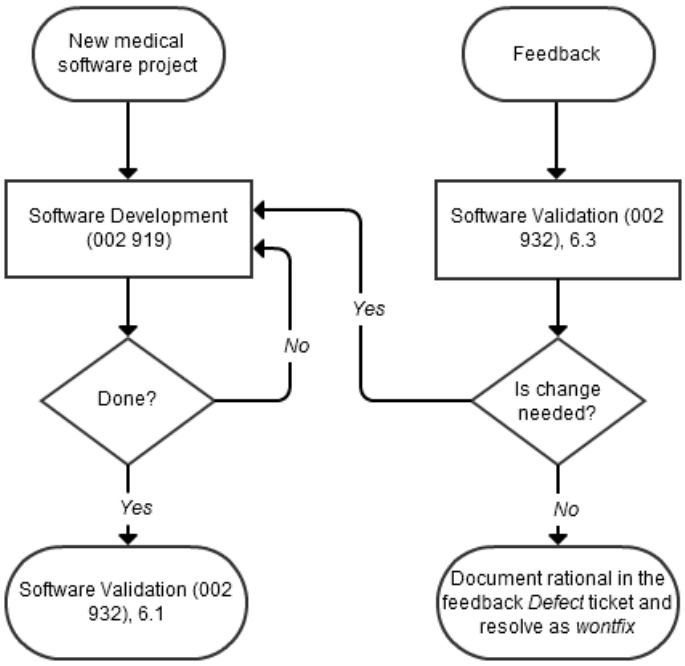


Figure 6.2: Software Development Procedure Flowchart

Software Product Release process defined in the *Software Validation* procedure.

The integration plan described in figure 6.3 refers to the process that is described by figure 6.4. This flowchart describes the integration testing process that all source code is required to go through before being implemented into the software product and the testing of the software system before being submitted to the *Software Product Release* process defined in the *Software Validation* procedure.

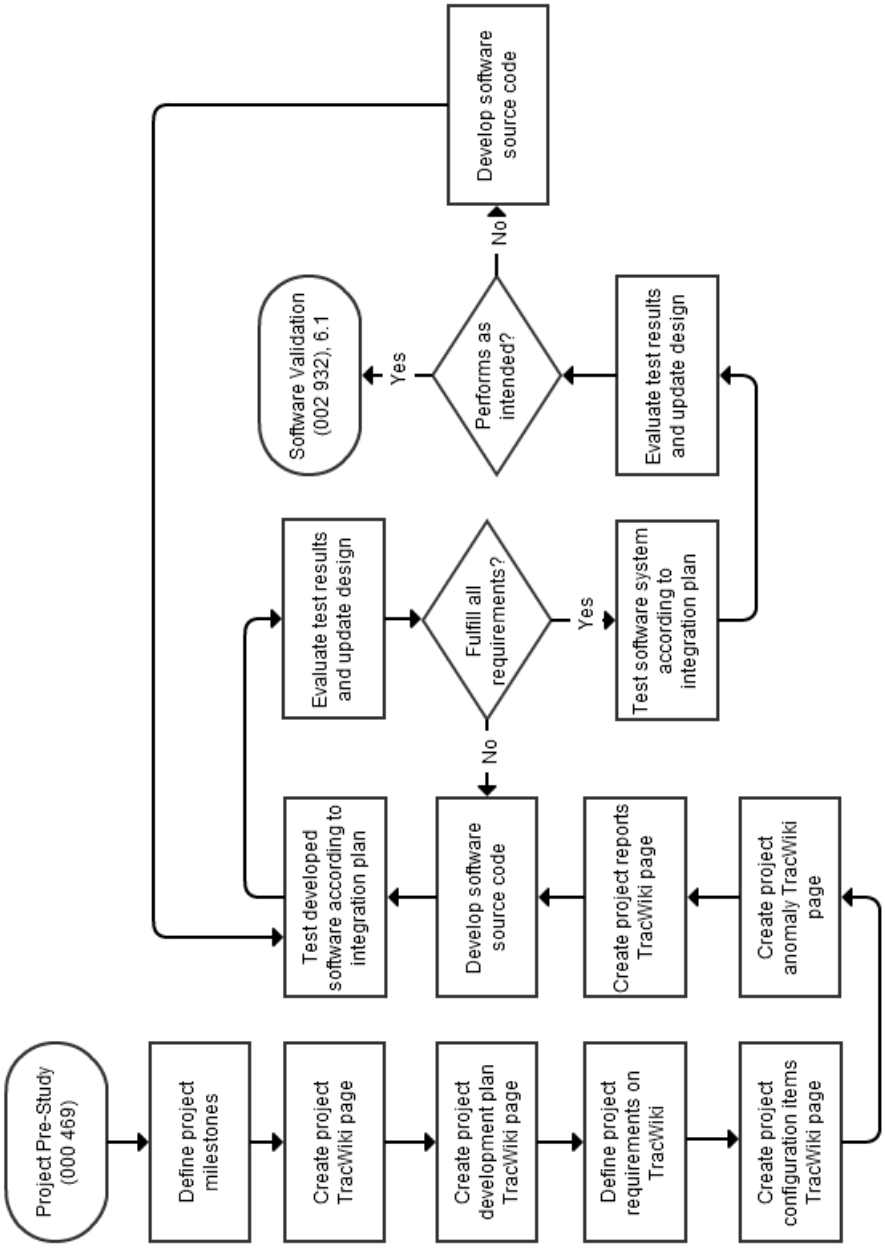


Figure 6.3: Software Development Method Flowchart

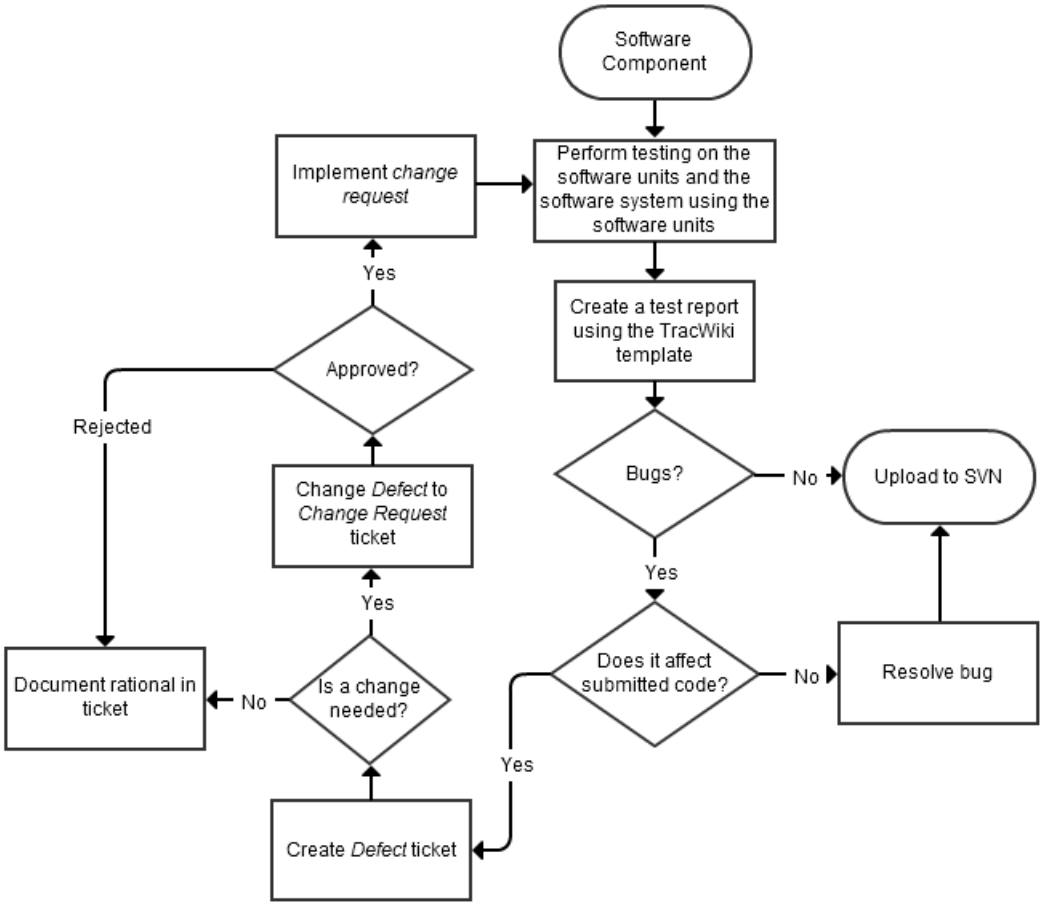


Figure 6.4: Software Integration Testing Flowchart

Chapter 7

Verification

This chapter describes the verification effort that was performed to ensure that the medical software development procedure works as intended and fulfill the regulatory requirements set by IEC 62304 Medical device software. The verification was performed in two different ways, one through the actual use of the procedure and another by verifying that each required regulatory clause was met.

7.1 Verification of Performance

This verification was initially conducted by running a dummy project through the procedure to catch any inconsistencies or redundancies.

When all identified redundancies and inconsistencies had been corrected, the procedure was handed to the medical software development team for "live" testing. Although the use of the software development procedure was highly limited during this test, new issues that needed to be resolved were still identified.

7.2 Verification of Compliance

This verification stage was performed at the end of the project and consisted of going through each regulatory clause and verify that the clause had been met. This was done over the course of several meeting between myself and the medical software development team. It was completed up to clause 7 of the IEC 62304 standard, Risk Management Process.

Chapter 8

Discussion

8.1 NCAL-Soft Safety Classification

The NCAL-Soft safety classification is based on a preliminary risk analysis that will be needed to be repeated once a more detailed design of the NCAL-Soft project has been established. It is also of importance that a risk analysis is performed continuously throughout the development process to ensure that no potential hazardous situations are missed.

Also, it was determined that Integrum AB will not be developing any medical software that would be classified as a Class C medical software given their current projects. However, in the event that Integrum AB start developing medical software that will actively stimulate biological tissue this needs to be evaluated.

8.2 Software Development Procedure

The proposed software development process created for Integrum AB is highly based on the IID model described in section 3.2.2 but adapted to fit the small development team at Integrum AB and the TracWiki development platform. It consists of a software development procedure, a software validation procedure and IEC 62304 Implementation guidance document.

The software development procedure provides a step-by-step method for how a medical software development project should be performed for compliance with IEC 62304 Medical software standard and includes flowchart representations of the software project development process and integration testing process. The software validation procedure provides instructions for performing validation prior to releasing the software product to the market, post-release activities for handling feedback and product maintenance as well as software risk management. The IEC 62304 Implementation guidance document provides more detailed explanations for why different activities needs to be performed by referencing directly to the IEC 62304 regulatory clause that activity complies with. It also offers additional information concerning definition of requirements and software design documentation.

As previously mentioned, it was evaluated that Integrum AB will not develop any Class C medical software. As such, the software development procedure is

restricted to only comply with the regulatory requirements for Class A and Class B medical software. In the event that Integrum AB will be developing Class C medical software, additions will be needed to be made to the software development procedure to include the regulatory requirements for Class C.

8.3 Future Work

As with most development work, there is always room for further work and improvements.

The verification of compliance needs to be completed by verifying that remaining clauses are met. Further, the software development procedures need to be used in a real development instead of simulated development projects. Although the software development procedures were used as part of the NCALSoft project, it was not used to a wide extent and the use could not give any qualitative feedback on the procedure's performance.

I also believe that Integrum AB would benefit from looking into obtaining guidance documents concerning software requirement specification and software system requirement specification. Though it is not required by the medical software standard, I believe it will aid Integrum AB in their expansion into a medical software development company. Example of these guidance documents is the IEEE STD 830-1998 for software requirement specification and the IEEE STD 1233-1998 for software system requirement specification. Although I believe these documents would result in changes needed to be made to the software development procedure. They should also result in better documented requirements which will lead to a smoother software development, as well as decrease the FDA processing.

Bibliography

[1] *Device Classification.* Food and Drug Administration [cited 2012-06-07]; Available from: *http://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/Overview/ClassifyYourDevice/ucm2005371.htm*

[2] *Premarket Notifications (510k).* Food and Drug Administration [cited 2012-06-07]; Available from: *http://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/ HowtoMarketYourDevice/PremarketSubmissions/PremarketNotification510k/default.htm*

[3] *Guide to the implementation of directives based on the New Approach and the Global Approach*, European Commission, 2000

[4] *General Principles of Software Validation; Final Guidance for Industry and FDA Staff*, Center for Devices and Radiological Health, Food and Drug Administration, January 2002

[5] *A Formal Methods-based verification approach to medical device software analysis.* EE Times [cited 2012-06-07]; Available from: *http://www.eetimes.com/design/embedded/4008888/A-Formal-Methods-based-verification-approach-to-medical-device-software-analysis*

[6] *Guidance for the Content of Pre-market Submissions for Software Contained in Medical Devices*, Office of Device Evaluation, Center for Devices and Radiological Health, Food and Drug Administration, May 2005

[7] Royce, Winston, *Managing the Development of Large Software Systems*, Proceedings of IEEE WESCON 26:1-9, August 1970

[8] Martin, Robert, *Iterative and Incremental Development*, Engineering Notebook Column, April 1999

[9] Sutherland, Jeff, *Agile Development: Lessons Learned From the First SCRUM*, Cutter Agile Project Management Advisory Service. Executive Update, Vol. 5, No. 20, October 2004

[10] Boehm, Barry, *A Spiral Model of Software Development and Enhancement*, TRW Defense Systems Group, May 1988

[11] *Guidance for Industry, FDA Reviewers and Compliance on Off-the-Shelf Software Use in Medical Devices*, Officer of Device Evaluation, Center for

Devices and Radiological Health, Food and Drug Administration, September 1999

[12] IEC 62304, *Medical device software - Software life cycle processes*, International Electrotechnical Commission, May 2006

[13] *Scrum (development).* Wikipedia [cited 2012-06-07]; Available from: *http://en.wikipedia.org/wiki/Scrum_(development)*

[14] *Iterative and incremental development.* Wikipedia [cited 2012-06-07]; *http://en.wikipedia.org/wiki/Iterative_and_incremental_development*

[15] *TracWiki - The Trac Project.* Trac [cited 2012-06-07]; Available from: *http://trac.edgewall.org/wiki/TracWiki*