Hunor Kovács, BSc

# Development of a software for the documentation of the recurrent safety testing of active medical devices according to the Medical Device Operator Ordinance

**Master's Thesis**

to achieve the university degree of

Dipl.-Ing.

Master's degree programme: Biomedical Engineering

submitted to

**Graz University of Technology**

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Baumgartner Christian

Institute for Health Care Engineering
Head: Univ.-Prof. Dipl.-Ing. Dr.techn. Baumgartner Christian

Graz, April 2020

# Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

# Acknowledgment

First of all I would like to thank Horst Friedrich Lechner, the domain expert and former employee of the European Testing Center of Medical Devices at the Graz University of Technology, for giving me a deeper insight into the area of the recurrent safety testing of medical devices - and for always having time for my questions. Unfortunately, he left the testing center before he could use this application.

Special thanks to Sebastian Lassacher, Bernhard Ablinger and Christoph van der Fecht, who stood by for subject-specific questions with their knowledge of software design and development. Furthermore, I would like to thank Tanya Kaindlbauer for her support in reviewing this master's thesis.

Finally, I want to express my gratitude to my parents for giving me their support and the opportunity to study.

# Kurzfassung

Die wiederkehrenden sicherheitstechnischen Prüfungen von aktiven Medizinprodukten wurden bisher von der Arbeitsgruppe der Europaprüfstelle für Medizinprodukte am Institut für Health Care Engineering an der Technischen Universität Graz von Hand über vorgefertigte DOCX-Protokollvorlagen abgewickelt. Die manuelle Digitalisierung dieser handschriftlichen Dokumentationen führte allerdings zu einem hohen Verwaltungsaufwand. Diese Masterarbeit zielt darauf ab, dieses Problem zu lösen und befasst sich daher mit der Entwicklung einer Software, die die Dokumentation der Prüfungen automatisiert. Im Umfang dieser Arbeit befindet sich nicht nur der regulatorische Hintergrund für die Durchführung und Dokumentation der Tests gemäß der Medizinproduktebetreiberverordnung und der Norm IEC 62353. Auch ein Überblick über die für das Software-Design relevanten Konzepte und Entwurfsmuster sowie ausgewählte Details zur tatsächlichen Implementierung sind enthalten. Letztendlich will diese Masterarbeit eine erweiterbare und wartbare Basis für weitere Entwicklungen bieten.

Schlüsselwörter: wiederkehrende sicherheitstechnische Prüfung, Medizinproduktebetreiberverordnung, IEC 62353, Qt framework, C++

# Abstract

As a working group of the Institute of Health Care Engineering at the Graz University of Technology, the staff of the European Testing Center of Medical Devices documented the recurrent safety tests of active medical devices by hand with prefabricated test protocol templates in DOCX format. The manual digitization of the these handwritten documentations resulted in a high administrative effort. This master's thesis aims to solve this problem and deals with the development of a software that takes over the documentation of these recurrent safety testings. The scope of this work includes not just the regulatory background for the performance and documentation of the tests in accordance with the Medical Device Operator Ordinance and the IEC 62353 standard. It furthermore provides an overview of the concepts and patterns used in relation to the software design as well as selected details on the actual implementation. Ultimately, this master's thesis offers an expandable and maintainable basis for further developments.

Keywords: recurrent safety testing, Medical Device Operator Ordinance, IEC 62353, Qt framework, C++

# Contents

# List of Figures

# List of Listings

# List of Tables

# 1. Introduction

The application mentioned in this thesis is being developed for the European Testing Center of Medical Devices (PMG) at the Graz University of Technology. The PMG is part of the Institute of Health Care Engineering (HCE) as a working group and is federally accredited by the Austrian Federal Ministry for Digital and Economic Affairs [1]. Before the development of this software, the Recurrent Safety Test (STK)[1] of medical devices was performed handwritten at the PMG with prefabricated test protocols that lead to a huge amount of administrative work. Besides the performance of accredited recurrent testings, the PMG also does type testings and special safety-related testings of active medical devices [1].

Before diving into the aims and goals of this thesis, the regulatory background for performing STKs is being discussed.

## 1.1. Regulatory background

The point to start with the regulatory background is the Medical Device Operator Ordinance (MPBV)[2] as it defines the duty of the operator of health care facilities in conjunction of which medical devices have to undergo a STK at specified intervals. Without this theoretical basis, no suitable application can be developed.

[1] STK is German and stands for "Recurrent Safety Test" (wiederkehrende sicherheitstechnische Kontrolle).

[2] MPBV is the German short term for "Medical Device Operator Ordinance" (Medizinproduktebetreiberverordnung).

### 1.1.1. Recurrent safety tests according to the MPBV

The important part for this thesis in the MPBV (BGBl.-Nr.70 / 2007) is paragraph 6 that subjects the STK [2]. The ordinance is only available in German and is therefore also cited here in its original version:

> *"§ 6. (1) Die Betreiberin/Der Betreiber hat*
>
> *1. bei aktiven nicht implantierbaren Medizinprodukten und*
>
> *2. auf Verlangen des Herstellers auch bei nicht aktiven nicht implantierbaren Medizinprodukten eine wiederkehrende sicherheitstechnische Prüfung vorzunehmen oder vornehmen zu lassen.*
>
> *(2) Abs. 1 Z 1 gilt nicht für aktive nicht implantierbare Medizinprodukte, die nicht im Anhang 1 [within this source] genannt und ausschließlich batteriebetrieben sind, es sei denn, der Hersteller hat für diese Medizinprodukte eine wiederkehrende sicherheitstechnische Prüfung vorgeschrieben.*
>
> *(3) Wenn der Hersteller eine wiederkehrende sicherheitstechnische Prüfung vorgeschrieben hat, ist diese nach dem in den Begleitpapieren enthaltenen Prüfumfang und in dem angegebenen Prüfintervall durchzuführen. In sicherheitstechnisch zu begründenden Einzelfällen kann eine fachlich geeignete Person ein kürzeres Prüfintervall vorschreiben oder den Prüfumfang erweitern. Derartige Festlegungen sind zusammen mit den Begründungen zu dokumentieren. Hat der Hersteller die Notwendigkeit einer wiederkehrenden sicherheitstechnischen Prüfung ausdrücklich ausgeschlossen, ist zumindest eine Sichtprüfung vorzunehmen. [...]*
>
> *(4) Wenn Angaben des Herstellers über eine wiederkehrende sicherheitstechnische Prüfung nicht vorliegen, ist die wiederkehrende sicherheitstechnische Prüfung, die auch sicherheitsrelevante Funktionskontrollen einzuschließen hat, nach dem Stand der Technik durchzuführen.*
>
> *(5) Liegen für das Prüfintervall keine Herstellerangaben vor, so ist dieses von einer fachlich geeigneten Person festzulegen, wobei das Prüfintervall*

*unter Berücksichtigung von Geräteart und Gefährdungspotential in der Regel zwischen sechs und 36 Monaten, für die im Anhang 1 [within this source] genannten Medizinprodukte jedoch zwischen sechs und 24 Monate, zu betragen hat. Zur Bestimmung des Gefährdungspotentials sind Gefährdungsgrad des Gerätes, Einsatzhäufigkeit, Unersetzbarkeit des Gerätes, Betriebsort (insbesondere Ordinationsstätte oder Krankenanstalt), Eigentumsverhältnisse, Einsatzort (stationär, mobil, Notfall) und Fehlerhäufigkeit zu berücksichtigen. Eine Überschreitung des von der fachlich geeigneten Person festgelegten Prüfintervalls ist unter Berücksichtigung von Geräteart und Gefährdungspotential bis zu sechs Monaten zulässig.*

*(6) Die Abs. 1 bis 5 gelten auch für die Zusammenschaltung von aktiven Medizinprodukten mit anderen Medizinprodukten oder nichtmedizinischen Produkten zu aktiven medizinischen Systemen.*

*(7) Eine wiederkehrende sicherheitstechnische Prüfung dürfen nur Personen oder Stellen durchführen, die die Anforderungen nach Anhang 3 [within this source] erfüllen.*

*(8) Über die wiederkehrende sicherheitstechnische Prüfung ist ein Protokoll anzufertigen, welches die Identifikation der Prüferin/des Prüfers, das Datum der Durchführung, Art und Umfang der Prüfung und die Ergebnisse unter Angabe der ermittelten Messwerte und der Messverfahren sowie die Gesamtbeurteilung zu enthalten hat. Die Betreiberin/der Betreiber hat das Protokoll mindestens fünf Jahre aufzubewahren.*

*(9) Die geprüften Medizinprodukte sind bei erfolgreicher Prüfung mit dem Datum der nächsten Prüfung (Monat, Jahr) zu kennzeichnen."*

Cited from [2]

The MPBV provides vital information about the duty of the operator of health care facilities with regard to the STK of active medical devices. This includes

when and at which intervals to perform STKs based on the type and risk potential of the devices.

An "active medical device" is defined by the Council Directive 93/42/EEC on medical devices as follows:

> *"Any medical device operation of which depends on a source of electrical energy or any source of power other than that directly generated by the human body or gravity and which acts by converting this energy. Medical devices intended to transmit energy, substances or other elements between an active medical device and the patient, without any significant change, are not considered to be active medical devices."*

Cited from [3]

#### 1.1.1.1. Scope of STKs

A STK according to IEC 62353[3], which is implemented in Austria as ÖVE/ÖNORM EN 62353, consists of the following three parts [4]:

- Visual Inspection
- Measurement of the electrical safety parameters
  - Protective Earth Resistance
  - Insulation Resistance (optional)
  - Leakage currents
    - Equipment Leakage Current
    - Touch Current
    - Applied Part Leakage Current
- Functional test

[3]STK is referred to in the IEC 62353 as "recurrent test".

When using the application for performing and documenting STKs, these three parts have to be covered.

Like the MPBV, IEC 62353 also proposes that the manufacturer's specifications should be taken into account when performing an STK [2, 4]. For measuring the "Equipment Leakage Current" or the "Applied Part Leakage Current", one of the following methods may be used [4]:

- alternative method
- direct method
- differential method

The chosen method depends on medical electrical devices or the medical electrical system and the limits vary on the applied part type (B, BF, CF) [4].

> *"Alternatively, for measurements of EARTH LEAKAGE CURRENT, TOUCH CURRENT and PATIENT LEAKAGE CURRENT test configurations derived from IEC 60601-1 (all editions) may be used."*
>
> Cited from [5]

The mentioned measurements of "Earth Leakage Current", "Touch Current" and "Patient Leakage Current" have to be done for Normal Condition (NC) and Single Fault Condition (NFC) each. The limit values differ depending on the applied part type. [5]

#### 1.1.1.2. Documentation of STKs

A documentation shall be prepared of the STK, which contains the identification of the inspector, the date of the test, the type and scope of the test as well as the results with the determined measured values and the measurement procedures and the overall assessment. The operator shall keep the record for at least five years. [2]

The MPBV paragraph 6 (see chapter 1.1.1) provides some basic information of which information must be given at the documentation of STKs. IEC 62353 offers a more detailed description of the documentation:

> *"All tests performed shall be documented. The set of documentation shall comprise at minimum the following data:*
>
> - *identification of the testing body (e.g. company, department);*
> - *name of the person(s) who has/have performed the testing and the evaluation(s);*
> - *identification of the equipment/system (e.g. type, serial number, inventory number) and the ACCESSORIES tested;*
> - *tests and measurements;*
> - *date, type and outcome / results of:*
>   - *visual INSPECTIONS;*
>   - *measurements (measured values, measuring method, measuring equipment);*
>   - *functional testing [...];*
> - *concluding evaluation;*
> - *date and confirmation of the individual who performed the evaluation; if using electronic documentation an assignment to inspector / evaluator shall be ensured.*
> - *if applicable (decided by the RESPONSIBLE ORGANIZATION), the equipment/system tested shall be marked / identified accordingly."*

Cited from [4]

### 1.1.2. Development of the software based on IEC 62304

The field of application of IEC 62304, which is implemented in Austria as OVE EN 62304, includes the following [6]:

> *"This standard applies to the development and maintenance of MEDICAL DEVICE SOFTWARE when software is itself a MEDICAL DEVICE or when software is an embedded or integral part of the final MEDICAL DEVICE."*

In this context, the development of this application does not fall within the scope of IEC 62304, since its functionality only covers the recording, implementation and documentation of the STK of medical electrical devices and is neither a medical device nor an integral part of a medical device.

## 1.2. Recurrent testing at the PMG

As already stated, the STK of medical devices previously was performed hand-written at the HCE through the working group PMG at the Graz University of Technology with prefabricated test protocol templates in DOCX format. These protocols differ when it comes to the following active medical device types:

- defibrillator
- laser device
- current stimulation device
- high-frequency surgical unit
- infusion pump
- nursing/sick bed
- electromedical device, which does not fall under the types mentioned above

Each of these device-related testing protocols were available in two versions and only differed according to the safety parameters of the IEC 62353 and the IEC 60601 standards (see chapter 1.1.1.1). Due to their digitization and summary in an overall report, the protocols resulted in a great deal of administrative work. To reduce the workload, a software tailored to the work of PMG needed to be developed.

## 1.3. Aims of the thesis

The main task of this master's thesis is the development of a software with which the STKs of active medical devices fundamentally can be carried out and whose design represents an expandable and maintainable basis for further development. The application is intended to be used at the HCE by the PMG. Due to the fact that it was developed from scratch, some goals had to be considered:

- development of a solid and expandable software architecture
- design of a Graphical User Interface (GUI) with consideration of user wishes and feedback
- taking care of data persistence with particular attention to further software development
- capturing of STKs of active medical devices
- export of STK documentations

The realization should be done as far as possible with freeware licenses and form the basis for the following future extensions:

- documentation of captured Recurrent Metrological Test (MTK)s
- performance of STKs for medical electrical systems
- changeability of the documentation of recurrent tests
- export of overall inspection reports
- analysis of safety parameter trends
- control and measurement by means of safety testers and import of measurement data

# 2. Concepts

This chapter describes the requirements regarding the development of the software for the documentation of the STK of medical devices. It covers the various conceptual design steps regarding the GUI and the software design based on those requirements.

## 2.1. Development requirements

The very first step in the concept phase was a requirements analysis to gather the needed information for developing the application in meetings with staff members of the PMG. Furthermore, the understanding of the recurrent testing at the PMG was crucial. Some requirements have already been mentioned in chapter 1.3 as part of the goals of this thesis.

There were no special requirements regarding the operating system on which the software should run. It should just run on a desktop and be operated classically with mouse and keyboard or alternatively via touchscreen. As there is often no internet connection in the health care facilities in which the STKs of their active medical devices are carried out, data persistence should be done locally, but in such a way that it can be accessed even without the new application.

The device lists of the operators, i.e. customers of the PMG, which have to undergo the recurrent testing, should be provided as CSV files and should be read or modified by this software. The data format shall hold the following information of a device:

- type of medical device
- medical area in which the device is operated
- model type
- manufacturer
- serial number
- inventory number
- test interval
- the state of whether the device is being actively used or currently inoperative (device can be reactivated any time)

This information is to be automatically adopted in STKs derived from the devices.

Due to the huge amount of work coming with the design and development of a wholesome text editor with table drawing and image handling support, the STK protocols are provided as editable DOCX templates. They are provided by the PMG for all of the active medical devices listed in chapter 1.2 and should act as the base for capturing STKs. This was not part of the requirements from the very beginning. The only condition was that the STK documentations exported by this application should be fully formatted like the existing recurrent test protocol templates mentioned in chapter 1.2. Another requirement was the editability of the filled STK protocols through the application after their export - which is also intuitively possible for the users thanks to that solution.

## 2.2. Framework and programming language

The chosen framework for the application is the C++ development framework Qt which offers tools for cross development of applications for all major 32-bit and 64-bit desktop and most mobile or embedded platforms. It is mainly used for developing GUI and multi-platform applications. Non-GUI programs can be developed as well, such as server-client applications. The Qt Application Programming Interface (API) offers a variety of features when working with

SQL databases, XML and JSON files, thread management and network support. The framework itself is written in C++ and supports this language in addition to Python and QML for coding. The declarative scripting language QML is provided by Qt Quick, is used for creating GUI and allows using JavaScript for inline logic. Qt also supports a variety of compilers including Minimalist GNU for Windows (MinGW) and Microsoft Visual C++ compiler on Microsoft Windows. MinGW imports the GNU Compiler Collection (GCC) to Windows for C and C++ and offers toolchains for both 32-bit and 64-bit. The Qt framework provides *qmake*, a cross-platform build script generation tool that automates the generation of Makefiles (based on the information in a project file) and Microsoft Visual Studio projects. The Qt framework furthermore provides optional modules like a built-in virtual keyboard. It also comes with its Integrated Development Environment (IDE) - called Qt Creator. Qt is available under a commercial licences or the following open-source licenses: GNU General Public License (GPL) version 2 and version 3, as well as the GNU Lesser General Public License (LGPL) version 3. [7]

With the initial thought of developing the application in Ubuntu and due to the comprehensive capabilities of the Qt framework of creating GUIs with QML as front end and implementing the back end logic with C++, the Qt Creator IDE was chosen as the development environment. The idea regarding Ubuntu was soon discarded in the process due to the decision of exporting the captured STKs into the DOCX format. The format was chosen because of the above mentioned desire for editable STK documentations and because the protocol templates provided by the PMG have been in the DOCX format before. Although one of the requirements for the development was the use of freeware licenses as far as possible, it was allowed to work with Microsoft Word - since the HCE and the PMG already have the necessary licenses. All of these considerations eventually led to the development of the application on Windows 10, initially using the Qt framework version 5.12, which was later updated to version 5.13.1.

### 2.2.1. Signals & Slots

A very important feature of Qt worth mentioning is the "Signals & Slots" mechanism. It is generally used when an object changes in such a way that it can be important to any client. Signals can be connected to slots and even to other signals - there is no limitation regarding the connection count to signals or slots. Communication between signals and slots is type-safe because they need matching signatures (although slots can ignore unnecessary arguments). The Signal & Slot system is available to all classes that inherit from *QObject* or its subclasses. Signals can be emitted publically from anywhere, not only from its defining class or its subclasses. After transmission, all connected signals or slots are normally called up immediately (however, this can be changed). Slots are not limited to their connection with signals. They are treated as normal C++ functions and can be therefore invoked conventionally - and can even be declared as *virtual*. The only difference is that they can react to connected signals. [8]

### 2.2.2. QML and C++ integration

Another important aspect when working with Qt is the C++ and QML integration. QML can be extended through C++ [9]. The following attributes of *QObject*-derived classes can be accessed from QML [10]:

- properties - through the *Q_PROPERTY* macro
- methods - if they are public slots or marked with the *Q_INVOKABLE* flag
- signals
- enumerations - if declared with the *Q_ENUMS* macro

A C++ class that is derived from *QObject* can be made available to QML as usable data type through several registration options depending on the type of use - e.g. whether it is a singleton or whether it should be instantiable in QML or not. [9]

Care should be taken when passing data from C++ to QML regarding ownership, since QML uses a garbage collector and C++ does not. The ownership remains with C++ when data is passed over through property. QML also does not delete obsolete *QObject* instances that have a parent set. However, when data is passed over a C++ method, the ownership goes to the QML engine, which results in deletion by the garbage collector when it is no longer needed. This can be prevented by explicitly setting the ownership by invoking *QQmlEngine::setObjectOwnership()* with *QQmlEngine::CppOwnership* specified. [11]

## 2.3. STK protocol templates in DOCX format

To achieve the requirements, Microsoft Word is used to load the needed information of the STKs provided as DOCX templates by the PMG. All of these templates are provided with UTF-8 encoding, so even special characters like Greek letters can be displayed in the GUI. These templates form the basis not only for all the information required to be displayed to the user, but also for the style and structure of the filled STKs that are exported after recurrent tests have been carried out.

### 2.3.1. Template structure for STKs

To properly rebuild the process of capturing STKs and the protocol structure itself, the provided DOCX templates - also shown in the appendix - have been analyzed carefully. The protocols all have the same layout, consisting of the PMG emblem with its logo and four tables. In order to further illustrate the most important components of the templates, the individual tables are now described.

The first table - which is located on the right-hand side of the emblem - is for assignment purposes of the device that undergoes the STK and therefore includes the following:

- the reference number
- the name of the customer, i.e. the device owner
- the medical area in which the device is used
- the inventory number

Each of these cells is a combination of a label and a cell for filling in the asked information. The second table lies underneath the PMGs logo and holds general information of the device:

- the device type
- the product type
- the serial number
- the manufacturer
- the equipment of the device - There might be preset options with check-boxes, that are typical for a special device type, e.g. defibrillator electrodes
- the date of performing the STK
- the name of the inspector performing the STK

Like the first table, these cells - except the ones for the equipment - consist of a label and a cell for filling in the data. The third table is located right next to the second one and provides information on the following device-related specifications:

- the inspection interval
- the application part
- the protection class
- the moisture protection
- the explosion protection

Each of these specifications has its own row with a label and several options where one has to be checked. The gray shaded options, either empty or with text, cannot be checked. The centerpiece of such a STK protocol is the fourth table, which in turn is split into the following three parts according to the scope of recurrent testing mentioned in chapter 1.1.1.1:

- Visual Inspection
- Electrical safety parameters
- Function

All three parts contain several test points, i.e. "inspection points", where each has its own row of different cells. The first cell contains the description of the test point - hereinafter referred to as "description cell". It is followed by five cells that serve as rating system of the inspection point at which one has to be checked:

- "not applicable" - the inspection point is not applicable to the device
- "OK" - the test point is correct or fulfilled
- "deficiency level 1" - tolerable error
- "deficiency level 2" - has to be resolved in the medium term
- "deficiency level 3" - the defect is dangerous for the patient or the user of the device

Those cells are followed by a result cell and a comment cell that can be filled in by the user. This appearance of an inspection point is the most basic one, whereas the description cell and the result cell can appear further divided later on. The description cell always consists of a descriptive part which is followed by:

- one information cell or
- one cell with two checkboxes plus their description and one information cell or
- one information cell and one cell for user input or
- two information cells or
- a second description cell and one information cell or
- a second description cell, one cell for user input, one unit cell and one information cell

Within the description cell, the order of each of these cell combinations is always the same from left to right. Information cells always have a grey background color and result cells can have a further unit cell. All of the cells are treated as

read-only when it comes to the application, except for the ones reserved for user input.

The last row of this table is reserved for the overall device status evaluation and the signature of the inspector. The device status has nearly the same rating system as the test points; only the not-applicable-cell is missing.

## 2.4. UI/UX design

The design of the GUI was the next step in the development of the application. That included a wholesome UX design of navigation and general interactions with the user. The basic principles were provided by Steve Krug's book *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability*. Although the title states that it approaches web usability, this book also covers usability principles for (mobile) applications. The screen design should therefore cover the following aspects [12]:

"• *Take advantage of conventions*
• *Create effective visual hierarchies*
• *Break pages up into clearly defined areas*
• *Make it obvious what's clickable*
• *Eliminate distractions*
• *Format content to support scanning*"

The drafts for the various screens - as well as all icons throughout the application - were drawn and designed using the free and open-source vector graphic editor Inkscape. The UI/UX design was planned in consultation with the employees of the PMG in several design meetings, resulting in an agile approach. The first decent design was implemented as a click dummy with test data in QML, which was used to test the design of the GUI with the navigation to certain areas and functionalities of the application. The purpose was to verify the basic correctness of the understanding regarding the capturing of the STKs and the

environment in order to make this possible. This served as vital starting point for the subsequent software design.

## 2.4.1. Screen building blocks

To make the GUI easily understandable, two basic page types were designed: menus and content pages. Content pages generally have a title and navigation bar. The content lies between them, providing clear and recurrent visual hierarchies as supposed by Steve Krug [12]. This enables the user to orientate himself quickly. Menus (except the main menu) just include a navigation bar. Each button mentioned in the following subsections comes with a custom made optical feedback when pressed.

### 2.4.1.1. Navigation bar

Every screen except the main menu has a navigation bar at the bottom as displayed by 2.1:

Figure 2.1.: Navigation bar

The navigation bar contains up to three buttons. The Back button on the left is for navigating back to the previous screen. The Home button in the middle of the navigation bar immediately returns the user to the main menu. These two buttons are essential for good navigation [12]. The Check button on the right is for confirming one or more selections or for progressing to the next screen. This button is not provided on every screen and sometimes is invisible until a selection is made.

#### 2.4.1.2. Title bar

All screens except for the menus contain a header bar with at least one of the following items:

- screen title
- search button (optional)
- Add or Remove/Reactivation button (both optional, these two buttons share the same position, so only one is displayed at a time)

Figure 2.2 displays (from left to right) the screen title, the Search button and the Add button. Figure 2.3 shows another version of the same header bar. The only difference is that there is a Remove instead of an Add button.


Figure 2.2.: Title bar with Add button


Figure 2.3.: Title bar with Remove button

The Add button is used when creating new inspectors, customers, devices, inspections or device STKs. The Remove/Reactivation button is used to remove selected inspectors, customers, devices, inspections or device STKs. Also, device STKs that have already been completed can be made editable again by changing its status from finished to unfinished.

When the Search button is pressed, the title bar changes its appearance to a search bar as shown in figure 2.4.

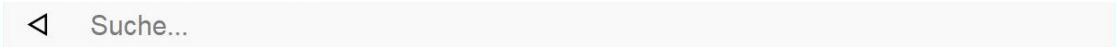


Figure 2.4.: Search bar

The search bar has a Back button on the left that ends the search function and changes back to the title bar when pressed. The Remove button is being displayed on the right when a search string is typed in by the user, as showed in

figure 2.5. Pressing the Remove button deletes the entered search string, which also resets the displayed options of the browsed content.


Figure 2.5.: Search bar with search string

#### 2.4.1.3. Dialogs

Dialogs are the main tool for providing information on functionalities when user decisions or inputs are necessary. Such inputs cover e.g. the naming of a customer or selecting his device list in CSV file format during the creation process.

### 2.4.2. Font and colors

The UI/UX design also involved the definition of the used font and the colors. The only used font in the whole GUI is "Roboto".

Table 2.1 displays general colors that are used on every screen.

| **Description** | **HTML color code** |
|---|---|
| Color for optical feedback, item selection and the border of menu buttons | #e6e6e6 |
| Background color for menu buttons, title bars and navigation bars | #f9f9f9 |

Table 2.1.: General color definitions

Table 2.2 lists all the colors that are used on screens that serve the actual capturing of STKs.

| Description | HTML color code |
|---|---|
| Color for table borders | #cccccc |
| Highlighting color of incomplete inspection points | #ff0000 |
| Background color of information cells | #c8beb7 |
| Background color of description cells | #e3f4d7 |

Table 2.2.: Colors used on screens for capturing STKs

### 2.4.3. GUI structure

This subchapter briefly describes the content structure of the GUI in a simple way in order to provide a basic overview. The German user manual of the application contains detailed information on the structure and gives an overview of the functionalities - it can be found in the appendix.

The content structure of the GUI is divided into three separate areas (see the following subchapters) that provide different functionalities and are accessible via the main menu:

- Management
- Inspections
- Reports

#### 2.4.3.1. Management

The management area is in turn divided into three subparts for managing inspectors as well as customers or functionality to convert the DOCX STK templates mentioned above into internal representation of the software.

The subsection about inspectors is for displaying, adding and deleting them. These options can be chosen when device STKs are being carried out.

Managing customers also involves adding and deleting them. When adding customers, their device list must be added in the form of a CSV file (see chapter 2.1) that contains a basic set of information about each particular device.

Furthermore, new devices can be added to the device list or sorted out. The active and sorted out devices are displayed on different screens.

#### 2.4.3.2. Inspections

This area is for managing inspections. If this section is being entered, all inspections that have already been captured but not yet completed are displayed. Pressing the Add button starts a process in which the customer must first be chosen and then all devices must be selected (at least one) that have to go through the STK. Captured inspections can also be deleted.

By selecting an inspection, the finished and unfinished device STKs can be viewed in separate areas. Finished device STKs are read-only by default, but can be made editable by pressing the Reactivation button.

The STK protocol templates provided by the PMG for the different active medical devices were checked for similarities so that the work processes of capturing STKs could be abstracted. To have clearly defined areas, as supposed by Steven Krug in his book *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability* [12], the procedure of the tests is split up into four parts - which is also reflected in the screens of the application:

- General information
- Visual inspection
- Electrical safety parameters
- Functional test

If any information is missing on these screens, a dialog appears which indicates missing information to the user. With the exception of the general information screen, all unfinished inspection points are highlighted when the check button is pressed.

An inspection can be finished when all device STKs are completed. Finished inspections are moved to the reports section of the program and are then available for export as DOCX STK documentations.

#### 2.4.3.3. Reports

The main functionality of this section is to browse through the inspections that are available for export. After selecting the customer and the desired inspection, the device STKs can be chosen for the creation of their documentations in DOCX format. It is possible to select all or just a subset of all available device STKs of that inspection to be exported.

## 2.5. Domain-Driven Design

After the workflows and processes for creating and capturing STKs were expressed through the GUIs design, the next step was to define the needed core classes of the application. The chosen method for this step was the Domain Driven Design (DDD).

DDD was introduced by Eric Evans in *Domain-Driven Design: Tackling Complexity in the Heart of Software* [13]. Millet and Tune's book with the title *Patterns, Principles, and Practices of Domain-Driven Design* is based on the concepts introduced by Evans and offers valuable practical examples and explanations [14]. Both are important references for this topic.

DDD is a development philosophy that focuses on bringing together a real-world problem, process or system, in other words, a bounded context, into an object-oriented design code model, i.e. the Domain Model (DM). This approach relies on the close cooperation of domain experts and developers. For this purpose, the introduction of a Ubiquitous Language (UL) is important, because it enhances communication by building up a common vocabulary between the field experts and the developer. [13, 14]

The UL in this case was cultivated through the close collaboration of the employees at the PMG and the author of this thesis, resulting in an analysis model. The DM was derived from this model and includes the necessary classes for the business use cases for capturing STKs.

Evans [13] also supposed the following building blocks for object oriented domain modeling:

- Entities
- Value objects
- Aggregates
- Services
- Modules
- Repositories
- Factories

Regarding these building blocks, Millet and Tune state the following in their book *Patterns, Principles and Practices of Domain-Driven Design* [14]:

> *"Mapping the implementation model back to the analysis model and ensuring they are bound to one another is hard. To guide developers and clarify designs, Evans has built upon the domain model pattern that was first catalogued in Martin Fowler's book Patterns of Enterprise Application Architecture. He introduces a pattern language containing a number of building block patterns to enable the creation of effective domain models. The patterns, built around best-practice, object-oriented techniques, are sometimes referred to as the tactical patterns of Domain-Driven Design (DDD)."*

The main reason for using DDD was to gain an understanding of the domain concerning the execution and capturing of STKs, and therefore to create a basic domain model by analyzing the workflows. DDD offers very extensive concepts for this purpose, but applying all the above-mentioned building blocks in detail would have required a quite high workload and would have resulted in overengineering, which would have gone beyond the scope of the given requirements. For this reason, modules and services are not covered in this thesis. There may be some classes that may offer service characteristics, but were not designated as such when designed, whether it was an application, an infrastructure or a domain service [13, 14].

"Entities" in this context are objects which are defined by their identity and not by their attributes. This means that entities with similar attributes are still treated as different. In contrast to that, value objects have no identity. So if e.g. two "value objects" have identical attributes, they are treated equally. "Aggregates" function as collections of entities, value objects or even other aggregates. An aggregate's root entity is often referred to as "aggregate root". [13, 14]

The "Repository pattern" takes care of saving and loading data, typically to or from a database [13, 14]. Millett and Tune suggest using a simpler Data Access Object (DAO) over a Repository pattern if the domain model is not rich, because implementing a Repository pattern is a very complex task [14].

The two main types of the "Factory pattern" are explained in chapters 2.7.1.1 and 2.7.1.3.

### 2.5.1. Domain model

The present domain model was derived from the analysis model, which is based on the evaluation of the STK protocol templates and also the knowledge of those work processes that were acquired through the design of the GUI in consultation with the domain experts of the PMG. It forms the heart of the application and was the foundation for the rest of the software design. The following explanation of the domain model is split into two parts for better readability and is displayed in the figures 2.6 and 2.7. The UML diagrams are kept simple for better understanding and only show the needed classes and their associations, without going into depth of listing all of their attributes and operations. All class names were chosen by the author in such a way that they are as meaningful and self-explanatory as possible. When it comes to occurrences of the *DeviceInspection* naming, the reason of not choosing e.g. the name "*DeviceTest*" was to create distance from the terms "test" or "testing" in the context of software development. The same goes for all occurrences of *InspectionPoint*.

Figure 2.6 displays the UML class diagram of the domain model. A *Customer* entity, which acts as aggregate root, owns the containers *DeviceContainer* and *InspectionContainer*, one of each type. These containers cannot exist without a *Customer*. The *DeviceContainer* holds *0* to * *Device* entities, whereas the latter cannot exist without the former, resulting in a composition. The *InspectionContainer* contains *0* to * *Inspection* entities, which cannot exist on their own. The *Inspection* entity owns a *DeviceInspectionContainer* that contains *1* to * *DeviceInspection* entities, acting as aggregate. An *Inspection* cannot be created without a single *DeviceInspection*. There is also a *DeviceInspectionTemplateContainer* that can hold *0* to * *DeviceInspection* entities, representing the STK templates mentioned above. *DeviceInspection* objects also cannot exist on their own and must be part of either a *DeviceInspectionTemplateContainer* or a *DeviceInspectionContainer*. A *Device* owns one *GeneralInformationDevice*, whereby the latter cannot exist without the former. The same goes for *DeviceInspection* and *GeneralInformationDeviceInspection*. This leads to an indirect association between *Device* and *DeviceInspection*, since *GeneralInformationDevice* is inherited from *GeneralInformationDeviceInspection*. This is important because when a *DeviceInspection* is created by the user, it receives some general information from its corresponding *Device*. These are represented by the members of the *GeneralInformationDevice* entity that correspond to the device information in CSV files mentioned in chapter 2.1, except the state whether the device is being actively used or currently inoperative. There is also an *InspectorContainer* holding *0* to * *Inspector* entities. The *Inspectors* represent those who are performing a STK of a medical device with their names recorded.

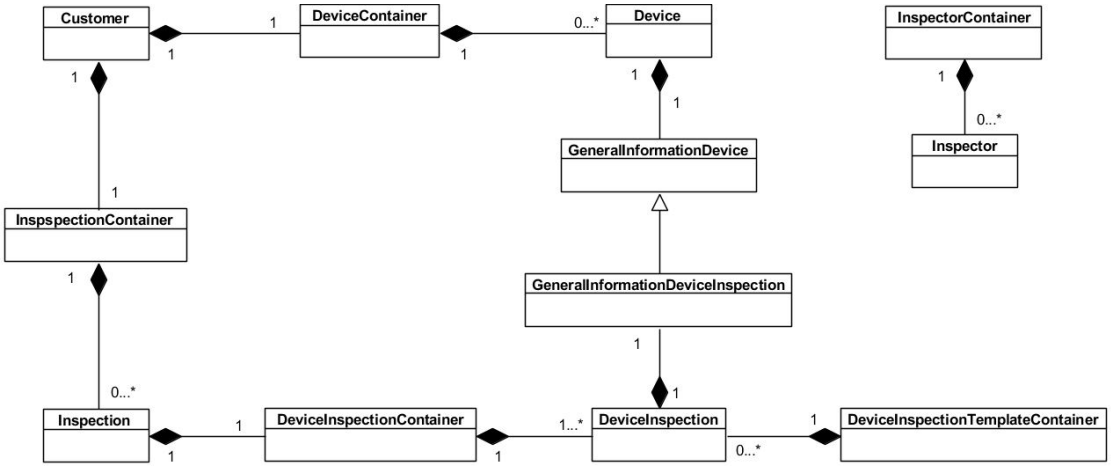


Figure 2.6.: Domain model

Figure 2.7 shows the *DeviceInspection* aggregate with all its related entities. The *DeviceInspection* entity, as well as the *GeneralInformationDeviceInspection* entity are the same as displayed in figure 2.6. The *DeviceInspection* entity owns one *ProtectionEntries* entity that cannot exist on its own. These *ProtectionEntries* cover the application part, protection class as well as moisture and explosion protection of the active medical device mentioned in chapter 2.3.1. The *DeviceInspection* entity has currently one derivative: the *DeviceInspectionSTK* class that represents a STK of an active medical device. In the future there could be additional derivatives, one of which could represent MTK. A *DeviceInspectionSTK* holds *o* to * *InspectionPoint* entities. The *InspectionPoint* entity has the most complex class hierarchy with eight derivatives representing the different test point types mentioned in chapter 2.3.1. The *InspectionPoint* class represents the basic type occurring in the DOCX STK templates. Its derivatives represent more complex types:

- *InspectionPointInfo*
  - description cell contains also an info cell
- *InspectionPointInfoCheckbox*
  - description cell has also a checkbox cell followed by an info cell
- *InspectionPointInfoUserData*

  - description cell has also an info cell followed by a cell for user input

- *InspectionPointInfoUnit*
  - similar description cell as *InspectionPointInfo*
  - result cell has an additional unit cell
- *InspectionPointInfoCheckboxUnit*
  - similar description cell as *InspectionPointInfoCheckbox*
  - similar unit cell as *InspectionPointInfoUnit*
- *InspectionPointInfoSpeacialUnit*
  - description cell has a second description cell followed by an info cell
  - similar unit cell as *InspectionPointInfoUnit*
- *InspectionPointInfoSpeacialAdvancedUnit*
  - description cell has a second description cell followed by a cell for user input, a unit cell and an info cell
  - similar unit cell as *InspectionPointInfoUnit*

The *Checkbox* entity can only exist as part of a *DeviceInspection*, *InspectionLineInfoCheckbox* or *InspectionLineInfoCheckboxUnit*. A *DeviceInspection* can hold *o* to * *Checkboxes*. Both *InspectionLineInfoCheckbox* and *InspectionLineInfoCheckboxUnit* hold two *Checkboxes* each.

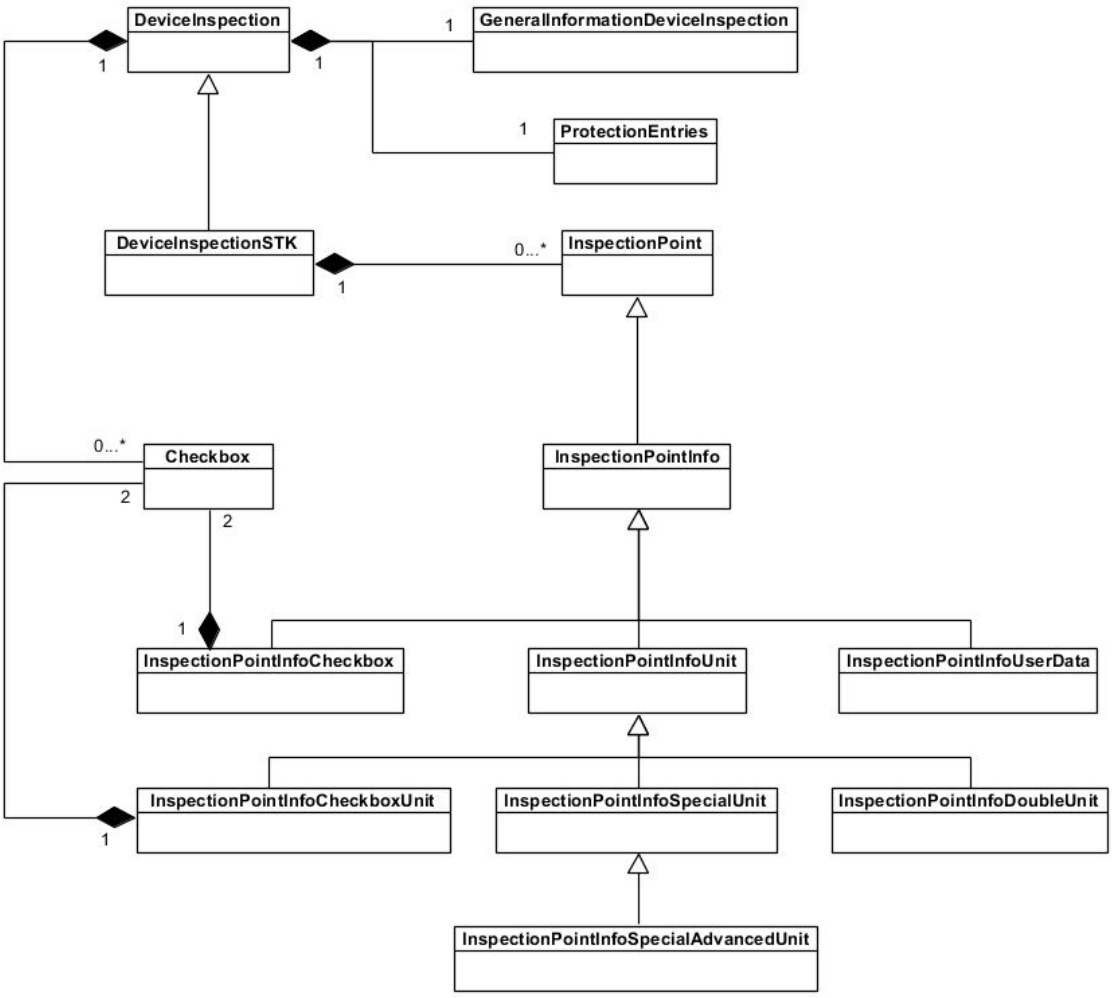


Figure 2.7.: *DeviceInspection* aggregate class diagram

## 2.6. Dependency Injection

Van Deursen and Seeman state in their book *Dependency Injection Principles, Practices, and Patterns* Dependency Injection (DI) that "Dependency Injection is a set of software design principles and patterns that enables you to develop loosely coupled code" [15].

Furthermore, the following statement points out the importance of DI when it comes to this application [15]:

> *"DI isn't a goal in itself; rather, it's a means to an end. Ultimately, the purpose of most programming techniques is to deliver working software as efficiently as possible. One aspect of that is to write maintainable code."*

The major idea behind DI is that a client does not instantiate a needed concrete class itself, but it rather gets an interface, i.e. an abstraction. This behavior of external configuration leads to loosely coupled code. A DAO (see chapter 2.9.1) for example depends on a persistence class that communicates with a database or file system. If that DAO instantiates that class itself, it results in the tight coupling, because the DAO depends on a concrete implementation. When the requirements for the persistence change, the code has to be modified because of the dependency change, which makes the code less maintainable. Loose coupling makes code extensible, which also leads to maintainability. Another important aspect of DI is that it allows late binding, which means that classes or modules can be replaced without having to recompile the code (although this is negligible with this software, but it is worth mentioning). This is made possible by the use of DI through injecting abstractions, following the Liskov Substitution Principle (see chapter 2.6.4). DI also promotes testability because the dependencies can easily be swapped for implementations that serve test purposes. [15]

When it comes to this application, it is also important to differ between "stable" and "volatile" dependencies, which Van Deursen and Seeman [15] do as follows:

> "• *DEPENDENCIES are considered STABLE in the case that they're already available, have deterministic behavior, don't require a setup runtime environment (such as a relational database), and don't need to be replaced, wrapped, or intercepted.*
> • *DEPENDENCIES are considered VOLATILE when they are under development, aren't always available on all development machines, contain nondeterministic behavior, or need to be replaced, wrapped, or intercepted."*

### 2.6.1. Types of Dependency Injection

The following subchapters deal with the three types of DI explained by Van Deursen and Seeman [15]:

- Constructor injection
- Method injection
- Property injection

#### 2.6.1.1. Constructor injection

In order to implement "Constructor injection", a class has to make a public constructor available in which all required dependencies are listed statically in the signature. These dependencies are mandatory for the class, adding them to the constructor guarantees the injection. This is the type of injection that should be used as a default. [15]

#### 2.6.1.2. Method injection

"Method injection" is being used when either the consumer of the injected dependency or the injected dependency itself varies on each call. The consumer offers a method for this purpose in which the dependency is injected via a parameter. [15]

#### 2.6.1.3. Property injection

"Property injection" is best used when a dependency is optional, so that it can be changed after the consuming class has already been instantiated. Therefore the class exposes a public writable property of the dependency's type. [15]

### 2.6.2. Composition Root pattern

The Composition Root (CROOT) "is a single, logical location in an application where modules are composed together" [15] and should be located as close as possible to the entry point of an application, i.e. the *Main* method. It serves the purpose of composing object graphs. CROOT acts as an application infrastructure component and can either be implemented with "Pure DI" or even with a tool called "DI container".

### 2.6.3. Ways of implementing Dependency Injection

Van Deursen and Seeman explain in a very detailed way how DI is implemented as Pure DI or with a DI container. The latter automates tasks that are important when using DI like object composition, lifetime management and interception. Its purpose is to automatically create registered dependencies based on the request and inject them where they are required. Pure DI is defined as applying DI without the use of such a DI container - so the object graphs are being composed together by hand. [15]

The authors [15] define "Interception" as follows:

> *"INTERCEPTION is the ability to intercept calls between two collaborating components in such a way that you can enrich or change the behavior of the DEPENDENCY without the need to change the two collaborators themselves."*

The use of a DI container would be the preferred option because of the previously mentioned features. Unfortunately, the Qt framework does not offer a library for DI container or similar functionality.

There are some outdated custom made options by users, with *injeqt* being one of the latest that could be found in the course of the investigation. The current release was on 04-21-2017. [16]

The enormous workload associated with implementing a self-made DI container for this application would go beyond the limits. Due to the rather straightforward requirements and not-needed complexity of the application (see chapter 2.1), the use of the more simple Pure DI is perfectly adequate. So the object graphs can be configured without the need of automation. Even if the requirements regarding e.g the data persistence will change to use databases over file systems, this is perfectly covered.

### 2.6.4. SOLID design principle

Van Deursen and Seeman point out in their book *Dependency Injection Principles, Practices, and Patterns* the "SOLID" software design principle [15], which is an acronym made up of the following principles:

- **S**ingle Responsibility Principle - classes should have only one single responsibility
- **O**pen-Closed Principle - classes should be open to extension but closed to modification

- **L**iskov Substitution Principle - a consumer should be able to use any implementation of an abstraction without breaking the correctness of a system
- **I**nterface Segregation Principle - a client should not be forced to depend on methods it doesn't use
- **D**ependency Inversion Principle - higher and lower level modules should depend on abstractions

These principles offer guidelines and goals for writing clean code, enhancing the maintainability of an application [15]. Since the SOLID principle is a very commonly known design concept when it comes to object-oriented design (not only in the scope of DI) and is covered a lot in the relevant literature and online resources, a detailed description of its concepts is not necessary - and therefore not a part of this thesis.

## 2.7. Gang of Four design patterns

The book *Design Patterns: Elements of Reusable Object-Oriented Software* written by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides [17], often referred to as Gang of Four (GOF), is a software engineering book describing 23 classic software design patterns. Object-oriented software design patterns are general and reusable solutions to recurrent issues in software design. They act as descriptive templates for how to solve a particular problem at its core. The implementation of such design patterns must be adapted to the respective context of the software or problem.

The following subchapter explains the five creational patterns introduced in *Design Patterns: Elements of Reusable Object-Oriented Software* because the creational processes of objects make up an important part of the development of this application. These five patterns provide various creational mechanisms for objects to enhance flexibility and enable the reusability of existing code, rather than instantiating objects directly. [17]

### 2.7.1. Creational design patterns

The creational patterns include the following design patterns [17]:

- Abstract factory
- Builder
- Factory method
- Prototype
- Singleton

Although not all of these creational patterns have been implemented, each is briefly described in the following subchapters. This serves the purpose of giving an overview of the applications, methods and conclusions when creating objects.

#### 2.7.1.1. Abstract factory

The "Abstract factory" is about defining an interface, i.e. an API, that creates families of associated objects that are intended to be utilized together. Its interface exposes multiple methods, each of which creates a different object. The objects created are common types like abstract classes or interfaces and not their concrete implementations. Which concrete implementation of the abstract factory is used depends on the system, but only one of the several families of products should be configured. [17]

The Abstract factory encapsulates the product creation process and returns its common interface to the client. This means that the concrete class is isolated within the "Concrete factory". This design pattern also promotes compatibility among products because all product objects that form a family are intended to work together. An application must use objects from only one family at a time, which is enforced by using one concrete implementation of the Abstract factory interface. It is easy to exchange product families by simply changing the Concrete factory at the location of its instantiation. One downside of implementing this pattern is the difficulty of supporting new kinds of products, because the

abstract factory interface needs to be extended, which leads to a change of all its subclasses. [17]

Figure 2.8 shows the structure of the participating classes of the Abstract factory design pattern:

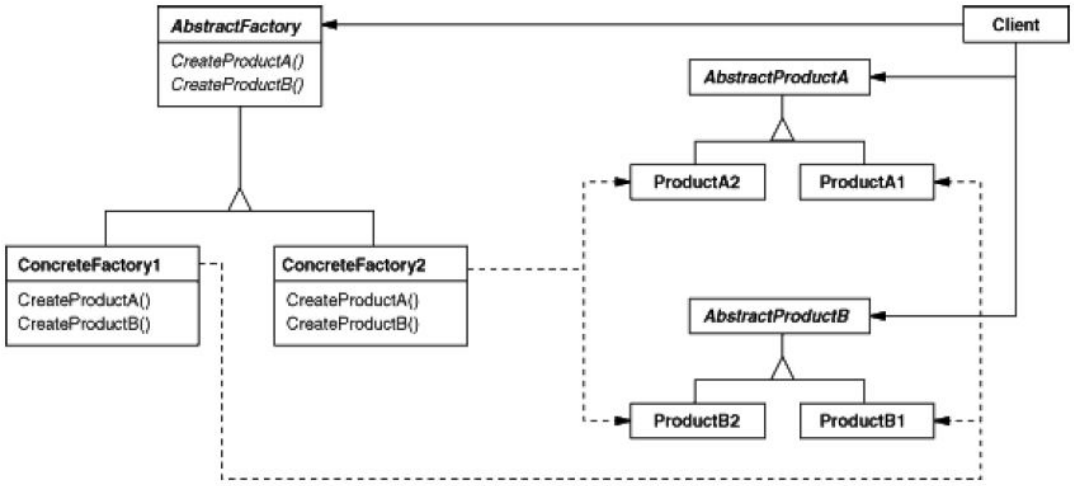


Figure 2.8.: Abstract factory structure [17]

The use of this pattern does not provide a huge benefit for this application for the given requirements (see chapter 2.1), since there are no related objects that from today's perspective can be used as a product family or replaced by another in the future. The "Factory method pattern" (see chapter 2.7.1.3) offers a better solution to the scope of this software in terms of creating objects.

#### 2.7.1.2. Builder

The "Builder pattern" is responsible for building a complex object by assembling its parts, which are simple objects. It is possible to create different types and representations of an object by using the same construction code. This pattern involves a director object that is configured with a concrete builder object. The product assembly happens step-by-step and is controlled by the director. [17]

Figure 2.9 displays the class diagram of the Builder design pattern with its participating classes:

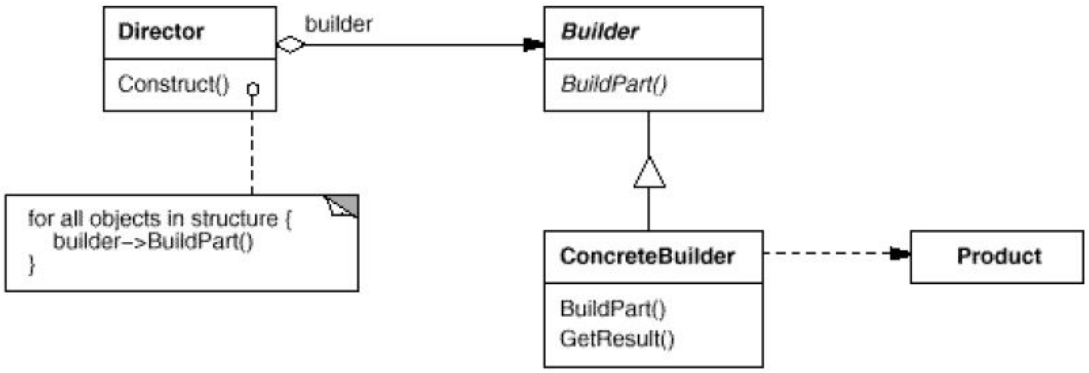


Figure 2.9.: Builder structure [17]

There are several consequences when implementing the Builder pattern. It provides the freedom to have many combinations of parts while isolating the complex construction of the product from the rest of the application, as only the finished product is retrieved from the builder. This can happen through the director or the builder itself. This leads to the decoupling of the created object and the actual construction. Another aspect is the finer control of the construction process through the gradual assembly. [17]

The Builder pattern is a good way to assemble complex objects. For this application, however, it does not offer a great advantage in the form displayed in figure 2.9. The *DeviceInspection* aggregate object graph, which is by far the most complex object as displayed in figure 2.7, is loaded directly from a STK protocol template that represents it, making a builder negligible. In general there are no object graphs that would justify the effort of implementing the builder pattern in the way described in this chapter - just to have used it. Of course, simple building steps had to be used, but not in the way this pattern works.

#### 2.7.1.3. Factory method

The Factory method pattern describes a method, which is exposed by an interface, i.e. an API, for creating a derived object of an abstract class, which serves as the return value. This is useful when a class can not anticipate the class of objects it needs to create, so its subclasses can specify it. There are two major variations of the Factory method pattern: One option is an abstract creator, where subclasses have to define the factory implementation. The other is a concrete creator that defines a default implementation of the method that can be overridden by subclasses. Factory methods can also be parameterized, which enables the creation of multiple kinds of objects that share the same interface. Which object is to be created depends on the given parameter. [17]

Figure 2.10 displays the structure of the Factory method design pattern:

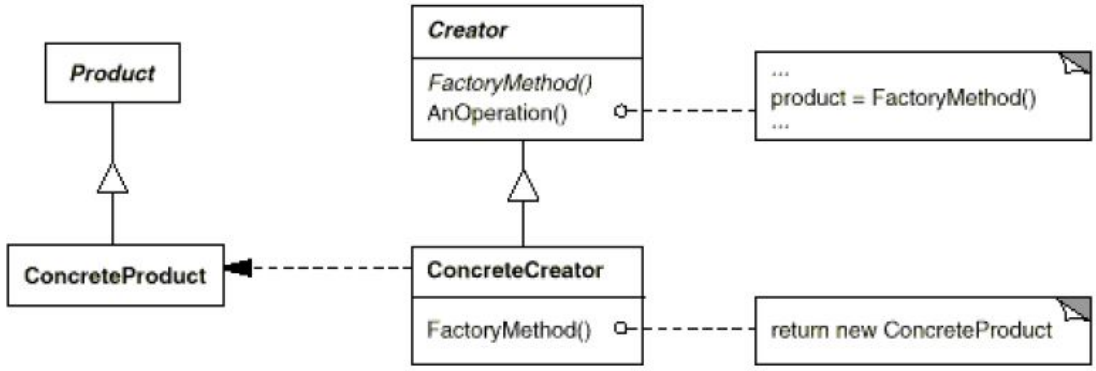


Figure 2.10.: Factory method structure [17]

The use of this design pattern eliminates the need to bind concrete classes into framework code. This means that the framework only deals with the interface of the created object, so it can work with any subclass. This provides more flexibility than creating a subclass directly by using the *new* operator, which leads to lesser dependencies because the creation of products happens in one place in the codebase. The Factory method pattern also connects parallel class hierarchies through e.g. returning an inherited concrete product based on a given parameter. [17]

#### 2.7.1.4. Prototype

The "Prototype pattern" involves implementing a cloning method that creates a duplicate of a prototype instance. This pattern is used to create objects whose classes and properties are not known before runtime. Another reason might be that it is easier to build a set of prototypes and clone them rather than building complex class hierarchies and factories. Avoiding long taking and complex instantiations might be a criterion as well. [17]

Figure 2.11 shows the structure of the participating classes of the Prototype design pattern:

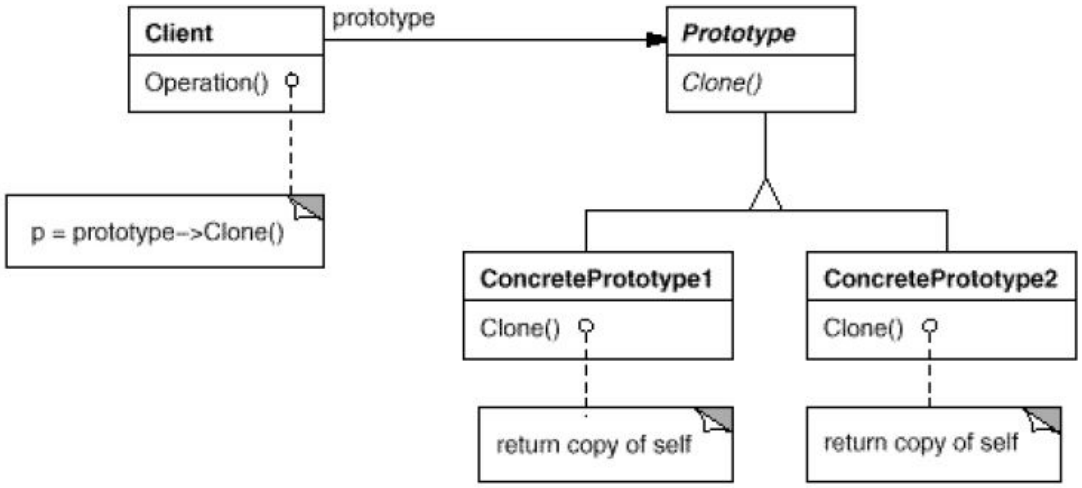


Figure 2.11.: Prototype structure [17]

By applying the Prototype pattern, the concrete object is hidden from the client similar to the Abstract factory and Builder patterns. Products can be created and deleted at runtime and complex initializations are done only when prototypes are built up. The retrieving of the concrete product is easier because there is no need of building complex creator hierarchies. Furthermore, the creation of prototypes is unlimited. New kinds of prototypes can be defined by instantiating existing classes and providing them a such. A downside of this pattern is that each subclass of a prototype must implement a *clone()* method, which might be challenging for already existing classes. It might also be difficult

if classes include member objects that do not support copying or contain circular references. [17]

The Protoype pattern offers a very good opportunity for handling *DeviceInspectionSTK*. The different prototypes are based on the device type and the standard used when it comes to the safety parameters and are loaded from the corresponding DOCX STK templates. If a *DeviceInspectionSTK* is created as part of an *Inspection*, it is being cloned based on its specific prototype.

#### 2.7.1.5. Singleton

The "Singleton pattern" involves a class that is responsible to create a single object while guaranteeing that it is the only instance that can be created. This class provides a way to access its only instance which can be done directly through the static *getInstance()* operation without the need to separately instantiate the object of the class. This ensures that it is not possible to create another instance of the object. The singleton class has to be accessible to clients from a global access point. [17]

Figure 2.12 displays the class diagram of the Singleton design pattern:

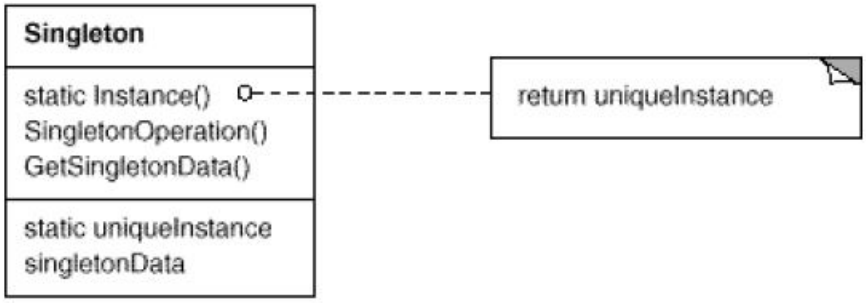


Figure 2.12.: Singleton structure [17]

The use of the Singleton pattern comes with a few consequences. By using this creational design pattern it is ensured that a class has only one instance with a global access point. This is due to the encapsulation of the creation and retrieval of its instance through the *Instance()* method. This approach gives the

singleton class strict control over who accesses its single instance. Furthermore, the singleton class can be subclassed, which means that an object that behaves differently than originally intended can be returned. This also enables the configuration of client applications at runtime by selecting a different subclass. This pattern increases the flexibility compared to using a class with only static members alternatively as well. Pure static members can not be *virtual* in C++, so that overriding by subclasses is not possible. [17]

For the application of this thesis the Singleton pattern is useful because there are classes that should only be instantiated once at runtime - which is ensured by the pattern. But the use of this design pattern can be problematic when it is being used alongside DI, because the static *Instance()* method can be accessed globally. This affects the testability of classes in terms of unit testing, where dependencies should be swapped for implementations that serve testing purposes. However, the ability to intitialize or configure singletons near the applications's entry point provides a way to overcome these issues, because in the case of unit testing the singletons can be initialized with instances that serve testing purposes.

Van Deursen and Seeman [15] state the following about the Singleton pattern when used along DI:

> *"The Singleton pattern should only be used either from within the COMPOSITION ROOT or when the DEPENDENCY is STABLE. On the other hand, when the Singleton pattern is abused to provide the application with global access to a VOLATILE DEPENDENCY, its effects are identical to those of the AMBIENT CONTEXT [...]."*

## 2.8. Software architecture

To provide solid and extensible software architecture, the author of the thesis chose a layered architecture. The main idea of this architecture is to divide the software architecture into separate areas, i.e. layers, with their specific roles.

Fowler explained in his book *Patterns of Enterprise Application Architecture* the following three principal layers [18]:

- Presentation
- Domain
- Data source

Layers were implemented, as shown in figure 2.13, based on Fowler's three principal types of layers:

- Business Logic Layer (BLL)
- Data Access Layer (DAL)
- Presentation layer, i.e. GUI

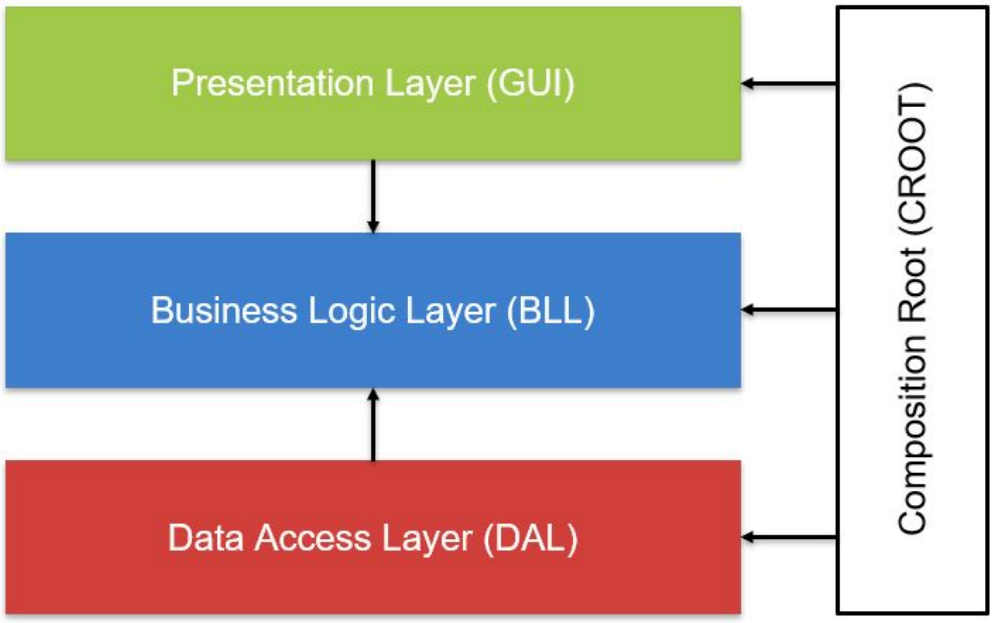


Figure 2.13.: Three Layer Architecture

The arrows in the figure describe dependencies. A layer is dependent on the layer, where the arrow is pointing to. Due to the use of DI and the associated use of the CROOT pattern (see chapter 2.6.2 for the composition of the object graphs, the layer architecture was used based on the example described by Van Deursen and Seeman in their work *Dependency Injection Principles, Practices, and Patterns* [15].

The already discussed works [13, 14, 15] offer further examples of other architectural types, such a hexagonal architecture or the extending of the "Three Layer Architecture", by adding e.g. an application (service) layer. As already mentioned in chapter 2.5, services were not taken into account when designing the application because it still meets the requirements for this thesis. There may be classes with application service characteristics that are not titled as such. This leads to the assumption that an application layer is not required. Having only three layers still provides many advantages while still meeting the given requirements at the same time. Such benefits are scalability, availability and performance. For example, if the persistence requirements change from file systems to databases in the future, the persistence layer can be replaced without to change the other two layers.

### 2.8.1. Business Logic Layer

The BLL, often referred as Domain Layer [18], contains the domain model (see 2.5.1) with its entities and furthermore holds the business logic for the application. To maintain the structure shown in figure 2.13, the BLL is not depending on any other layer.

### 2.8.2. Data Access Layer

The DAL, often referred to as "Data Source Layer" [18], handles the data, which is stored and retrieved from file systems. This application uses XML and CSV file systems (see chapter 2.1) for persisting the data. To achieve this, it has to depend on the BLL.

### 2.8.3. Presentation layer

The "Presentation layer" [18] defines the front end layer, i.e. the GUI, and is the top-most level of the application. The main task of the GUI is to provide

an environment with which the user can interact. This means that tasks and data are visualized in a way that is easy to understand by the user. For these purposes, a dependency on the BLL is inevitable.

## 2.9. Data persistence

As mentioned in the chapter 2.1, the data for the recurrent testing has to be stored locally due to the possible lack of internet connection during the execution of STKs. But in such a way that they are accessible even without the application. Given these requirements, the use of file systems via a local database was chosen for data persistence. As given by the requirements, the device list of customers has to be stored in CSV file format. The rest of the domain model data, except *DeviceInspection* data, (see chapter 2.5.1) is being saved in XML file format. A *DeviceInspection*, its derivative *DeviceInspectionSTK* and their associated objects shown in figure 2.7 are converted into a JSON object and put into the corresponding XML section. In the case of a collection of *DeviceInspection* objects, this is implemented as a comma-separated collection of JSON objects. The reason for this decision is explained in chapter 3.3.1.

### 2.9.1. Data Access Object pattern

This subsection describes the DAO pattern, which is used in this application for accessing and storing relevant data.

The "DAO pattern" involves a DAO class that implements the concrete logic to communicate with a data source. Such data sources can be data storages such as databases or file systems. The DAO provides its clients with a fairly simple interface, i.e. an API, while hiding the implementation details needed for communicating with the data source. Even if the implementation for communicating with the data source changes, this does not affect its client as long as the interface does not change. [19]

Figure 2.14 shows the class diagram of the DAO pattern:

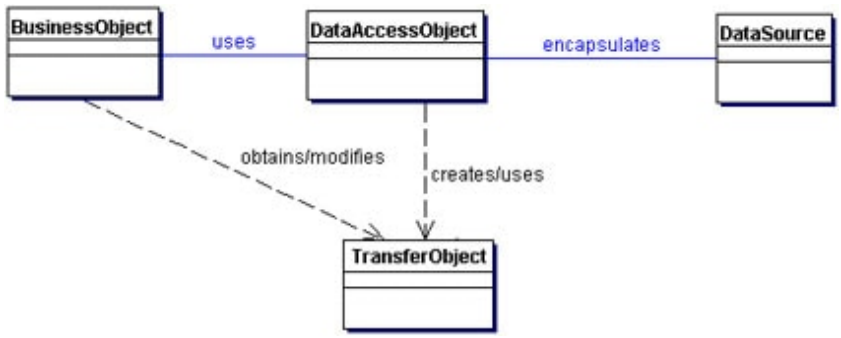


Figure 2.14.: Data Access Object structure [20]

The DAO pattern enables transparency because its client can use a data source without knowing the implementation details for communicating with it. This also reduces the code complexity of the client by offering a simple API for communicating with the data source. The concrete implementation of a DAO is located in an own data access layer - separated from the rest of the application. This also makes it easy to migrate to different data source implementations without changing the DAO's client, which improves the maintainability of the application. Another thing to mention is the need to design and implement a class hierarchy for the concrete DAOs, which can also lead to the design and implementation of a factory class hierarchy for their creation. [19]

The DAO pattern offers a good opportunity when it comes to the implementation of the logic that is needed to serialize and deserialize objects of the various domain model entities mentioned in chapter 2.5.1. The used data sources for this application are XML, CSV and DOCX files, meeting the given requirements.

## 2.10. Programming with Microsoft Word

Microsoft's Component Object Model (COM) offers "a binary interoperability standard for creating reusable software libraries that interact at run time" [21]. These libraries can be furthermore used without the need of compiling them into an application and are the basis for several Microsoft services. [21]

The COM technical overview page offers a detailed description of how COM objects are built up and how they should be used:

> *"A COM object exposes its features through an interface, which is a collection of member functions. A COM interface defines the expected behavior and responsibilities of a component, and it specifies a strongly-typed contract that provides a small set of related operations. All communication among COM components occurs through interfaces, and all services offered by a component are exposed through its interface. A caller can access only the interface member functions. Internal state is unavailable to a caller unless it is exposed in the interface. [...]*
>
> *You cannot create an instance of a COM interface by itself. Instead, you create an instance of a class that implements the interface. In C++, a COM interface is modeled as an abstract base class, which means that the interface is a C++ class that contains only pure virtual member functions. A C++ library implements COM objects by inheriting the member function signatures from one or more interfaces, overriding each member function, and providing an implementation for each function. [...]*
>
> *All COM interfaces inherit from the IUnknown interface. The IUnknown interface contains the fundamental COM operations for polymorphism and instance lifetime management. The IUnknown interface has three member functions, named QueryInterface, AddRef, and Release. All COM objects are required to implement the IUnknown interface."*
>
> Cited from [21]

COM objects also implement the *IDispatch* interface that exposes objects, methods and properties for automation purposes [22].

Furthermore, COM objects can be used by the Qt-Framework through the *QAxObject* class that inherits the *QAxBase* and the *QObject* classes and acts as a wrapper for COM objects. The instatiation of a *QAxObject* can be done with either the name of a COM object or its representative *IUnknown* pointer. [23]

The abstract *QAxBase* class, which cannot be instantiated on its own, offers an API for initializing and accessing a COM object. It is important to mention that the properties and methods of the COM objects can only be accessed if the *IDispatch* interface is implemented. Methods become also available as slots and can be accessed using the *dynamicCall()* method, returning a *QVariant*. Properties become then available as Qt properties. They can be read by using the *property()* method, returning a *QVariant*. Properties can be written through *setProperty()*. Subobjects can be gathered by calling *querySubObject()* with *QAxObject* as return value, wrapping the provided COM object. [24]

These classes, with their methods and properties, are important for getting the needed information for capturing and exporting STKs from DOCX files.

### 2.10.1. Microsoft Word object model overview

In order to properly read and write the provided STK protocol templates, it is important to understand the hierarchical structure of the Microsoft Word Object Model (MWOM).

The website *https://docs.microsoft.com/en-us/office/vba/api/overview/word* provides a detailed description of the concepts and the use of the model in programming; helpful examples in Visual Basic for Applications (VBA) are provided as well. An important feature is the MWOM, which offers a variety of objects as building blocks with their properties, methods and events. [25]

The MWOM has five top-level objects [26]:

"• *Application object*
- *Document object*
- *Selection object*
- *Range object*
- *Bookmark object"*

The *Application* object acts as the hierarchical parent object because it represents the Microsoft Word application. The Word environment can be controlled through its properties and members. *Document* objects represent an entire DOCX document with its content. When a document is newly created or opened, it is added to the *Documents* collection of the *Application* object. [26]

The *Range* object [26] is defined as follows:

> *"The Range object represents a contiguous area in a document, and is defined by a starting character position and an ending character position. You are not limited to a single Range object. You can define multiple Range objects in the same document. A Range object has the following characteristics:*
>
> - *It can consist of the insertion point alone, a range of text, or the entire document.*
> - *It includes non-printing characters such as spaces, tab characters, and paragraph marks.*
> - *It can be the area represented by the current selection, or it can represent an area different from the current selection.*
> - *It is not visible in a document, unlike a selection, which is always visible.*
> - *It is not saved with a document and exists only while the code is running.*
> - *When you insert text at the end of a range, Word automatically expands the range to include the inserted text."*

The *Selection* object represents the currently selected area in the Word GUI [26]. When working with the STK templates, i.e. loading and exporting, this object is not being used, because there are no user interactions in the process of loading the templates or exporting the finished STKs at all. This means that it does not matter much in the development of the application. The same applies to the *Bookmark* object [26], which is defined as follows:

> *"The Bookmark object represents a contiguous area in a document, with both a starting position and an ending position. You can use bookmarks to mark a location in a document, or as a container for text in a document. A Bookmark object can consist of the insertion point, or be as large as the entire document. A Bookmark has the following characteristics that set it apart from the Range object:*
>
> - *You can name the bookmark at design time.*
> - *Bookmark objects are saved with the document, and thus are not deleted when the code stops running or your document is closed.[...]"*

Since the navigation through the STK protocol templates is quite simple due to their generalized structure with the layout consisting of four tables mentioned in chapter 2.3.1, the bookmark functionality was not used.

Figure 2.15 gives an example of the hierarchical structure of the MWOM. The topmost item, i.e. the parent object, is the *Application* object. The objects with the plural names represent collections [25] of the underlying objects. Such collections can generally be accessed through a property of the superordinate object. Items of collections can usually be retrieved through the *Item(index)* method. This figure also gives an idea of the minimum number of calls required to get the font of a character in a table cell.

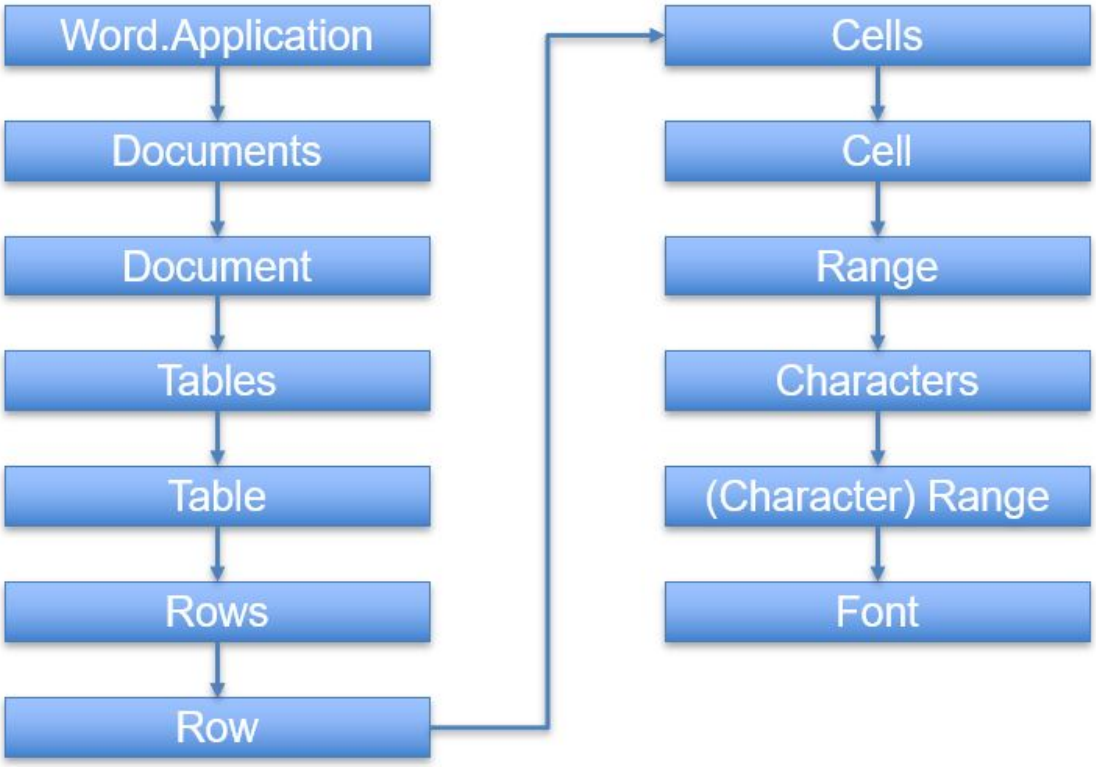


Figure 2.15.: Structure of MWOM from Word's *Application* object to a cell's character *Font* object - based on [25]

# 3. Implementation

This chapter describes the implementation details of the used concepts and patterns described in chapter 2 - and how they interact with each other to build up the software, which leads to a basic understanding of the application. It would go beyond the scope of this master's thesis to delve deeply into the implementations of the entities and containers described in chapter 2.5.1, so they were omitted here. Please note that the actual implementation of this application is generally more complex, as described in this chapter. The codebase can be viewed at the HCE institute, where it is stored.

The words "instance" and "objects" are treated as synonyms and stand for instantiated classes.

## 3.1. Implementation of the GOF creational patterns

This chapter gives a brief overview about the implementation of the GOF creational patterns used in this application that have been described in chapter 2.7.

### 3.1.1. Factory method pattern

The factory method is the design pattern used for creating objects of the domain entities and containers displayed in figures 2.6 in chapter 2.5.1, that build up

the heart of the application. It is the mostly used creational pattern in this software. Both of the major varieties of the factory method pattern mentioned in 2.7.1.3 were used. The following subchapters describe the creation of the domain entities and the containers holding them.

#### 3.1.1.1. Implementations of the abstract creator factory method

The classes showed in table 3.1 have an abstract creator base class located in the BLL and derived concrete implementation in the DAL. The naming of each abstract base class start with an 'A' for abstract. Each abstract creator class has a *pure virtual* method *create()* that has to be implemented by its derivative - the return value is an instance of the corresponding entity or container.

The abstract creator structure for creating *DeviceInspection* instances is necessary due to a technical debt that is explained in chapter 3.3.1.

No factories are designed for the *CustomerContainer* and *InspectorContainer* classes, since they are composed directly in the CROOT, as they are only created once. This will be covered in chapter 3.4.2.

| **Abstract creator base class** | **Concrete creator** |
|---|---|
| *ADeviceInspectionFactory* | *DeviceInspectionFactory* |
| *ADeviceContainerFactory* | *DeviceContainerFactory* |
| *ADeviceInspectionContainerFactory* | *DeviceInspectionContainerFactory* |
| *AInspectionContainerFactory* | *InspectionContainerFactory* |

Table 3.1.: Abstract creator base class and their concrete creators

#### 3.1.1.2. Implementations of the concrete creator factory method

The following entities have own concrete creators that implement the *create()* method, returning an instance of the corresponding entity:

- *DeviceFactory*
- *InspectorFactory*

One might notice, that the *Customer* and *Inspection* entities have no creator class that implements the factory method. Their creation requires the building up of object graphs which is provided by the *CustomerLogic* and *InspectionLogic* classes. Their functionality is explained later on.

### 3.1.2. Singleton pattern

The following four classes are designed as singletons:

- *CustomerLogic* - see chapter 3.4.1.1
- *InspectionLogic* - see chapter 3.4.1.2
- *Modelprovider* - see chapter 3.2.2
- *DeviceInspectionTemplateContainer*

Each of these singletons comes with the public methods *initInstance()* and *getInstance()*. The first is used to initialize the singleton with the depedencies it needs for working and is called in CROOT. The second method is to access the only instance of the singleton. The singleton has to be initialized through the *initInstance()* before the first *getInstance()* call. When it comes to unit testing, the implementation of which is suggested by Van Deursen and Seeman [15] and strongly recommended for future work on this application, the singletons can be initialized with instances that serve testing purposes. When DI occurs, all of these singletons are considered as stable dependencies because they meet their criteria described in chapter 2.6, so they can be referenced used outside of CROOT as discussed in chapter 2.7.1.5.

#### 3.1.2.1. *DeviceInspectionTemplateContainer* singleton

The singleton class *DeviceInspectionTemplateContainer* holds all *DeviceInspection-STK* prototypes, each of which represents a DOCX STK template provided by the PMG. As already mentioned in chapter 1.2, each device type supported by this software comes with two templates that only differ in terms of the safety parameters. The templates stored in the *DeviceInspectionTemplateContainer*

are accessed when the user creates new *DeviceInspectionSTK* instances via the GUI using the *DeviceInspectionLogic* class (see chapter 3.2.5). Another use is the change of the used standard of a *DeviceInspection* and thus its safety parameters. All newly created *DeviceInspectionSTK* instances are initially loaded from the corresponding prototype with the IEC 60601 version of a DOCX template. By changing the standard used to IEC 62353 standard when capturing an STK, the safety parameters are replaced by those of the corresponding IEC 62353 prototype.

### 3.1.3. Prototype pattern

The implementation of the Prototype pattern is all about the *DeviceInspection* class. It has the *pure virtual* method *clone()*, so it must be implemented by all of its derivatives. Currently, only one class inherits from it: the *DeviceInspectionSTK* class, which represents the STKs of active medical devices. Also every class that is associated with *DeviceInspection* or *DeviceInspectionSTK* has this cloning method implemented, resulting in deep copying. These classes are:

- *GeneralInformationsDeviceInspection*
- *ProtectionEntries*
- *Checkbox*
- *InspectionPoint* or one of its derivatives mentioned in chapter 2.5.1

## 3.2. Presentation layer

This chapter will only give a basic overview of the Presentation layer's implementation. Screenshots of the application are provided in the German user manual in the appendix.

### 3.2.1. Model/view programming

As shown in figure 3.1, the Qt-framework implements a model/view architecture that consists of three kinds of objects - models, views and delegates:

> *"The model communicates with a source of data, providing an interface for the other components in the architecture. The nature of the communication depends on the type of data source, and the way the model is implemented.*
>
> *The view obtains model indexes from the model; these are references to items of data. By supplying model indexes to the model, the view can retrieve items of data from the data source.*
>
> *In standard views, a delegate renders the items of data. When an item is edited, the delegate communicates with the model directly using model indexes."*

Cited from [27]

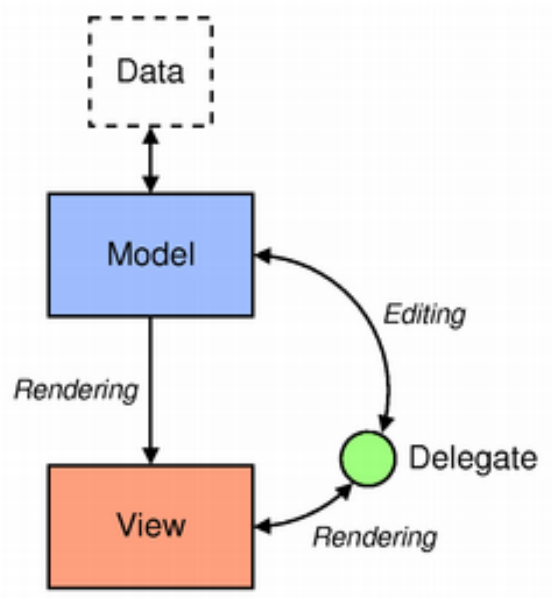


Figure 3.1.: The model/view architecture [27]

These three object types are each defined by abstract base classes that are meant to be subclassed, so they give functionality that is expected by the other components. This is particularly important when displaying data sets, for this software especially in the form of lists. To this end, the *QAbstractListModel* and *QStringListModel* model classes - that both inherit from *QAbstractItemModel* - are the most important for this application. This is because only models (not all) are being implemented in C++. All delegates and views that work with them are implemented in QML. [27]

### 3.2.2. *ModelProvider* singleton

The *ModelProvider* singleton is only used by the GUI as it provides a central object for gathering the different model types implemented in C++ in order to display data in the form of lists. It is registered as singleton type in QML. The following model classes derive either from *QAbstractListModel* or *QStringListModel*:

- *ModelCheckboxes*
- *ModelCustomers*
- *ModelDevices*
- *ModelDeviceInspections*
- *ModelInspectionPoints*
- *ModelInspectionsFinished*
- *ModelInspectionsUnfinished*
- *ModelInspectors*
- *ModelMedicalAreas*
- *ModelProtectionEntries*

These are filled with data sets of the corresponding entity that is part of the class name. The only exception is *ModelMedicalAreas*, which represents all available medical areas of a *Customer* object. These models are only implemented in C++. All delegates and views that work with them are implemented in QML. At this point, details on how to implement these models are skipped because they are

not overly complex. As already mentioned, the code base can be looked up at the HCE.

```
DeviceInspection *createDeviceInspectionFromDevice(
    Device *device);
```

Listing 3.1: *ADeviceInspectionLogic::createDeviceInspectionFromDevice()* method signature

### 3.2.3. Implementation of the GUI

The implementation of the GUI is based on the *StackView* QML type [28]. Each available screen is based on the *Item* QML type, which acts as a basic visual QML type [29]. The initial item of the *StackView* is the main menu. Each subsequent screen is pushed onto the *StackView* when navigating further into the GUI's structure, in which the current screen is always on top. When navigating back, a screen is popped - or all pages are popped when navigating directly back to the main menu, respectively. Animated transitions also were part of the UI/UX design by using *Transition* QML types when moving from one screen to another [30].

#### 3.2.3.1. Screens

Except for the main menu, there are two types of screens, as mentioned in chapter 2.4.1:

- menus - have a navigation bar only
- content pages - have both a title and a navigation bar

When it comes to the content pages, there is one underlying *ContenPageEmpty* screen of the *Item* QML type with placeholders for the title bar and the navigation bar. The area in-between is reserved for the actual content of the screen depending on its purpose. Every other content page is derived from it and places a bar in the corresponding placeholders, which differ in the combination of the optional buttons. The available buttons can be guessed by the naming of

the content pages. A Check button is available on all screens except for the ones that contain "NoCheck". The given options from *ContenPageEmpty* are:

- *ContentPageAdd*
- *ContentPageAddRemoveSearch*
- *ContentPageAddRemoveSearchNoCheck*
- *ContentPageAddSearch*
- *ContentPageRemoveSearch*
- *ContentPageRemoveSearchNoCheck*
- *ContentPageTitleOnly*

#### 3.2.3.2. Dialogs

When it comes to creating new *Customer* instances, a *FileDialog* QML type is used [31]. This is because the CSV file that represents a customer's device list must be selected. All other dialogs are based on the *Dialog* QML type. Each dialog has a custom appearance when it comes to the layout of the text and the buttons.

### 3.2.4. Relation to the BLL

The Presentation layer is fully dependent on the BLL with respect to the layer architecture described in chapter 2.8 and shown in figure 2.13. Other than the loose coupling between BLL and DAL, the Presentation layer is tightly coupled with the BLL. This is because all the needed concrete C++ instances from the BLL are being registered directly to become available in QML under the Uniform Resource Identifier (URI) "*at.hce.qml*". This URI has to be imported wherever the data types defined in the BLL are used. To loosely couple the GUI to the BLL would mean huge effort, since a separate wrapper of the classes used must be implemented plus e.g. factories and other infrastructural classes. This would also make the implementation of pure DI much more complex and harder. From the current point of view, the only layer that could be exchanged

is the DAL. So it is no big deal that the GUI is tightly coupled to the BLL as the dependency on it is still correct. The only backdraw is that unit testing would becoming harder (if automated GUI tests are implemented in the future).

### 3.2.5. Creation of *DeviceInspectionSTK* objects by the user

The *DeviceInspectionLogic* class is the only derivative of its abstract *ADeviceInspectionLogic* base class with the *pure virtual* method shown in listing 3.2.

```
1 virtual DeviceInspectionList createDeviceInspectionsFromDeviceList
    (DeviceList device_list) = 0;
```

Listing 3.2: *ADeviceInspectionLogic::createDeviceInspectionsFromDeviceList() pure virtual* method signature

The concrete implementation of this method in the *DeviceInspectionLogic* class is responsible for creating a list of *DeviceInspection* instances from a list of *Device* objects selected by the user on the GUI. For each *Device* instance in the list, the method displayed in listing 3.1 is called, passing it as parameter. This method uses the prototypes in the *DeviceInspectionTemplateContainer* to search for the one with the matching device type and the IEC 60601 safety parameters. The selected prototype is cloned and a *GeneralinformationDeviceInspection* instance is created with the information contained in the *Device*'s *GeneralinformationDevice* object attribute. This newly created *DeviceInspection* instance, more precisely *DeviceInspectionSTK*, is returned and then stored in a list. This process is part of creating new *Inspection* instances through the user.

## 3.3. Data persistence

As already mentioned in chapter 2.9, the CSV and the XML format are used for persisting the relevant data. Before looking into the details of the file system used as persistent storage (to meet the requirements outlined in chapter 2.1), a technical debt must first be covered in the next subchapter.

### 3.3.1. Technical debt regarding the persistence of *DeviceInspection* objects

Unfortunately, there is a technical debt regarding the persistence of *DeviceInspection* objects. This is because all *DeviceInspection* objects that do not act as prototypes are persisted as part of their parent *Inspection*. An *Inspection* instance persists all of its *DeviceInspection* instances into one XML tag in the form of a JSON object that acts as collection of all *DeviceInspection* instances in JSON object form, which might seem strange at the first glance. This is also true for the prototypes - as they are persisted in the same way in an own XML file. This is a technical debt that originated in the first attempts to build the prototypes out of the DOCX STK templates provided by the PMG. The first approach was to rebuild these templates in JSON format. In general, this was achieved because the structure mentioned in chapter 2.3 was completely covered by the JSON representation. Text formatting was rebuilt with the help of HTML tags.

Work on the application was based on this approach until editability of these JSON templates was required: If a DOCX template had to be changed, its JSON representation also had to, which could have resulted in inconsistency. This, and the fact that the staff members of the PMG using the application must be capable of changing the JSON structure of the corresponding *DeviceInspection* objects, lead to a change of this approach so that the templates could be serialized and converted into the JSON format already used. Unfortunately, the development was so advanced that changing the persistence of *DeviceInspection* objects in XML format only would have resulted in an enormous workload - so this change has not been made. The persistence of entities and the functionality of the application still meets all necessary concepts and requirements, nevertheless.

Each class shown in figure 2.7 is responsible to build itself up from JSON and to convert itself to JSON respectively. Therefore, all classes implement the *toJson()* and *fromJson()* methods. For data lists, reading is done using the static *JsonReader* class (only static methods, therefore no instantiation is needed) and writing is done using the static *JsonWriter* class.

### 3.3.2. File systems

This application uses different files for storing relevant data. As a given requirement (see chapter 2.1), a customer's device list is stored in a CSV file. All other files for persistent storage are in the XML format:

- one XML file for all *Customer* objects (customers.xml)
- one XML file for all *Inspector* objects (inspectors.xml)
- one XML file for all *DeviceInspection* objects treated as prototypes that represent the DOCX STK templates provided by the PMG
- one XML file for all *Inspection* objects per *Customer* object

The nomenclature for the XML files regarding the persistence of all *Inspection* instances per *Customer* object is the name of the corresponding customer, followed by "_inspections.xml", e.g. "CustomerX_inspections.xml".

Each of these mentioned configuration files comes with a corresponding backup file and their paths are provided by the static *Systempathprovider* class located in the DAL.

### 3.3.3. *Configbearer* class

The classes *Customer*, *Device*, *Inspection* and *Inspector* inherit from the *Configbearer* class. It is responsible for holding all the attributes that need to be persisted in the custom defined key-value storage that is a member variable, displayed in listing 3.3:

```
1 typedef QMap<QString,QString> Config;
```

Listing 3.3: *Config* key-value storage

The *Configbearer* class also has a *pure virtual* method, that has to be implemented by all of its derivatives shown in listing 3.4. This method is called every time the key-value storage changes. The method's behaviour is implemented through its derivatives.

```
virtual void configChanged() = 0;
```

Listing 3.4: *Configbearer::configChanged() pure virtual* method signature

### 3.3.4. *MicrosoftWordApplicationConnector* class

The *MicrosoftWordApplicationConnector* class is responsible for managing the connection to the Microsoft Word's *Application* COM object. Its method, *connectToMicrosoftWord()*, establishes a connection to the top-level object in the Word object model hierarchy (see chapter 2.10.1) - the *Application* object - by creating a new *QAxObject* as shown in listing 3.5. This leads to the start of the Microsoft Word application itself. The *QAxObject* instance is furthermore stored as member and can be accessed through the *getMicrosoftWordApplication()* method by its consumers. Every operation needed for reading the STK DOCX protocol templates and writing the STK documentation is based on the *Application* COM object.

```
void MicrosoftWordApplicationConnector::connectToMicrosoftWord()
{
    QAxObject *microsoft_word_application = nullptr;

    microsoft_word_application = new QAxObject("Word.Application",
                                               this);

    if(microsoft_word_application)
    {
        m_microsoft_word_application = microsoft_word_application;

        emit signalMicrosoftWordApplicationConnected();
    }
}
```

Listing 3.5: *MicrosoftWordApplicationConnector::connectToMicrosoftWord()* method implementation

The *MicrosoftWordApplicationConnector* also has the method *disconnectFromMicrosoftWord()* that closes all open documents as well as the Microsoft Word application itself, as shown in listing 3.6:

```
void MicrosoftWordApplicationConnector::
    disconnectFromMicrosoftWord()
{
    if(m_microsoft_word_application)
    {
        //close all open documents
        QAxObject *documents = m_microsoft_word_application->
            querySubObject("Documents");

        if(documents->property("Count").toInt())
            documents->dynamicCall("Close(qint32)",
                                   0);

        delete documents;

        m_microsoft_word_application->dynamicCall("Quit()");

        m_microsoft_word_application->deleteLater();
        m_microsoft_word_application = nullptr;
    }
}
```

Listing 3.6: *MicrosoftWordApplicationConnector::disconnectFromMicrosoftWord()* method implementation

### 3.3.5. Serializers and deserializers

In order to achieve persistence with file system as well as importing the *DeviceInspection* template prototypes and exporting the STK documentations in DOCX format, classes had to be implemented that act as serializers and deserializers for the corresponding file format. These classes are described in the following subsections.

#### 3.3.5.1. *XmlConfigProvider* class

The *XmlConfigProvider* class provides the logic when it comes to serialize or deserialize data from or to XML format. The following explanation does not cover details of the XML format, but merely gives a simple overview about how the files are structured.

The mentioned class is used by a corresponding DAO for communication with the XML data source. Each XML file has elements, i.e. sections, each representing one object, e.g. a *Customer* object. The tags of these section usually include a pre-string (e.g. "customer_config_"), followed by the object's Universally Unique Identifier (UUID). Sub-elements of such sections represent data that has to be stored and lie in-between CDATA sections. Any character or combination can usually be placed within these sections without harming the integrity of the XML files (even HTML tags). The only exception is the escape sequence "]]>", which means the end of a CDATA section. Additionally, all data within these CDATA sections is Base64 encoded, so that even if e.g. a customer's name that contains the escape sequence is put into the CDATA section, the XML file would still remain intact.

Since navigating through an XML file is very time-consuming if only a certain element needs to be changed, these files are completely rewritten, i.e. flushed, with each change. To reduce the amount of such rewriting, this only happens when a delay timer implemented in the logic is triggered. If subsequent changes occur, the timer restarts.

#### 3.3.5.2. *CsvConfigProvider* class

The *CsvConfigProvider* class provides the logic when it comes to serializing or deserializing the device lists of customers, meeting the corresponding requirements mentioned in chapter 2.1.

A *CsvConfigProvider* instance is used by a corresponding DAO for the communication with the CSV data source. The CSV files used by this application

have a fixed structure that is equal to the list of device-related information provided in chapter 2.1, in the very same order. The first line acts as header and contains the strings that are defined in the *csvconfigprovider.h* file, separated by a semicolon. Each additional line represents a single *Device* object, in which each device-related information is also separated by a semicolon. A detailed description plus examples can be found in the German user manual located in the appendix.

The behaviour when writing a CSV file is similar to the behaviour when writing XML files described in the previous chapter - as each file is also written as a whole at once.

#### 3.3.5.3. Reading of STK DOCX templates

Three static classes are responsible for the reading of the STK DOCX templates: *DocxDeviceInspectionTemplateReader*, *DocxInspectionPointReader* and *DocxHelper*.

The *DocxDeviceInspectionTemplateReader* class reads and creates *DeviceInspection* instances from the STK DOCX templates that act as prototypes. It only has one public static method, shown in listing 3.7:

```
1 static DeviceInspection *read(DeviceInspectionType type,
2                                QAxObject *docx_template);
```

Listing 3.7: *DocxDeviceInspectionTemplateReader::read()* method signature

Based on the *DeviceType*, it is anticipated which concrete instance of the *DeviceInspection* class, in this case only its single *DeviceInspectionSTK* derivative, is created. The *QAxObject* hereby wraps a document object of a STK DOCX protocol template. All other static methods of the *DocxDeviceInspectionTemplateReader* are protected and are used to build together a *DeviceInspectionSTK* object - by navigating through the STK DOCX template structure explained in chapter 2.3 and deserializing all necessary information. This also involves the

serializing and creating of a *ProtectionEntries* instance. The *DocxDeviceInspectionTemplateReader* invokes the only static public method from the *DocxDeviceInspectionTemplateReader* shown in listing 3.8 when inspection points in the fourth (and largest) table are read. The *qint32* represents the cell count of a row in table four, whereas the *QAxObject* parameter points to the *Cells* collection of a *Row* COM object.

Both the *DocxDeviceInspectionTemplateReader* and *DocxInspectionPointReader* classes are using the static methods of the *DocxHelper* class to get their job done. The *DocxHelper* holds several static methods, that are invoked by other classes involved for reading and writing DOCX files. Its static methods cover tasks such as the reading and writing of *Checkbox* objects. It also contains methods for parsing formatted or unformatted cell texts. In order to rebuild the textual formatting in the STK DOCX templates, HTML tags are inserted that can be displayed via the QT framework [32]. This is done by the static method *getCellTextFormatted()* of the *DocxHelper* class by checking the font of every single character contained in a cell. The method gets a *QAxObject* instance that represents a *Cell* COM object.

As shown in figure 2.15, four subobjects have to be accessed to gather the *Font* COM object. Through this object's properties - *Bold*, *Italic* and *Subscript* - the corresponding HTML tags can be set to rebuild the textual formatting of a STK DOCX protocol template. [25]

The static *DocxInspectionPointReader* class reads and creates *InspectionPoint* objects or one of its derivatives shown in figure 2.7. This is done by its only public static method *read()* - its signature is shown in listing 3.8. The anticipation of which concrete *InspectionPoint* instance should be created is based on the cell count per row in the fourth table and also on the count of cells with a gray background color (which represent information cells). With these two parameters every type of *InspectionPoint* can be built up. All other static methods of the *DocxInspectionPointReader* class are protected and serve to build up every *InspectionPoint* object type.

```
static InspectionPoint *read(qint32 cell_count, QAxObject *cells);
```

Listing 3.8: *DocxInspectionPointReader::read()* method signature

The reading of STK protocol templates heavily depends on the given structure. Major layout changes in the protocol templates therefore also require a change of the algorithm. Only the fourth table containing the inspection points is dynamic when it comes to the serialization of their different types.

#### 3.3.5.4. Writing of STK DOCX documentations

Three static classes are responsible for writing the STK DOCX documentation: *DocxDeviceInspectionExporter*, *DocxInspectionPointExporter* and *DocxHelper*.

The static *DocxDeviceInspectionExporter* class is responsible for writing the DOCX documentation of captured STKs. Its only public static method *exportDeviceInspectionSTK()* - shown in listing 3.9 - is invoked by its client, which starts the exporting process.

```
static bool exportDeviceInspectionSTK(QAxObject *docx_template,
                                      DeviceInspectionSTK* di);
```

Listing 3.9: *DocxDeviceInspectionExporter::exportDeviceInspectionSTK()* method signature

This method gets a *QAxObject* instance, which acts as wrapper to the *Document* COM of the STK protocol to be filled, and an instance of the *DeviceInspectionSTK*, which holds the relevant data. The static *DocxDeviceInspectionExporter* also has protected methods for exporting *GeneralInformationDeviceInspection* and *ProtectionEntries* objects - as well as for the equipment in the form of a list holding *Checkbox* objects. This class also uses the only public static method of the *DocxInspectionPointExporter* class to export the different *InspectionPoint* object types - *setInspectionPointFromInspectionPointList()* - displayed in listing 3.10. The instance *QAxObject* is received as a parameter, wrapping the *Cells* collection of a *Row* COM object and list containing *InspectionPoint* objects that are to be exported.

Both the *DocxDeviceInspectionTemplateReader* and *DocxInspectionPointReader* classes are using the static methods of the *DocxHelper* class.

```
static void setInspectionPointFromInspectionPointList(
            QAxObject* cells, InspectionPointList ip_list);
```

Listing 3.10: *DocxInspectionPointExporter::setInspectionPointFromInspectionPointList()* method signature

Just like the reading of the STK DOCX templates, the writing of the STK documentation also depends heavily on the given structure of the DOCX templates explained in chapter 2.3.1.

### 3.3.6. Data access objects

Each container class introduced in chapter 2.5.1 uses a DAO for persisting its corresponding domain entities. Each of these DAOs has an abstract base class located in the BLL, with each class name starting with an 'A' for abstract. The concrete implementations of these abstract classes are located in the DAL and are dependent on the data source(s) they are communicating with. The containers take the concrete DAOs in form of constructor injections. The following subsections include UML class diagrams for each of the containers, their DAOs and the used transfer objects. Please note that the classes covered are more complex and that only the attributes and operations required to explain the persistence are dealt with here.

#### 3.3.6.1. DAO for CSV data source

Figure 3.2 shows the association between the classes *DeviceContainer* and *ADeviceDAO*. The latter has one derivative - *DeviceDAOCsv* - which is injected into the former when composing the object graphs. Furthermore, the *DeviceDAOCsv* object receives a *CsvConfigProvider* instance at its instantiation, which is needed for the (de)serialization of *Device* objects.

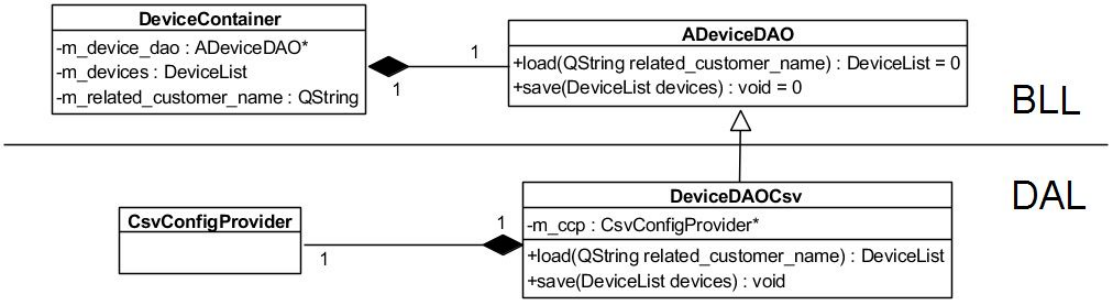


Figure 3.2.: *DeviceContainer* and its DAO structure

### 3.3.6.2. DAOs for XML data source

The *CustomerContainer* gets a DAO injected through constructor injection, in form of a *CustomerDAOXml* instance at the object graph composition as shown in figure 3.3. The *CustomerDAOXml* object furthermore receives a *XmlConfigProvider* object at its instantiation, which is needed for the (de)serialization of *Device* objects from or to XML files.

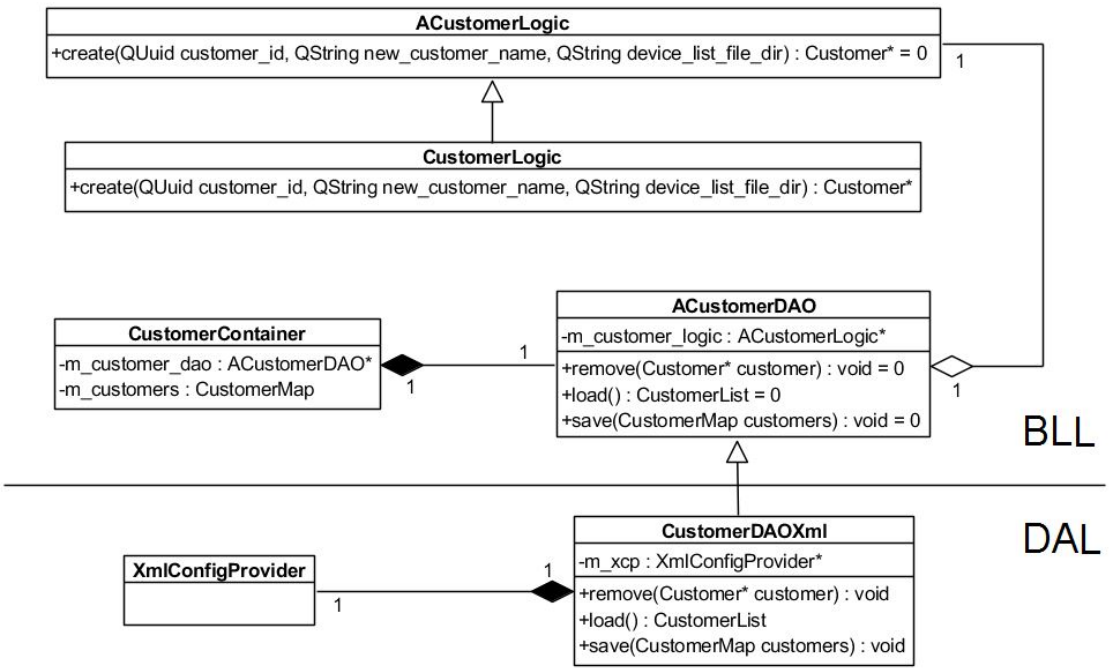


Figure 3.3.: *CustomerContainer* and its DAO structure

The same is true for the following containers and their DAOs persisting *Inspec-*

*tion*, *Inspectors* and *DeviceInspection* objects respectively:

- *InspectionContainer* - figure 3.4
- *InspectorContainer* - figure 3.5
- *DeviceInspectionTemplateContainer* - figure 3.6

The *CustomerDAOXml* instance also receives a *CustomerLogic* instance via constructor injection. It is responsible for creating the object graphs of *Customer* objects that have *DeviceContainer* and *InspectionContainer* instances as members.

The *InspectionDAOXml* gets an *InspectionLogic* instance via constructor injection. It is used to create *Inspection* entities with their *DeviceInspectionContainer* members already injected.

The *DeviceInspectionTemplateContainer* class has a second DAO responsible for reading DOCX data sources - which is covered in the next chapter. It stores all of the DOCX STK templates in JSON representations into an own XML file. This is because of the technical debt explained in chapter 3.3.1. The mentioned class furthermore receives a *DeviceInspectionFactory* instance through constructor injection. Its purpose is to create *DeviceInspection* objects from JSON, that are stored in XML.

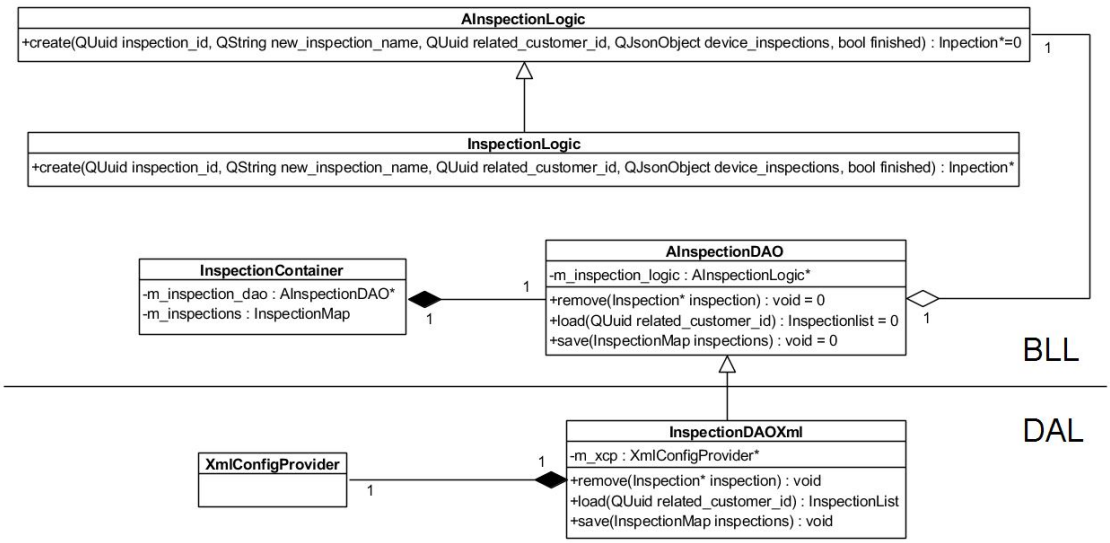


Figure 3.4.: *InspectionContainer* and its DAO structure

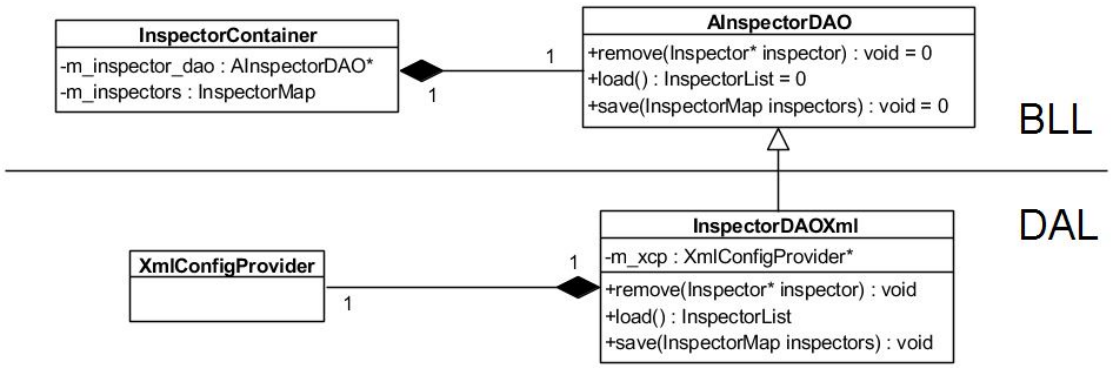


Figure 3.5.: *InspectorContainer* and its DAO structure

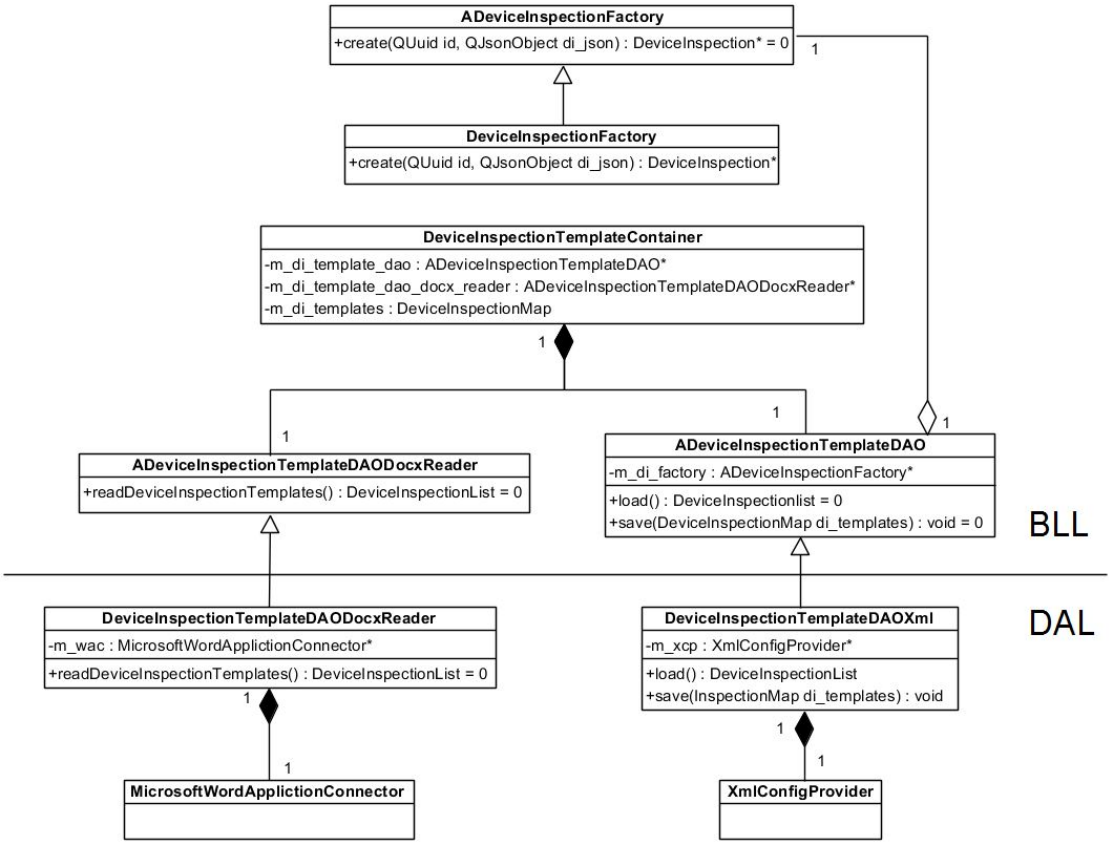


Figure 3.6.: *DeviceInspectionTemplateContainer* and its DAO structure

#### 3.3.6.3. DAOs for DOCX data source

A *DeviceInspectionDAODocxWriter* object is injected into a *DeviceInspectionContainer* when composing the object graphs as shown in figure 3.7. To meet the requirements, a DAO for exporting STK documentation was designed. There is no second DAO for XML persistence involved because of the mentioned technical debt.

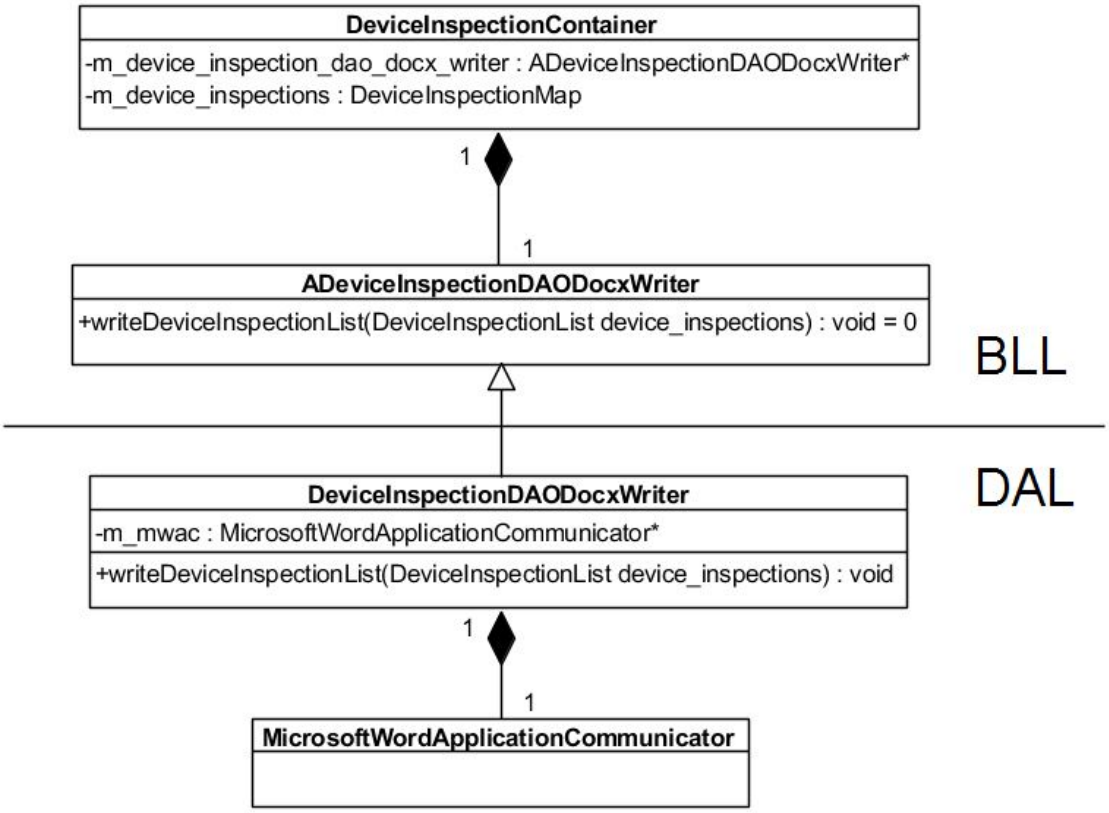


Figure 3.7.: *DeviceInspectionContainer* and its DAO structure

The *DeviceInspectionDAODocxWriter* object receives a *MicrosoftWordApplicationConnector* object at its instantiation, which is needed for establishing a connection to the Microsoft Word *Application* COM object. It uses the only static method *openDocxDocument()* of the static *DocxDocumentOpener* class internally for opening the right DOCX template and for returning a *QAxObject* instance wrapping a *Document* COM object (see listing 3.11). Its parameters are a *QAxObject* instance

wrapping the *Application* COM object, a *QString* that contains the directory, and a *QString* representing the DOCX file name. Finally, the static *DocxDeviceInspectionExporter* class is used for exporting the finished STK documentations.

```
static QAxObject* openDocxDocument(QAxObject *word_application,
                                   QString dir_path,
                                   QString file_name);
```

Listing 3.11: *DocxDocumentOpener::openDocxDocument()* method signature

The workflow of exporting STK documentations into the DOCX format involves the *writeDeviceInspectionList()* method with its parameter of the *DeviceInspectionList*, which is a *QList<DeviceInspection*> typedef*. For each *DeviceInspection* to be exported, the corresponding STK template is opened and then filled with data. All names of the STK DOCX template files are defined in the file *definitions_device_inspection_template_file_name_static.h* under the *DeviceInspectionTemplateFileNameStatic namespace*. Through the invocation of the *SaveAs2()* method of the*Document* COM object - that is wrapped by a *QAxObject* instance - the file is saved to the location that is handed over as a parameter (see listing 3.12). Then the still open template is closed without saving and remains in its original state.

```
docx_template->dynamicCall("SaveAs2(QVariant)",
                           QVariant(export_path +
                                    export_file_name));
```

Listing 3.12: Invocation of the *Document* COM object's *SaveAs2()* method

The *DeviceInspectionTemplateContainer* is the only container with two DAOs as displayed in figure 3.6. The DAO for persisting the STK templates into XML was mentioned in the previous chapter. The second object that is necessary for it to work is injected into the *DeviceInspectionTemplateContainer* via constructor injection and is a *DeviceInspectionTemplateDAODocxReader* instance, which first receives a *MicrosoftWordApplicationConnector* at its instantiation. It furthermore uses the static *DocxDocumentOpener* to open the DOCX STK templates - that internally act as prototypes after their serializing - and uses *DocxDeviceInspectionReader* classes to read the templates.

# 3.4. Object graph composition

This chapter describes the composition of object graphs for both at runtime and in the CROOT close to the application's entry point.

## 3.4.1. Composition of object graphs at runtime

Two classes are responsible to build up the object graphs:

- *CustomerLogic* for creating *Customer* instances with their containers
- *InspectionLogic* for creating *Inspection* instances with their containers

### 3.4.1.1. Creation of *Customer* objects

The *CustomerLogic* is the only derivative of its abstract base class *ACustomerLogic*. This abstraction was made to maintain loose coupling between BLL and DAL. Although the concrete *CustomerLogic* class is designed as singleton, its only instance is injected into the *CustomerDAOXml* object using constructor injection, as mentioned in chapter 3.3.6.2. The abstract base class *ACustomerLogic* gets its variable instances *DeviceContainerFactory* and *InspectionContainerFactory* via constructor injection. These are stored as members and can be accessed by its derivative through corresponding protected getter methods.

Listing 3.13 displays the object graph composition of figure 3.2, wherein a new *DeviceContainer* instance is created and returned by the *DeviceContainerFactory*'s *create()* method.

```
DeviceContainer *DeviceContainerFactory::create(
    QString customer_name, QString device_list_file_dir)
{
    CsvConfigProvider* ccp_device =
        new CsvConfigProvider(device_list_file_dir);

    DeviceDAOCsv *device_dao = new DeviceDAOCsv(ccp_device);

    DeviceContainer *device_container =
        new DeviceContainer(device_dao,customer_name);

    return device_container;
}
```

Listing 3.13: *DeviceContainerFactory::create()* method implementation

Listing 3.14 shows the composition of the object graph in the *InspectionContainerFactory*'s *create()* method of figure 3.4. First, an *XmlConfigProvider* instance is created. As already mentioned, the static *SystemPathProvider*'s methods return the directories necessary for persistence. *FileNameStatic* is a *namespace* that is defined in the file *definitions_file_name_static.h*, in which the file names for persisting are stated.

Listing 3.15 shows the implementation of the *create()* method that returns a newly created *Customer* instance. It gets the container objects injected that are instantiated prior by the corresponding factories. Afterwards, the *load()* method of both containers is called to load the persisted data.

```
InspectionContainer *InspectionContainerFactory::create(
    QUuid customer_id, QString customer_name)
{
    XmlConfigProvider *xcp_inspection = new XmlConfigProvider(
                    SystemPathProvider::getPathConfigBase(),
                    customer_name + "_" +
                        FileNameStatic::FILENAME_INSPECTIONS);

    InspectionDAOXml *inspection_dao = new InspectionDAOXml(
                getInspectionLogic(), xcp_inspection);

```

```
12      InspectionContainer *inspection_container =
13              new InspectionContainer(inspection_dao, customer_id);
14
15      return inspection_container;
16  }
```

Listing 3.14: *InspectionContainer::create()* method implementation

```
1   Customer *CustomerLogic::create(QUuid customer_id,
2                                   QString new_customer_name,
3                                   QString device_list_file_dir)
4   {
5       DeviceContainer *device_container;
6       device_container = getADeviceContainerFactory()->create(
7                   new_customer_name, device_list_file_dir);
8
9       InspectionContainer *inspection_container;
10      inspection_container = getAInspectionContainerFactory()->
11          create(customer_id, new_customer_name);
12
13      Customer *new_customer = new Customer(
14          customer_id, new_customer_name, device_list_file_dir,
15          device_container, inspection_container);
16
17      device_container->load();
18      inspection_container->load();
19
20      return new_customer;
21  }
```

Listing 3.15: *CustomerLogic::create() method* implementation

#### 3.4.1.2. Creation of *Inspection* objects

The *InspectionLogic* is the only derivative of its abstract base class: *AInspection-Logic*. The reason for this abstraction is the same as in the previous subchapter. The *InspectionLogic* singleton's instance is injected into the *InspectionDAOXml* object using constructor injection. The abstract base class *AInspectionLogic* gets a

*DeviceInspectionContainerFactory* instance injected via constructor injection. It is stored as member variable and can be accessed by its derivative through the corresponding getter method.

Listing 3.16 displays the composition of the object graph of figure 3.7. The first step is the instantiation of a *MicrosoftWordApplicationConnector* object, which is passed to the *DeviceInspectionDAODocxWriter* at its creation. This *DeviceInspectionDAODocxWriter* instance is then injected into the *DeviceInspectionContainer* through constructor injection at its instantiation. The newly created *DeviceInspectionContainer* object is finally returned by the *DeviceInspectionContainer*'s *create()* method.

```
1  DeviceInspectionContainer *DeviceInspectionContainerFactory::
       create()
2  {
3      MicrosoftWordApplicationConnector *mwac =
4              new MicrosoftWordApplicationConnector();
5
6      DeviceInspectionDAODocxWriter *device_inspection_dao =
7              new DeviceInspectionDAODocxWriter(mwac);
8
9      DeviceInspectionFactory *di_factory =
10         new DeviceInspectionFactory();
11
12     DeviceInspectionContainer *device_inspection_container =
13             new DeviceInspectionContainer(
14                 device_inspection_dao, di_factory);
15
16     return device_inspection_container;
17 }
```

Listing 3.16: *DeviceInspectionContainer::create()* method implementation

Listing 3.17 shows the implementation of the *InspectionLogic*'s *create()* method, that is defined by its abstract base class as *pure virtual*. A *DeviceInspectionContainer* instance is created through the *DeviceInspectionContainerFactory*'s *create()* method which is needed by the *Inspection* object at its instantiation. Afterwards, the container's *fromJson()* method is being called with the data stored

in a *QJsonObject* that is passed as a parameter. This method is used in the *InspectionDAOXml* class when loading the persisted *Inspection* objects.

```
Inspection *InspectionLogic::create(
        QUuid inspection_id, QString new_inspection_name,
        QUuid related_customer_id,
        QJsonObject device_inspections, bool finished)
{
    DeviceInspectionContainer *di_container =
        getADeviceInspectionContainerFactory()->create();

    Inspection* new_inspection = new Inspection(
            inspection_id, new_inspection_name,
            related_customer_id, di_container,
            finished);

    di_container->fromJson(device_inspections);

    return new_inspection;
}
```

Listing 3.17: *InspectionLogic::create()* method implementation

The *InspectionLogic* has an overloaded *create()* method, displayed in listing 3.18, which is defined by its abstract base class. Its purpose is to create *Inspection* instances through the GUI. In contrast to its abstract base class, the concrete *InspectionLogic* receives an *DeviceInspectionLogic* instance additionally through constructor injection. This purpose is to create a list of *DeviceInspection* objects out of a list of *Device* instances selected by the user on the GUI, as explained in chapter 3.2.5.

```
Inspection *InspectionLogic::create(
    QString new_inspection_name, QUuid related_customer_id,
    DeviceContainer* dev_container)
{
    QUuid new_inspection_id = QUuid::createUuid();

    DeviceInspectionContainer *di_container =
        getADeviceInspectionContainerFactory()->create();

```

```
10      di_container->add(m_di_logic->
11                        createDeviceInspectionsFromDeviceList(
12                          dev_container->getDevicesSelected()),
13                        false);
14
15      Inspection* new_inspection = new Inspection(
16                  new_inspection_id, new_inspection_name,
17                  related_customer_id, di_container);
18
19      di_container->slotDeviceInspectionChanged();
20
21      return new_inspection;
22  }
```

Listing 3.18: *InspectionLogic::create()* method implementation (overloaded)

### 3.4.2. Composition of object graphs in the CROOT

The *Initializer* class that represents the CROOT is responsible for building up the initial object graphs. This class is instantiated directly in the *main.cpp*, after which the only public *startInit()* method is invoked (see listing 3.19). The latter method calls the protected *initLogics()* first and then calls the also protected *initContainers()* method that sets the only *CustomerContainer* and *InspectorContainer* instances. They are finally passed to the *ModelProvider* singleton's *initInstance()* method as parameters.

```
void Initiator::startInit()
{
    CustomerContainer *customer_container = nullptr;
    InspectorContainer *inspector_container = nullptr;

    initLogics();
    initContainers(customer_container, inspector_container);

    ModelProvider::initInstance(customer_container,
                                inspector_container,
                                this);
}
```

Listing 3.19: *Initiator::startInit()* method implementation

Listing 3.20 shows the implementation of the *initLogics()* method of the of the *Initializer* classes. At first, the *InspectionLogic* singleton is initialized through the *initInstance()* method with the prior created *InspectionLogic* and *DeviceInspectionContainerFactory* instances and gets them as parameters. The same goes for the *CustomerLogic* singleton, as it gets *DeviceContainerFactory* and *InspectionContainerFactory* instances as parameters passed into its *initInstance()* method.

```
void Initiator::initLogics()
{
    /// InspectionLogic
    DeviceInspectionLogic *device_inspection_logic =
            new DeviceInspectionLogic();
    DeviceInspectionContainerFactory
        *device_inspection_container_factory =
            new DeviceInspectionContainerFactory();

    InspectionLogic::initInstance(
        device_inspection_logic,
        device_inspection_container_factory,
        this);

    /// CustomerLogic
    DeviceContainerFactory *device_container_factory =
            new DeviceContainerFactory();
```

```
18
19      InspectionContainerFactory *inspection_container_factory =
20              new InspectionContainerFactory(
21                  InspectionLogic::getInstance());
22
23      CustomerLogic::initInstance(device_container_factory,
24                                  inspection_container_factory,
25                                  this);
26  }
```

Listing 3.20: *Initiator::initLogics()* method implementation

Listing 3.21 displays the implementation of the *initContainers()* method. At first, a *CustomerContainer* instance object graph is composed (see figure 3.3). A *XmlConfigProvider* instance is created, which is needed by *CustomerDAOXml* objects creation. This *CustomerDAOXml* instance is furthermore injected into the newly created *CustomerContainer* instance. The composition of the *InspectorContainer*'s object graph like in figure 3.5 is similar to the composition of the *CustomerContainer*'s.

At last, the *DeviceInspectionTemplateContainer* singleton is initialized. Its object graph is built up as shown in figure 3.6. This involves the creation of a *DeviceInspectionTemplateDAOXml* instance, which gets a prior created *XmlConfigProvider* and the *DeviceInspectionFactory* instances injected. The second DAO of the *DeviceInspectionTemplateContainer*, the *DeviceInspectionTemplateDAODocxReader*, gets a *MicrosoftWordApplicationConnector* object at its instantiation. After this, both DAO instances are being passed to the *DeviceInspectionTemplateContainer* singleton's *initInstance()* method as parameters.

```
1  void Initiator::initContainers(
2      CustomerContainer *&customer_container,
3      InspectorContainer *&inspector_container)
4  {
5      /// CustomerContainer
6      XmlConfigProvider* xcp_customer =
7          new XmlConfigProvider(
8              SystemPathProvider::getPathConfigBase(),
9              FileNameStatic::FILENAME_CLIENTS);
```

```
10
11      CustomerDAOXml *customer_dao = new CustomerDAOXml(
12                  CustomerLogic::getInstance(), xcp_customer);
13
14      customer_container = new CustomerContainer(customer_dao);
15
16      customer_container->load();
17
18      /// InspectorContainer
19      XmlConfigProvider* xcp_inspector =
20          new XmlConfigProvider(
21              SystemPathProvider::getPathConfigBase(),
22              FileNameStatic::FILENAME_INSPECTORS);
23
24      InspectorDAOXml *inspector_dao =
25          new InspectorDAOXml(xcp_inspector);
26
27      inspector_container = new InspectorContainer(inspector_dao);
28
29      inspector_container->load();
30
31      /// DeviceInspectionTemplateContainer
32      XmlConfigProvider* xcp_di_template =
33          new XmlConfigProvider(
34              SystemPathProvider::
35                  getDeviceInspectionTemplatesPath(),
36              FileNameStatic::
37                  FILENAME_DEVICE_INSPECTION_TEMPLATES);
38
39      DeviceInspectionFactory *di_factory =
40          new DeviceInspectionFactory();
41
42      DeviceInspectionTemplateDAOXml *di_template_dao =
43          new DeviceInspectionTemplateDAOXml(
44              di_factory, xcp_di_template);
45
46      MicrosoftWordApplicationConnector *mwac_di_template =
47          new MicrosoftWordApplicationConnector();
48
```

```
49      DeviceInspectionTemplateDAODocxReader *
        di_template_dao_docx_reader =
50          new DeviceInspectionTemplateDAODocxReader(mwac_di_template
        );
51
52      DeviceInspectionTemplateContainer::initInstance(
53              di_template_dao, di_template_dao_docx_reader, this);
54      DeviceInspectionTemplateContainer::getInstance()->load();
55  }
```

Listing 3.21: *Initiator::initContainers()* method implementation

# 4. Discussion and outlook

This chapter begins by discussing relevant topics, followed by an outlook on further functionalities related to the software design and the application, respectively.

## 4.1. DOCX templates formatting

The exported STKs depend heavily on the formatting of the DOCX templates. As mentioned in chapter 3.3.4, the DOCX templates are opened after the connection to a Microsoft Word *Application* COM object is established. The data is then filled in and changes get saved as an separate file. The template is then closed, maintaining its state and form as a template. This means, that the formatting of the templates is crucial for the style of the exported STK reports. The formatting style can of course be set via the COM object, which leads to more API calls. Doing so would result in poorer performance of generating the reports. Each of these reports is checked by the inspector when the finished document is signed. This was a part of the decision to make filled STK protocols editable, as mentioned in chapter 1.3. Another reason was to check of the entire document style after export. With this in mind, the decision was made to rely fully on the generalized pre-formatting of the DOCX STK templates. Some cells may need to be adjusted after export, e.g. the font size when there is a large amount of text to fit in. However, such changes take less time than the additional workload associated with the needed API calls to edit the cell and text layout through the application.

It is also important to mention that the algorithm for serializing and deserializing STK templates in order to export the documentations of the captured STKs depends heavily on the current structure of the templates mentioned in chapter 2.3. If major structural changes occur in the future, the logic must be updated to maintain the corresponding functionalities.

## 4.2. Test phase

The functionality of this application was tested based on the German user manual, which is also provided in the appendix. It was carried out and verified by a staff member of the PMG and a student assistant of the HCE. A test phase was also planned, in which a complete inspection with STKs of several active medical devices was to be carried out. Unfortunately, the staff member mainly responsible for the STKs left the PMG, which is why it could not be done. It is therefore strongly advised to make up for this as reviewing this application in action is very important.

## 4.3. Technical debt

For future development, the technical debt described in chapter 3.3.1 is negligible if the persistence involving file systems does not change. It is true that the DAL layer can be easily swapped due to the loose coupling between BLL and DAL through the use of DI. But the way of persisting *DeviceInspection* objects as part of an *Inspection* in JSON format may be unsuitable when it comes to the design of the databases. If a change from file systems to databases is necessary, the persistence of *DeviceInspection* objects as a collection of JSON objects into one JSON object shall be resolved first in order to meet a database design where *DeviceInspection* objects are being stored in a different way.

## 4.4. Important features for future application versions

Although all the necessary specifications mentioned in chapters 1.3 and 2.1 have been met, the application should cover some functionalities in the future in order to achieve a better overall experience and usability.

### 4.4.1. Important features regarding the software design

#### 4.4.1.1. Logging

Implementing a wholesome logging feature is recommended. Such logs can cover vital events, processes and messages that occur at runtime and should be written into a log file. This feature is also important for tracking data changes and events that have been triggered by the user for supporting or debugging purposes. By looking up the log file, some bugs can be more easily understood or even reproduced.

#### 4.4.1.2. Unit testing

Unit tests should be written for the individual modules already implemented, i.e. units of the application. These automated tests are important to verify the correctness of the functionality or behaviour of such modules for e.g. after changes or the addition of features. If the software is further developed by students, especially one after the other and not parallel, this is a mandatory function. Students who add features as part of their projects or theses should also add unit test that cover the functionality of the modules they implement, so that subsequent developers can more easily test and maintain the correct behaviour of the modules of previous application versions.

#### 4.4.1.3. Multithreading

The software currently uses a single main thread. The main downside of using only one thread is to generate the STK reports in DOCX format or to import the templates of the same data format into the application. The large number of API calls needed for even a single STK report or template results in a high workload on the thread. This leads to the inaccessibility of the other functionalities of the software while the import of the STK DOCX templates or export of STK documentations is processed. This can be overcome by using separate threads that are responsible for importing the STK templates and reporting of the respective STKs. If multiple threads are added to this software, the developer must take thread safety into account when dealing with data that may be accessible by different threads at the same time.

### 4.4.2. Important features regarding the functionality

#### 4.4.2.1. Additional recurrent testing options

In the future, this application should also be able to cover the capturing of STKs of medical electrical systems that contain at least one active medical device as well as other medical or non-medical electrical devices. It should be also possible to capture MTKs. These additional inspections should be exported as protocols too. The application should also be able to cover the creation of a testing summary of an entire inspection. This should include all covered recurrent tests that have been captured.

#### 4.4.2.2. Electrical safety parameters

At this first version of the application, the electrical safety parameters of EN 60601 and EN 62353 must be filled in manually. Ideally, the application should be able to establish a connection to an automatic testing device in which the measured values are to be automatically entered into the corresponding safety

parameter cells. Another feature should be the automatic evaluation of the deficiency level based on measurements and their limits. Functionality for statistical trend analysis of the electrical safety parameters should also be added to previously captured STKs of the same medical device.

#### 4.4.2.3. Versioning

It is currently possible to reload the DOCX STK templates into their application-internal representation, although STK documentations might be exported incompletely, because a tracking of changes is not implemented yet. For this reason, a versioning feature must be implemented, in which older template versions are stored as well - so that every completed inspection (regardless of how old) can be exported.

# 5. Conclusion

In conclusion to the findings of this master's thesis, all aims and requirements were met - resulting in an extendable and user-friendly software for the documentation of STKs of active medical devices according to the Medical Device Operator Ordinance.

The first step was to analyze the DOCX templates provided by the HCE through the PMG and to plan the UI/UX design in order to gain a basic understanding of the work processes and the environment for capturing and documenting STKs. This led to the cultivation of an UL as an important part of DDD. The analytical model obtained through the use of UL during the UI/UX design phase was the basis for designing the domain model, the heart of the application and the foundation for the software design in general. By using strategic patterns alongside DDD, the software design got more and more shape up to its current version. The use of DI added additional maintainability to the software design and loosely coupled the three layers of the software architecture:BLL, DAL and GUI.

Further features and functions need to be added for this application to be fully applicable. However, the foundation for maintainability and expendability is laid by the attempts of this master's thesis.

# Bibliography


[1] European Testing Center of Medical Devices at the Institute of Health Care Engineering. *European Testing Center of Medical Devices*. 2019. URL: https://www.tugraz.at/en/arbeitsgruppen/pmg/home/ (visited on 10/18/2019) (cit. on p. 1).

[2] Bundesministerium für Digitalisierung und Wirtschaftsstandort. *Verordnung der Bundesministerin für Gesundheit, Familie und Jugend über das Errichten, Betreiben, Anwenden und Instandhalten von Medizinprodukten in Einrichtungen des Gesundheitswesens (Medizinproduktebetreiberverordnung – MPBV)*. 2019. URL: https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=20005279 (visited on 10/10/2019) (cit. on pp. 2, 3, 5).

[3] *Council Directive 93/42/EEC concerning medical devices*. Directive. THE COUNCIL OF THE EUROPEAN COMMUNITIES, 1993 (cit. on p. 4).

[4] *Medical electrical equipment – Recurrent test and test after repair of medical electrical equipment (IEC 62353:2014)*. Standard. OVE Austrian Electrotechnical Association, Edition: 2015-11-01 (cit. on pp. 4–6).

[5] *Medical electrical equipment - Part 1: General requirements for basic safety and essential performance (IEC 60601-1:2005 + Corr.:2006 + Corr.:2007 + A1:2012)*. Standard. OVE Austrian Electrotechnical Association, Edition: 2014-02-01 (cit. on p. 5).

[6] *Medical device software – Software life-cycle processes (IEC 62304:2006 + A1:2015)*. Standard. OVE Austrian Electrotechnical Association, Edition 2016-11-01 (cit. on p. 6).

[7] The Qt Company. *Qt Documentation*. 2020. URL: https://doc.qt.io/ (visited on 01/22/2020) (cit. on p. 11).

[8] The Qt Company. *Signals & Slots*. 2020. URL: https://doc.qt.io/qt-5/signalsandslots.html (visited on 01/24/2020) (cit. on p. 12).

[9] The Qt Company. *Overview - QML and C++ Integration*. 2020. URL: https://doc.qt.io/qt-5/qtqml-cppintegration-overview.html (visited on 01/24/2020) (cit. on p. 12).

[10] The Qt Company. *Exposing Attributes of C++ Types to QML*. 2020. URL: https://doc.qt.io/qt-5/qtqml-cppintegration-exposecppattributes.html (visited on 01/25/2020) (cit. on p. 12).

[11] The Qt Company. *Data Type Conversion Between QML and C++*. 2020. URL: https://doc.qt.io/qt-5/qtqml-cppintegration-data.html (visited on 01/25/2020) (cit. on p. 13).

[12] Steve Krug. *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability (3rd Edition)*. New Riders, 2014. ISBN: 0321965515 (cit. on pp. 16, 17, 21).

[13] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2003. ISBN: 0321125215 (cit. on pp. 22–24, 42).

[14] Scott Millett and Nick Tune. *Patterns, Principles, and Practices of Domain-Driven Design*. Wrox, 2015. ISBN: 1118714709 (cit. on pp. 22–24, 42).

[15] Steven van Deursen and Mark Seemann. *Dependency Injection Principles, Practices, and Patterns*. Manning Publications, 2019. ISBN: 161729473X (cit. on pp. 29–33, 40–42, 53).

[16] Rafal Malinowski. *Dependency injection framework for Qt*. 2017. URL: https://github.com/vogel/injeqt (visited on 03/17/2020) (cit. on p. 32).

[17] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. ISBN: 0201633612 (cit. on pp. 33–40).

[18] M. Fowler and D. Rice. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2003. ISBN: 9780321127426 (cit. on pp. 41, 42).

[19] Crupi John Alur Deepak and Malks Dan. *Core J2EE patterns : best practices and design strategies*. Prentice Hall PTR, 2003. ISBN: 0131422464 (cit. on pp. 43, 44).

[20] Oracle. *QAxBase Class*. 2019. URL: https://www.oracle.com/technetwork/java/dataaccessobject-138824.html (visited on 02/26/2020) (cit. on p. 44).

[21] Microsoft Corporation. *COM Technical Overview*. 2018. URL: https://docs.microsoft.com/en-us/windows/win32/com/com-technical-overview (visited on 02/09/2020) (cit. on p. 45).

[22] Microsoft Corporation. *IDispatch interface*. 2018. URL: https://docs.microsoft.com/de-de/windows/win32/api/oaidl/nn-oaidl-idispatch (visited on 02/10/2020) (cit. on p. 46).

[23] The Qt Company. *QAxObject Class*. 2020. URL: https://doc.qt.io/qt-5/qaxobject.html (visited on 02/07/2020) (cit. on p. 46).

[24] The Qt Company. *QAxBase Class*. 2020. URL: https://doc.qt.io/qt-5/qaxbase.html (visited on 02/07/2020) (cit. on p. 46).

[25] Microsoft Corporation. *Word VBA reference*. 2018. URL: https://docs.microsoft.com/en-us/office/vba/api/overview/word (visited on 02/04/2020) (cit. on pp. 46, 48, 49, 66).

[26] Microsoft Corporation. *Word object model overview*. 2017. URL: https://docs.microsoft.com/en-us/visualstudio/vsto/word-object-model-overview?view=vs-2019 (visited on 02/04/2020) (cit. on pp. 47, 48).

[27] The Qt Company. *Model/View Programming*. 2020. URL: https://doc.qt.io/qt-5/model-view-programming.html (visited on 01/25/2020) (cit. on pp. 55, 56).

[28] The Qt Company. *StackView QML Type*. 2020. URL: https://doc.qt.io/qt-5/qml-qtquick-controls-stackview.html (visited on 04/09/2020) (cit. on p. 57).

[29] The Qt Company. *Item QML Type*. 2020. URL: https://doc.qt.io/qt-5/qml-qtquick-item.html (visited on 04/09/2020) (cit. on p. 57).

[30] The Qt Company. *Transition QML Type*. 2020. URL: https://doc.qt.io/qt-5/qml-qtquick-transition.html (visited on 04/09/2020) (cit. on p. 57).

[31] The Qt Company. *FileDialog QML Type*. 2020. URL: https://doc.qt.io/qt-5/qml-qtquick-dialogs-filedialog.html (visited on 04/09/2020) (cit. on p. 58).

[32] The Qt Company. *Supported HTML Subset*. 2020. URL: https://doc.qt.io/qt-5/richtext-html-subset.html (visited on 04/08/2020) (cit. on p. 66).

# Appendix

# Appendix A.

# List Of Abbreviations

**API** Application Programming Interface

**BLL** Business Logic Layer

**COM** Component Object Model

**CROOT** Composition Root

**DAL** Data Access Layer

**DAO** Data Access Object

**DDD** Domain Driven Design

**DI** Dependency Injection

**DM** Domain Model

**GCC** GNU Compiler Collection

**GOF** Gang of Four

**GUI** Graphical User Interface

**HCE** Institute of Health Care Engineering

**IDE** Integrated Development Environment

**MPBV** Medical Device Operator Ordinance

**MTK** Recurrent Metrological Test

**MinGW** Minimalist GNU for Windows

**MWOM** Microsoft Word Object Model

**PMG** European Testing Center of Medical Devices

**STK** Recurrent Safety Test

**UL** Ubiquitous Language

**URI** Uniform Resource Identifier

**UUID** Universally Unique Identifier

**VBA** Visual Basic for Applications

# Appendix B.

# Recurrent safety test protocol templates

STAATLICH AKKREDITIERTE
**EUROPAPRÜFSTELLE FÜR MEDIZINPRODUKTE**
am Institut für Health Care Engineering

| | |
|---|---|
| **Referenz-Nr.** | |
| **Kunde** | |
| **Med. Bereich** | |
| Inventar-Nr. | |

# Wiederkehrende Prüfung elektromedizinischer Geräte nach EN 62353

| | | | |
|---|---|---|---|
| Gerät | | | |
| Typ | | Serien-Nr. | |
| Hersteller | | | |
| Zubehör | | | |
| | | | |
| **Datum** | | Prüfer | |

| Prüfintervall | 6M | 12M | 24M | 36M |
|---|---|---|---|---|
| Anwend.-Teil | B | BF | CF | keiner |
| Schutzklasse | I | II | III | Batterie |
| Feuchteschutz | IPX 0 | IP | | |
| Ex.-Schutz | 0 | AP | AP G | |

| Prüfpunkt | | | n. z. | o. k. | M 1 | M 2 | M 3 | Ergebnis | Bemerkungen |
|---|---|---|---|---|---|---|---|---|---|
| **Sichtprüfung außen** | | | | | | | | | |
| Gebrauchsanweisung *(zusätzliche Prüfpunkte notwendig)* | | | | | | | | | |
| Überprüfungshinweise in Gebrauchsanweisung | | | | | | | | | |
| Typenschild | | | | | | | | | |
| Hersteller | | | | | | | | | |
| CE- Kennzeichnung *(ggf. mit Kenn- Nr.)* | | | | | | | | | |
| Warnhinweise | | | | | | | | | |
| Gerät *(Standfestigkeit, Beschädigung, Betriebsart)* | | | | | | | | | |
| Gehäuse *(Berührschutz, Verschmutzung, Feuchtigkeitsschutz)* | | | | | | | | | |
| Netzstecker *(Adernendhülsen, nacheil.SL, Isolation, Zugentlastung)* | | | | | | | | | |
| Netzleitungsisolation | | | | | | | | | |
| Knickschutz | | r > 1,5D | | | | | | | |
| Zugentlastung | | | | | | | | | |
| Sicherungshalter (*Kennzeichnung, Berührschutz)* | | | | | | | | | |
| Sicherung(en) *(Anzahl, Nennwert)* | | SKI: allpolig SKII: einpolig | | | | | | | |
| Netzschalter *(Kennzeichnung, Schaltrichtung, Leuchte)* | | | | | | | | | |
| Patientenanschlüsse *(unverwechselbar)* | | | | | | | | | |
| Kennzeichnung Pat.-Anschl. *(AT-Type, GA-Hinweis)* | | | | | | | | | |
| EKG Kabel (*Steckvorr. 0.5mm Luftstrecke bei patientennahem Ende)* | | | | | | | | | |
| Einstellteile *(Kennzeichnung, Skalenposition, Betätigungsschutz)* | | | | | | | | | |
| Zubehör *(vollzählig, mech. i.O.)* | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| **Sicherheitstechnische Parameter** | | | | | | | | | |
| $R_{SL}$ ☐ *(Netzstecker-Gehäuse)* ☐ *(Gerätestecker-Gehäuse)* | | ≤ 0,3Ω | | | | | | **mΩ** | |
| Erdableitstrom | *(NC)* | ≤ 5mA | | | | | | **µA** | |
| Erdableitstrom | *(SFC)* | ≤ 10mA | | | | | | **µA** | |
| Berührungsstrom | *(NC)* | ≤ 100µA | | | | | | **µA** | |
| Berührungsstrom | *(SFC)* | ≤ 500µA | | | | | | **µA** | |
| Patientenableitstrom | *(NC)* | B,BF,CF:≤ 10µA DC | B,BF:≤ 100µA AC CF: ≤ 10µA AC | | | | | **µA** | |
| Patientenableitstrom | *(SFC)* | B,BF,CF: ≤ 50µA DC | B,BF:≤ 500µA AC CF: ≤ 50µA AC | | | | | **µA** | |
| Patientenableitstrom U-AWT *(SFC)* | | | B,BF:≤ 5mA CF: ≤ 50µA | | | | | **µA** | |
| Patientenhilfsstrom | *(NC)* | B,BF,CF:≤ 10µA DC | B,BF:≤ 100µA AC CF: ≤ 10µA AC | | | | | **µA** | |
| Patientenhilfsstrom | *(SFC)* | B,BF,CF: ≤ 50µA DC | B,BF:≤ 500µA AC CF: ≤ 50µA AC | | | | | **µA** | |
| **Funktion** | | | | | | | | | |
| Netzkontroll- Leuchte | | | | | | | | | |
| Ausgangskontroll- Leuchte | | | | | | | | | |
| Akku- Ladekontroll- Leuchte | | | | | | | | | |
| Akku- (Unterspannungs)- Kontroll- Leuchte | | | | | | | | | |
| Kontroll- Leuchte: innere Glüh- Heizelemente | | | | | | | | | |
| Zustand n. Spannungswiederkehr/Funktionsprüfung | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| **Gerätezustand** | | | | | | | | Unterschrift: | |

**o.k.**...Gerät in Ordnung, **n.z.**...nicht zutreffend, Mängelstufen: **M1**…tolerierbar, **M2**...mittelfristig zu beheben, **M3**...gefährlich

STAATLICH AKKREDITIERTE
**EUROPAPRÜFSTELLE FÜR MEDIZINPRODUKTE**
am Institut für Health Care Engineering

| | |
|---|---|
| **Referenz-Nr.** | |
| **Kunde** | |
| **Med. Bereich** | |
| Inventar-Nr. | |

# Wiederkehrende Prüfung elektromedizinischer Geräte nach EN 62353

| | | | |
|---|---|---|---|
| Gerät | | | |
| Typ | | Serien-Nr. | |
| Hersteller | | | |
| Zubehör | | | |
| | | | |
| **Datum** | | Prüfer | |

| Prüfintervall | 6M | 12M | 24M | 36M |
|---|---|---|---|---|
| Anwend.-Teil | B | BF | CF | keiner |
| Schutzklasse | I | II | III | Batterie |
| Feuchteschutz | IPX 0 | IP | | |
| Ex.-Schutz | 0 | AP | AP G | |

| Prüfpunkt | | n. z. | o. k. | M 1 | M 2 | M 3 | Ergebnis | Bemerkungen |
|---|---|---|---|---|---|---|---|---|
| **Sichtprüfung außen** | | | | | | | | |
| Gebrauchsanweisung *(zusätzliche Prüfpunkte notwendig)* | | | | | | | | |
| Überprüfungshinweise in Gebrauchsanweisung | | | | | | | | |
| Typenschild | | | | | | | | |
| Hersteller | | | | | | | | |
| CE- Kennzeichnung *(ggf. mit Kenn- Nr.)* | | | | | | | | |
| Warnhinweise | | | | | | | | |
| Gerät *(Standfestigkeit, Beschädigung, Betriebsart)* | | | | | | | | |
| Gehäuse *(Berührschutz, Verschmutzung, Feuchtigkeitsschutz)* | | | | | | | | |
| Netzstecker *(Adernendhülsen, nacheil.SL, Isolation, Zugentlastung)* | | | | | | | | |
| Netzleitungsisolation | | | | | | | | |
| Knickschutz | r > 1,5D | | | | | | | |
| Zugentlastung | | | | | | | | |
| Sicherungshalter (*Kennzeichnung, Berührschutz)* | | | | | | | | |
| Sicherung(en) *(Anzahl, Nennwert)* | SKI: allpolig SKII: einpolig | | | | | | | |
| Netzschalter *(Kennzeichnung, Schaltrichtung, Leuchte)* | | | | | | | | |
| Patientenanschlüsse *(unverwechselbar)* | | | | | | | | |
| Kennzeichnung Pat.-Anschl. *(AT-Type, GA-Hinweis)* | | | | | | | | |
| EKG Kabel (*Steckvorr. 0.5mm Luftstrecke bei patientennahem Ende)* | | | | | | | | |
| Einstellteile *(Kennzeichnung, Skalenposition, Betätigungsschutz)* | | | | | | | | |
| Zubehör *(vollzählig, mech. i.O.)* | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| **Sicherheitstechnische Parameter** | | | | | | | | |
| $R_{SL}$ ☐ *(Netzstecker-Gehäuse)* ☐ *(Gerätestecker-Gehäuse)* | ≤ 0,3Ω | | | | | | **mΩ** | |
| Geräteableitstrom (Ersatzmessung) | SKI: max. 1mA | | | | | | **µA** | |
| Geräteableitstrom (Ersatzmessung) | SKII: max. 0,5mA | | | | | | **µA** | |
| **Geräteableitstrom *(Direkt-/Differenzstrommess.)*** | SKI: max. 0,5mA | | | | | | **µA** | |
| **Geräteableitstrom *(Direkt-/Differenzstrommess.)*** | SKII: max. 0,1mA | | | | | | **µA** | |
| Ableitstrom vom Anwendungsteil | BF: max. 5mA CF: max. 50µA | | | | | | **µA** | |
| **Funktion** | | | | | | | | |
| Netzkontroll- Leuchte | | | | | | | | |
| Ausgangskontroll- Leuchte | | | | | | | | |
| Akku- Ladekontroll- Leuchte | | | | | | | | |
| Akku- (Unterspannungs)- Kontroll- Leuchte | | | | | | | | |
| Kontroll- Leuchte: innere Glüh- Heizelemente | | | | | | | | |
| Zustand n. Spannungswiederkehr/Funktionsprüfung | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| **Gerätezustand** | | | | | | | Unterschrift: | |

**o.k.**...Gerät in Ordnung, **n.z.**...nicht zutreffend, Mängelstufen: **M1**…tolerierbar, **M2**...mittelfristig zu beheben, **M3**...gefährlich

STAATLICH AKKREDITIERTE
**EUROPAPRÜFSTELLE FÜR MEDIZINPRODUKTE**
am Institut für Health Care Engineering

| | |
|---|---|
| **Referenz-Nr.** | |
| **Kunde** | |
| **Med. Bereich** | |
| Inventar-Nr. | |

## Wiederkehrende Prüfung
# Lasergeräte nach EN 62353

| | | | |
|---|---|---|---|
| Gerät | Lasergerät | | |
| Typ | | Serien-Nr. | |
| Hersteller | | | |
| Zubehör | ☐ *Lichtleitfaser* ☐ *Handstück* ☐ *Fußschalter* ☐ *Spitzen* ☐ *Schutzbrille* | | |
| | | | |
| **Datum** | | Prüfer | |

| Prüfintervall | 6M | 12M | 24M | 36M |
|---|---|---|---|---|
| Anwend.-Teil | B | BF | CF | keiner |
| Schutzklasse | I | II | III | Batterie |
| Feuchteschutz | IPX 0 | IP | | |
| Ex.-Schutz | 0 | AP | AP G | |

| Prüfpunkt | | | | n. z. | o. k. | M 1 | M 2 | M 3 | Ergebnis | Bemerkungen |
|---|---|---|---|---|---|---|---|---|---|---|
| **Sichtprüfung außen** | | | | | | | | | | |
| Gebrauchsanweisung *(zusätzliche Prüfpunkte notwendig)* | | | | | | | | | | |
| Typenschild | | | | | | | | | | |
| Hersteller | | | | | | | | | | |
| CE- Kennzeichnung *(ggf. mit Kenn- Nr.)* | | | | | | | | | | |
| Warnhinweise *(**Laserstrahlung, -Austrittsöffnung**)* | | | | | | | | | | |
| Gerät *(Standfestigkeit, Beschädigung, Betriebsart)* | | | | | | | | | | |
| Gehäuse *(Berührschutz, Verschmutzung, Feuchtigkeitsschutz)* | | | | | | | | | | |
| Netzstecker *(Adernendhülsen, nacheil.SL, Isolation, Zugentlastung)* | | | | | | | | | | |
| Netzleitungsisolation | | | | | | | | | | |
| Knickschutz | | | r > 1,5D | | | | | | | |
| Zugentlastung | | | | | | | | | | |
| Sicherungshalter (*Kennzeichnung, Berührschutz)* | | | | | | | | | | |
| Sicherung(en) *(Anzahl, Nennwert)* | | | SKI: allpolig SKII: einpolig | | | | | | | |
| Netzschalter *(Kennzeichnung, Schaltrichtung, Leuchte)* | | | | | | | | | | |
| **Schlüsselschalter** | | | Klasse 3B, 4 | | | | | | | |
| Patientenanschlüsse *(unverwechselbar)* | | | | | | | | | | |
| Kennzeichnung Pat.-Anschl. *(AT-Type, GA-Hinweis)* | | | | | | | | | | |
| Einstellteile *(Kennzeichnung, Skalenposition, Betätigungsschutz)* | | | | | | | | | | |
| Zubehör *(**Lichtleitfaser, Applikator**, vollzählig, mech. i.O.)* | | | | | | | | | | |
| Fußschalter (***betätigungsgeschützt, wasserdicht***) | | | | | | | | | | |
| **Schutzbrille(n)** (für λ geeignet, Leuchten sichtbar) | | | Klasse 3B, 4 | | | | | | | |
| | | | | | | | | | | |
| **Sicherheitstechnische Parameter** | | | | | | | | | | |
| $R_{SL}$ ☐ *(Netzstecker-Gehäuse)* ☐ *(Gerätestecker-Gehäuse)* | | | ≤ 0,2Ω | | | | | | **mΩ** | |
| Erdableitstrom *(NC)* | | | < 5mA | | | | | | **µA** | |
| Erdableitstrom *(SFC)* | | | < 10mA | | | | | | **µA** | |
| Berührungsstrom *(NC)* | | | < 100µA | | | | | | **µA** | |
| Berührungsstrom *(SFC)* | | | < 500µA | | | | | | **µA** | |
| Patientenableitstrom *(NC)* | | B,BF,CF:≤ 10µA DC | B,BF:≤ 100µA AC CF: ≤ 10µA AC | | | | | | **µA** | |
| Patientenableitstrom *(SFC)* | | B,BF,CF: ≤ 50µA DC | B,BF:≤ 500µA AC CF: ≤ 50µA AC | | | | | | **µA** | |
| Patientenableitstrom U-AWT *(SFC)* | | | B,BF:≤ 5mA CF: ≤ 50µA | | | | | | **µA** | |
| Patientenhilfsstrom *(NC)* | | B,BF,CF:≤ 10µA DC | B,BF:≤ 100µA AC CF: ≤ 10µA AC | | | | | | **µA** | |
| Patientenhilfsstrom *(SFC)* | | B,BF,CF: ≤ 50µA DC | B,BF:≤ 500µA AC CF: ≤ 50µA AC | | | | | | **µA** | |
| **Funktion** | | | | | | | | | | |
| Netzkontroll- Leuchte | | | | | | | | | | |
| Ausgangsaktivierungs- Kontroll- Leuchte | | | | | | | | | | |
| Akku- Ladekontroll- Leuchte | | | | | | | | | | |
| Akku- (Unterspannungs)- Kontroll- Leuchte | | | | | | | | | | |
| **Emissionsalarm** (optisch/akustisch) | | | Klasse 3B,4 | | | | | | | |
| Schaltfolge ***(Aus-Standby-Bereit-Aktivierung)*** | | | | | | | | | | |
| **Alarm/Abschaltung bei Unterbr. der opt. Verb.** | | | Klasse 3B,4 | | | | | | | |
| **Laserbereit- Alarm** *(2s vor Aktivierung)* | | | Kl.2,3A,3B,4 | | | | | | | |
| **Notaus- Taster** | | | Klasse 3B,4 | | | | | | | |
| **Türkontaktschalter** *(ausgen. Laserpens)* | | | Klasse 3B,4 | | | | | | | |
| **stand nach Spannungswiederkehr** *(ein/aus)* | | | | | | | | | | |
| **Laserausgangsleistung** | $P_{max}$ | W | ± 20% | | | | | | **W** | |
| | $P_{mittel}$ | W | ± 20% | | | | | | **W** | |
| | $P_{min}$ | W | ± 20% | | | | | | **W** | |
| **Gerätezustand** | | | | | | | | | Unterschrift: | |

**o.k.**...Gerät in Ordnung, **n.z.**...nicht zutreffend, Mängelstufen: **M1**…tolerierbar, **M2**...mittelfristig zu beheben, **M3**...gefährlich

STAATLICH AKKREDITIERTE
**EUROPAPRÜFSTELLE FÜR MEDIZINPRODUKTE**
am Institut für Health Care Engineering

| | |
|---|---|
| **Referenz-Nr.** | |
| **Kunde** | |
| **Med. Bereich** | |
| Inventar-Nr. | |

## Wiederkehrende Prüfung
# Lasergeräte nach EN 62353

| | | | |
|---|---|---|---|
| Gerät | Lasergerät | | |
| Typ | | Serien-Nr. | |
| Hersteller | | | |
| Zubehör | ☐ *Lichtleitfaser* ☐ *Handstück* ☐ *Fußschalter* ☐ *Spitzen* ☐ *Schutzbrille* | | |
| | | | |
| **Datum** | | Prüfer | |

| Prüfintervall | 6M | 12M | 24M | 36M |
|---|---|---|---|---|
| Anwend.-Teil | B | BF | CF | keiner |
| Schutzklasse | I | II | III | Batterie |
| Feuchteschutz | IPX 0 | IP | | |
| Ex.-Schutz | 0 | AP | AP G | |

| Prüfpunkt | | n. z. | o. k. | M 1 | M 2 | M 3 | Ergebnis | Bemerkungen |
|---|---|---|---|---|---|---|---|---|
| **Sichtprüfung außen** | | | | | | | | |
| Gebrauchsanweisung *(zusätzliche Prüfpunkte notwendig)* | | | | | | | | |
| Typenschild | | | | | | | | |
| Hersteller | | | | | | | | |
| CE- Kennzeichnung *(ggf. mit Kenn- Nr.)* | | | | | | | | |
| Warnhinweise *(**Laserstrahlung, -Austrittsöffnung**)* | | | | | | | | |
| Gerät *(Standfestigkeit, Beschädigung, Betriebsart)* | | | | | | | | |
| Gehäuse *(Berührschutz, Verschmutzung, Feuchtigkeitsschutz)* | | | | | | | | |
| Netzstecker *(Adernendhülsen, nacheil.SL, Isolation, Zugentlastung)* | | | | | | | | |
| Netzleitungsisolation | | | | | | | | |
| Knickschutz | r > 1,5D | | | | | | | |
| Zugentlastung | | | | | | | | |
| Sicherungshalter (*Kennzeichnung, Berührschutz)* | | | | | | | | |
| Sicherung(en) *(Anzahl, Nennwert)* | SKI: allpolig SKII: einpolig | | | | | | | |
| Netzschalter *(Kennzeichnung, Schaltrichtung, Leuchte)* | | | | | | | | |
| **Schlüsselschalter** | Klasse 3B, 4 | | | | | | | |
| Patientenanschlüsse *(unverwechselbar)* | | | | | | | | |
| Kennzeichnung Pat.-Anschl. *(AT-Type, GA-Hinweis)* | | | | | | | | |
| Einstellteile *(Kennzeichnung, Skalenposition, Betätigungsschutz)* | | | | | | | | |
| Zubehör *(**Lichtleitfaser, Applikator**, vollzählig, mech. i.O.)* | | | | | | | | |
| Fußschalter (***betätigungsgeschützt, wasserdicht***) | | | | | | | | |
| **Schutzbrille(n)** (für λ geeignet, Leuchten sichtbar) | Klasse 3B, 4 | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| **Sicherheitstechnische Parameter** | | | | | | | | |
| $R_{SL}$ ☐ *(Netzstecker-Gehäuse)* ☐ *(Gerätestecker-Gehäuse)* | ≤ 0,3Ω | | | | | | **mΩ** | |
| Geräteableitstrom *(Ersatzmessung)* | SKI: max. 1mA | | | | | | **µA** | |
| Geräteableitstrom *(Ersatzmessung)* | SKII: max. 0,5mA | | | | | | **µA** | |
| **Geräteableitstrom *(Direkt-/Differenzstrommess.)*** | SKI: max. 0,5mA | | | | | | **µA** | |
| **Geräteableitstrom *(Direkt-/Differenzstrommess.)*** | SKII: max. 0,1mA | | | | | | **µA** | |
| Ableitstrom vom Anwendungsteil | BF: max. 5mA CF: max. 50µA | | | | | | **µA** | |
| **Funktion** | | | | | | | | |
| Netzkontroll- Leuchte | | | | | | | | |
| Ausgangsaktivierungs- Kontroll- Leuchte | | | | | | | | |
| Akku- Ladekontroll- Leuchte | | | | | | | | |
| Akku- (Unterspannungs)- Kontroll- Leuchte | | | | | | | | |
| **Emissionsalarm** (optisch/akustisch) | Klasse 3B,4 | | | | | | | |
| Schaltfolge ***(Aus-Standby-Bereit-Aktivierung)*** | | | | | | | | |
| **Alarm/Abschaltung bei Unterbr. der opt. Verb.** | Klasse 3B,4 | | | | | | | |
| **Laserbereit- Alarm** *(2s vor Aktivierung)* | Kl.2,3A,3B,4 | | | | | | | |
| **Notaus- Taster** | Klasse 3B,4 | | | | | | | |
| **Türkontaktschalter** *(ausgen. Laserpens)* | Klasse 3B,4 | | | | | | | |
| **stand nach Spannungswiederkehr** *(ein/aus)* | | | | | | | | |
| **Laserausgangsleistung** | $P_{max}$ W ± 20% | | | | | | **W** | |
| | $P_{mittel}$ W ± 20% | | | | | | **W** | |
| | $P_{min}$ W ± 20% | | | | | | **W** | |
| | | | | | | | | |
| | | | | | | | | |
| **Gerätezustand** | | | | | | | Unterschrift: | |

**o.k.**...Gerät in Ordnung, **n.z.**...nicht zutreffend, Mängelstufen: **M1**…tolerierbar, **M2**...mittelfristig zu beheben, **M3**...gefährlich

STAATLICH AKKREDITIERTE
**EUROPAPRÜFSTELLE FÜR MEDIZINPRODUKTE**
am Institut für Health Care Engineering

| | |
|---|---|
| **Referenz-Nr.** | |
| **Kunde** | |
| **Med. Bereich** | |
| Inventar-Nr. | |

## Wiederkehrende Prüfung
# Reizstromgeräte nach EN 62353

| | | | |
|---|---|---|---|
| Gerät | Reizstromgerät | | |
| Typ | | Serien-Nr. | |
| Hersteller | | | |
| Zubehör | ☐ *Elektroden* ☐ *Saugelektroden* | | |
| | | | |
| **Datum** | | Prüfer | |

| Prüfintervall | 6M | 12M | 24M | 36M |
|---|---|---|---|---|
| Anwend.-Teil | **B** | **BF** | **CF** | **keiner** |
| Schutzklasse | **I** | **II** | **III** | **Batterie** |
| Feuchteschutz | **IPX 0** | **IP** | | |
| Ex.-Schutz | **0** | **AP** | **AP G** | |

| Prüfpunkt | | | n. z. | o. k. | M 1 | M 2 | M 3 | Ergebnis | Bemerkungen |
|---|---|---|---|---|---|---|---|---|---|
| **Sichtprüfung außen** | | | | | | | | | |
| Gebrauchsanweisung *(zusätzliche Prüfpunkte notwendig)* | | | | | | | | | |
| Typenschild | | | | | | | | | |
| Hersteller | | | | | | | | | |
| CE- Kennzeichnung *(ggf. mit Kenn- Nr.)* | | | | | | | | | |
| Warnhinweise | | | | | | | | | |
| Gerät *(Standfestigkeit, Beschädigung, Betriebsart)* | | | | | | | | | |
| Gehäuse *(Berührschutz, Verschmutzung, Feuchtigkeitsschutz)* | | | | | | | | | |
| Netzstecker *(Adernendhülsen, nacheil.SL, Isolation, Zugentlastung)* | | | | | | | | | |
| Netzleitungsisolation | | | | | | | | | |
| Knickschutz | | r > 1,5D | | | | | | | |
| Zugentlastung | | | | | | | | | |
| Sicherungshalter (*Kennzeichnung, Berührschutz)* | | | | | | | | | |
| Sicherung(en) *(Anzahl, Nennwert)* | | SKI: allpolig<br>SKII: einpolig | | | | | | | |
| Netzschalter *(Kennzeichnung, Schaltrichtung, Leuchte)* | | | | | | | | | |
| Patientenanschlüsse *(unverwechselbar)* | | | | | | | | | |
| Kennzeichnung Pat.-Anschl. *(AT-Type, GA-Hinweis)* | | | | | | | | | |
| Einstellteile *(Kennzeichnung, Skalenposition, Betätigungsschutz)* | | | | | | | | | |
| Zubehör *(vollzählig, mech. i.O.)* | | | | | | | | | |
| | | | | | | | | | |
| **Sicherheitstechnische Parameter** | | | | | | | | | |
| $R_{SL}$ ☐ *(Netzstecker-Gehäuse)* ☐ *(Gerätestecker-Gehäuse)* | | | ≤ 0,2Ω | | | | | | **mΩ** | |
| Erdableitstrom | *(NC)* | | < 5mA | | | | | | **µA** | |
| Erdableitstrom | *(SFC)* | | < 10mA | | | | | | **µA** | |
| Berührungsstrom | *(NC)* | | < 100µA | | | | | | **µA** | |
| Berührungsstrom | *(SFC)* | | < 500µA | | | | | | **µA** | |
| Patientenableitstrom | *(NC)* | B,BF,CF:≤ 10µA DC | B,BF:≤ 100µA AC<br>CF: ≤ 10µA AC | | | | | | **µA** | |
| Patientenableitstrom | *(SFC)* | B,BF,CF: ≤ 50µA DC | B,BF:≤ 500µA AC<br>CF: ≤ 50µA AC | | | | | | **µA** | |
| Patientenableitstrom U-AWT *(SFC)* | | | B,BF:≤ 5mA<br>CF: ≤ 50µA | | | | | | **µA** | |
| Patientenhilfsstrom | *(NC)* | B,BF,CF:≤ 10µA DC | B,BF:≤ 100µA AC<br>CF: ≤ 10µA AC | | | | | | **µA** | |
| Patientenhilfsstrom | *(SFC)* | B,BF,CF: ≤ 50µA DC | B,BF:≤ 500µA AC<br>CF: ≤ 50µA AC | | | | | | **µA** | |
| **Funktion** | | | | | | | | | |
| Netzkontroll- Leuchte | | | | | | | | | |
| **Ausgangskontroll-Leuchte *wenn Gerät unter Spannung*** | | | | | | | | | |
| Akku- Ladekontroll- Leuchte | | | | | | | | | |
| Akku- (Unterspannungs)- Kontroll- Leuchte | | | | | | | | | |
| **Anzeige, wenn $I_A$ > 10mA an 1kΩ Last** | | | | | | | | | |
| **Ausgangs- Stromform** | | | | | | | | | |
| **Ausgangs- Frequenz** | | | | | | | | | |
| **Ausgangsstrom- Maximum** | | | 80mA<br>=/~(f>400/<1,5kHz)<br>50mA~(f<400Hz)<br>100mA~(f>1,5kHz) | | | | | | |
| **Ausgangsstrom- Genauigkeit** bei $I_{max}$ | | | ΔI < 20% | | | | | | |
| **Ausgangsregler- Minimum- Stellung** | | | 0 bzw.<2%Max | | | | | | |
| **Ausgangsregelung** | | | Kont.od.Stufen<br><1mA/<1Veff | | | | | | |
| **Strom nach Einschalten bzw. Spgswiederkehr** | | | Minimum | | | | | | |
| **Vakuum** *(Saugelektroden werden angesaugt)* | | | | | | | | | |
| **Gerätezustand** | | | | | | | | Unterschrift: | |

**o.k.**...Gerät in Ordnung, **n.z.**....nicht zutreffend, Mängelstufen: **M1**…tolerierbar, **M2**...mittelfristig zu beheben, **M3**...gefährlich

STAATLICH AKKREDITIERTE
**EUROPAPRÜFSTELLE FÜR MEDIZINPRODUKTE**
am Institut für Health Care Engineering

| | |
|---|---|
| **Referenz-Nr.** | |
| **Kunde** | |
| **Med. Bereich** | |
| Inventar-Nr. | |

## Wiederkehrende Prüfung
# Reizstromgeräte nach EN 62353

| | | | |
|---|---|---|---|
| Gerät | Reizstromgerät | | |
| Typ | | Serien-Nr. | |
| Hersteller | | | |
| Zubehör | ☐ *Elektroden* ☐ *Saugelektroden* | | |
| | | | |
| **Datum** | | Prüfer | |

| Prüfintervall | 6M | 12M | 24M | 36M |
|---|---|---|---|---|
| Anwend.-Teil | **B** | **BF** | **CF** | **keiner** |
| Schutzklasse | **I** | **II** | **III** | **Batterie** |
| Feuchteschutz | **IPX 0** | **IP** | | |
| Ex.-Schutz | **0** | **AP** | **AP G** | |

| Prüfpunkt | | n. z. | o. k. | M 1 | M 2 | M 3 | Ergebnis | Bemerkungen |
|---|---|---|---|---|---|---|---|---|
| **Sichtprüfung außen** | | | | | | | | |
| Gebrauchsanweisung *(zusätzliche Prüfpunkte notwendig)* | | | | | | | | |
| Typenschild | | | | | | | | |
| Hersteller | | | | | | | | |
| CE- Kennzeichnung *(ggf. mit Kenn- Nr.)* | | | | | | | | |
| Warnhinweise | | | | | | | | |
| Gerät *(Standfestigkeit, Beschädigung, Betriebsart)* | | | | | | | | |
| Gehäuse *(Berührschutz, Verschmutzung, Feuchtigkeitsschutz)* | | | | | | | | |
| Netzstecker *(Adernendhülsen, nacheil.SL, Isolation, Zugentlastung)* | | | | | | | | |
| Netzleitungsisolation | | | | | | | | |
| Knickschutz | r > 1,5D | | | | | | | |
| Zugentlastung | | | | | | | | |
| Sicherungshalter (*Kennzeichnung, Berührschutz)* | | | | | | | | |
| Sicherung(en) *(Anzahl, Nennwert)* | SKI: allpolig SKII: einpolig | | | | | | | |
| Netzschalter *(Kennzeichnung, Schaltrichtung, Leuchte)* | | | | | | | | |
| Patientenanschlüsse *(unverwechselbar)* | | | | | | | | |
| Kennzeichnung Pat.-Anschl. *(AT-Type, GA-Hinweis)* | | | | | | | | |
| Einstellteile *(Kennzeichnung, Skalenposition, Betätigungsschutz)* | | | | | | | | |
| Zubehör *(vollzählig, mech. i.O.)* | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| **Sicherheitstechnische Parameter** | | | | | | | | |
| $R_{SL}$ ☐ *(Netzstecker-Gehäuse)* ☐ *(Gerätestecker-Gehäuse)* | ≤ 0,3Ω | | | | | | **mΩ** | |
| Geräteableitstrom *(Ersatzmessung)* | SKI: max. 1mA | | | | | | **µA** | |
| Geräteableitstrom *(Ersatzmessung)* | SKII: max. 0,5mA | | | | | | **µA** | |
| **Geräteableitstrom *(Direkt-/Differenzstrommess.)*** | SKI: max. 0,5mA | | | | | | **µA** | |
| **Geräteableitstrom *(Direkt-/Differenzstrommess.)*** | SKII: max. 0,1mA | | | | | | **µA** | |
| Ableitstrom vom Anwendungsteil | BF: max. 5mA CF: max. 50µA | | | | | | **µA** | |
| Geräteableitstrom *(Ersatzmessung)* | gleichmäßig | | | | | | | |
| **Funktion** | | | | | | | | |
| Netzkontroll- Leuchte | | | | | | | | |
| **Ausgangskontroll-Leuchte *wenn Gerät unter Spannung*** | | | | | | | | |
| Akku- Ladekontroll- Leuchte | | | | | | | | |
| Akku- (Unterspannungs)- Kontroll- Leuchte | | | | | | | | |
| **Anzeige, wenn $I_A$ > 10mA an 1kΩ Last** | | | | | | | | |
| **Ausgangs- Stromform** | | | | | | | | |
| **Ausgangs- Frequenz** | | | | | | | | |
| **Ausgangsstrom- Maximum** | 80mA =/~(f>400/<1,5kHz) 50mA~(f<400Hz) 100mA~(f>1,5kHz) | | | | | | | |
| **Ausgangsstrom- Genauigkeit** bei $I_{max}$ | ΔI < 20% | | | | | | | |
| **Ausgangsregler- Minimum- Stellung** | 0 bzw.<2%Max | | | | | | | |
| **Ausgangsregelung** | Kont.od.Stufe <1mA/<1Veff | | | | | | | |
| **Strom nach Einschalten bzw. Spgswiederkehr** | Minimum | | | | | | | |
| **Vakuum** *(Saugelektroden werden angesaugt)* | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| **Gerätezustand** | | | | | | | Unterschrift: | |

**o.k.**...Gerät in Ordnung, **n.z.**...nicht zutreffend, Mängelstufen: **M1**…tolerierbar, **M2**...mittelfristig zu beheben, **M3**...gefährlich

STAATLICH AKKREDITIERTE
**EUROPAPRÜFSTELLE FÜR MEDIZINPRODUKTE**
am Institut für Health Care Engineering

| | |
|---|---|
| **Referenz-Nr.** | |
| **Kunde** | |
| **Med. Bereich** | |
| Inventar-Nr. | |

## Wiederkehrende Prüfung
# HF-Chirurgiegeräte nach EN 62353

| | | | |
|---|---|---|---|
| Gerät | HF-Chirurgiegerät | | |
| Typ | | Serien-Nr. | |
| Hersteller | | | |
| Zubehör | ☐ *aktive Elektrode* ☐ *Neutralelektrode* ☐ *Fußschalter* ☐ *bipolar Elektrode* | | |
| | ☐ *keine Neutralelekt. (<50W)* | | |
| **Datum** | | Prüfer | |

| Prüfintervall | 6M | 12M | 24M | 36M |
|---|---|---|---|---|
| Anwend.-Teil | **B** | **BF** | **CF** | **keiner** |
| Schutzklasse | **I** | **II** | **III** | **Batterie** |
| Feuchteschutz | **IPX 0** | **IP** | | |
| Ex.-Schutz | **0** | **AP** | **AP G** | |

| **Prüfpunkt** | | | | **n. z.** | **o. k.** | **M 1** | **M 2** | **M 3** | **Ergebnis** | **Bemerkungen** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Sichtprüfung außen** | | | | | | | | | | |
| Gebrauchsanweisung *(zusätzliche Prüfpunkte notwendig)* | | | | | | | | | | |
| Typenschild | | | | | | | | | | |
| Hersteller | | | | | | | | | | |
| CE- Kennzeichnung *(ggf. mit Kenn- Nr.)* | | | | | | | | | | |
| **Warnhinweise** *(GA-Hinweis,* ***Erzeugung von HF-Feldern****)* | | | | | | | | | | |
| Gerät *(Standfestigkeit, Beschädigung, Betriebsart)* | | | | | | | | | | |
| Gehäuse *(Berührschutz, Verschmutzung, Feuchtigkeitsschutz)* | | | | | | | | | | |
| Netzstecker *(Adernendhülsen, nacheil.SL, Isolation, Zugentlastung)* | | | | | | | | | | |
| Netzleitungsisolation | | | | | | | | | | |
| Knickschutz | | | r > 1,5D | | | | | | | |
| Zugentlastung | | | | | | | | | | |
| Sicherungshalter (*Kennzeichnung, Berührschutz)* | | | | | | | | | | |
| Sicherung(en) *(Anzahl, Nennwert)* | | | SKI: allpolig<br>SKII: einpolig | | | | | | | |
| Netzschalter *(Kennzeichnung, Schaltrichtung, Leuchte)* | | | | | | | | | | |
| Patientenanschlüsse *(unverwechselbar)* | | | | | | | | | | |
| Kennzeichnung Pat.-Anschl. *(AT-Type, GA-Hinweis)* | | | | | | | | | | |
| **Einstellteile** *(Kennzeichnung, Skalenposition, Betätigungsschutz)* | | | | | | | | | | |
| **Fußschalter** ***(wasserdicht; mech. geschützt)*** | | | **IP Code:** | | | | | | | |
| **Aktive Elektrode, Neutralelektrode** *(mech. Zustand, Ablaufdatum)* | | | | | | | | | | |
| **Neutralelektrode:** ☐ **geerdet** ☐ **isoliert** | | | **Nicht starr geerdet** | | | | | | | |
| **Sicherheitstechnische Parameter** | | | | | | | | | | |
| $R_{SL}$ ☐ *(Netzstecker-Gehäuse)* ☐ *(Gerätestecker-Gehäuse)* | | | ≤ 0,2Ω | | | | | | **mΩ** | |
| Erdableitstrom *(NC)* | | | < 5mA | | | | | | **µA** | |
| Erdableitstrom *(SFC)* | | | < 10mA | | | | | | **µA** | |
| Berührungsstrom *(NC)* | | | < 100µA | | | | | | **µA** | |
| Berührungsstrom *(SFC)* | | | < 500µA | | | | | | **µA** | |
| Patientenableitstrom *(NC)* | | B,BF,CF:≤ 10µA DC | B,BF:≤ 100µA AC<br>CF: ≤ 10µA AC | | | | | | **µA** | |
| Patientenableitstrom *(SFC)* | | B,BF,CF: ≤ 50µA DC | B,BF:≤ 500µA AC<br>CF: ≤ 50µA AC | | | | | | **µA** | |
| Patientenableitstrom U-AWT *(SFC)* | | | B,BF:≤ 5mA<br>CF: ≤ 50µA | | | | | | **µA** | |
| Patientenhilfsstrom *(NC)* | | B,BF,CF:≤ 10µA DC | B,BF:≤ 100µA AC<br>CF: ≤ 10µA AC | | | | | | **µA** | |
| Patientenhilfsstrom *(SFC)* | | B,BF,CF: ≤ 50µA DC | B,BF:≤ 500µA AC<br>CF: ≤ 50µA AC | | | | | | **µA** | |
| Neutralelektrodenwiderstand *(bei Gummielektrode)* | | | gleichmäßig | | | | | | | |
| R= *(zwischen Elektrodenanschlüssen bei 500V)* | | | ≥2MΩ | | | | | | | |
| **Patientenableitstrom** ***(HF monopolar)*** | | | ≤150mA | | | | | | **mA** | |
| **Patientenableitstrom** ***(HF bipolar)*** | | | ≤1%$P_{max}$ @200Ω | | | | | | **mA** | |
| **Funktion** | | | | | | | | | | |
| Netzkontroll- Leuchte | | | | | | | | | | |
| **Aktivierungsalarm** ***(optisch, akustisch)*** | | | | | | | | | | |
| **Funktions- Voranzeige** | | | bei umschalt-barem Ausgang | | | | | | | |
| **Neutralelektr.unterbrechungs- Alarm** ***(rote Lampe)*** | | | bei $P_{max}$>50W | | | | | | | |
| **Neutralelektroden-Teilablösungsalarm** ***(optional)*** | | | bei $P_{max}$>50W | | | | | | | |
| **Zustand nach Spannungswiederkehr** ***(ein/aus)*** | | | Default Werte od. Minimum | | | | | | | |
| Ausgangsleistung *(Cut)* | $P_{Cut,max}$: | W | **≤400W**±20% | | | | | | **W** | |
| | $P_{Cut,mittel}$: | W | ± 20% | | | | | | **W** | |
| | 0,1 $P_{Cut,max}$: | W | ± 20% | | | | | | **W** | |
| Ausgangsleistung *(Coag)* | $P_{Coag,max}$: | W | ± 20% | | | | | | **W** | |
| | $P_{Coag,mittel}$: | W | ± 20% | | | | | | **W** | |
| | 0,1 $P_{Coag,max}$: | W | ± 20% | | | | | | **W** | |
| **Gerätezustand** | | | | | | | | | Unterschrift: | |

**o.k.**...Gerät in Ordnung, **n.z.**....nicht zutreffend, Mängelstufen: **M1**…tolerierbar, **M2**...mittelfristig zu beheben, **M3**...gefährlich

STAATLICH AKKREDITIERTE
**EUROPAPRÜFSTELLE FÜR MEDIZINPRODUKTE**
am Institut für Health Care Engineering

| | |
|---|---|
| **Referenz-Nr.** | |
| **Kunde** | |
| **Med. Bereich** | |
| Inventar-Nr. | |

## Wiederkehrende Prüfung
# HF-Chirurgiegeräte nach EN 62353

| | | | |
|---|---|---|---|
| Gerät | HF-Chirurgiegerät | | |
| Typ | | Serien-Nr. | |
| Hersteller | | | |
| Zubehör | ☐ *aktive Elektrode* ☐ *Neutralelektrode* ☐ *Fußschalter* ☐ *bipolar Elektrode* | | |
| | ☐ *keine Neutralelekt. (<50W)* | | |
| **Datum** | | Prüfer | |

| Prüfintervall | **6M** | **12M** | **24M** | **36M** |
|---|---|---|---|---|
| Anwend.-Teil | **B** | **BF** | **CF** | **keiner** |
| Schutzklasse | **I** | **II** | **III** | **Batterie** |
| Feuchteschutz | **IPX 0** | **IP** | | |
| Ex.-Schutz | **0** | **AP** | **AP G** | |

| **Prüfpunkt** | | | | **n. z.** | **o. k.** | **M 1** | **M 2** | **M 3** | **Ergebnis** | **Bemerkungen** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Sichtprüfung außen** | | | | | | | | | | |
| Gebrauchsanweisung *(zusätzliche Prüfpunkte notwendig)* | | | | | | | | | | |
| Typenschild | | | | | | | | | | |
| Hersteller | | | | | | | | | | |
| CE- Kennzeichnung *(ggf. mit Kenn- Nr.)* | | | | | | | | | | |
| **Warnhinweise** *(GA-Hinweis,* ***Erzeugung von HF-Feldern****)* | | | | | | | | | | |
| Gerät *(Standfestigkeit, Beschädigung, Betriebsart)* | | | | | | | | | | |
| Gehäuse *(Berührschutz, Verschmutzung, Feuchtigkeitsschutz)* | | | | | | | | | | |
| Netzstecker *(Adernendhülsen, nacheil.SL, Isolation, Zugentlastung)* | | | | | | | | | | |
| Netzleitungsisolation | | | | | | | | | | |
| Knickschutz | | | r > 1,5D | | | | | | | |
| Zugentlastung | | | | | | | | | | |
| Sicherungshalter (*Kennzeichnung, Berührschutz)* | | | | | | | | | | |
| Sicherung(en) *(Anzahl, Nennwert)* | | | SKI: allpolig<br>SKII: einpolig | | | | | | | |
| Netzschalter *(Kennzeichnung, Schaltrichtung, Leuchte)* | | | | | | | | | | |
| Patientenanschlüsse *(unverwechselbar)* | | | | | | | | | | |
| Kennzeichnung Pat.-Anschl. *(AT-Type, GA-Hinweis)* | | | | | | | | | | |
| **Einstellteile** *(Kennzeichnung, Skalenposition, Betätigungsschutz)* | | | | | | | | | | |
| **Fußschalter** ***(wasserdicht; mech. geschützt)*** | | | **IP Code:** | | | | | | | |
| **Aktive Elektrode, Neutralelektrode** *(mech. Zustand, Ablaufdatum)* | | | | | | | | | | |
| **Neutralelektrode:** ☐ **geerdet** ☐ **isoliert** | | | **Nicht starr geerdet** | | | | | | | |
| | | | | | | | | | | |
| **Sicherheitstechnische Parameter** | | | | | | | | | | |
| $R_{SL}$ ☐ *(Netzstecker-Gehäuse)* ☐ *(Gerätestecker-Gehäuse)* | | | ≤ 0,3Ω | | | | | | **mΩ** | |
| Geräteableitstrom *(Ersatzmessung)* | | | SKI: max. 1mA | | | | | | **µA** | |
| Geräteableitstrom *(Ersatzmessung)* | | | SKII:max.0,5mA | | | | | | **µA** | |
| **Geräteableitstrom** ***(Direkt-/Differenzstrommess.)*** | | | SKI: max. 0,5mA | | | | | | **µA** | |
| **Geräteableitstrom** ***(Direkt-/Differenzstrommess.)*** | | | SKII:max.0,1mA | | | | | | **µA** | |
| Ableitstrom vom Anwendungsteil | | | BF: max. 5mA<br>CF: max. 50µA | | | | | | **µA** | |
| Neutralelektrodenwiderstand *(bei Gummielektrode)* | | | gleichmäßig | | | | | | | |
| R= *(zwischen Elektrodenanschlüssen bei 500V)* | | | ≥2MΩ | | | | | | | |
| **Patientenableitstrom** ***(HF monopolar)*** | | | ≤150mA | | | | | | **mA** | |
| **Patientenableitstrom** ***(HF bipolar)*** | | | ≤1%$P_{max}$ @200Ω | | | | | | **mA** | |
| **Funktion** | | | | | | | | | | |
| Netzkontroll- Leuchte | | | | | | | | | | |
| **Aktivierungsalarm** ***(optisch, akustisch)*** | | | | | | | | | | |
| **Funktions- Voranzeige** | | | bei umschalt-barem Ausgang | | | | | | | |
| **Neutralelektr.unterbrechungs- Alarm** ***(rote Lampe)*** | | | bei $P_{max}$>50W | | | | | | | |
| **Neutralelektroden-Teilablösungsalarm** ***(optional)*** | | | bei $P_{max}$>50W | | | | | | | |
| **Zustand nach Spannungswiederkehr** ***(ein/aus)*** | | | Default Werte od. Minimum | | | | | | | |
| Ausgangsleistung *(Cut)* | $P_{Cut,max}$: | W | **≤400W**±20% | | | | | | **W** | |
| | $P_{Cut,mittel}$: | W | ± 20% | | | | | | **W** | |
| | 0,1 $P_{Cut,max}$: | W | ± 20% | | | | | | **W** | |
| Ausgangsleistung *(Coag)* | $P_{Coag,max}$: | W | ± 20% | | | | | | **W** | |
| | $P_{Coag,mittel}$: | W | ± 20% | | | | | | **W** | |
| | 0,1 $P_{Coag,max}$: | W | ± 20% | | | | | | **W** | |
| | | | | | | | | | | |
| **Gerätezustand** | | | | | | | | | Unterschrift: | |

**o.k.**...Gerät in Ordnung, **n.z.**....nicht zutreffend, Mängelstufen: **M1**…tolerierbar, **M2**...mittelfristig zu beheben, **M3**...gefährlich

STAATLICH AKKREDITIERTE
**EUROPAPRÜFSTELLE FÜR MEDIZINPRODUKTE**
**am Institut für Health Care Engineering**

| | |
|---|---|
| **Referenz-Nr.** | |
| **Kunde** | |
| **Med. Bereich** | |
| Inventar-Nr. | |

## Wiederkehrende Prüfung
# Infusionspumpe nach EN 62353

| | | | |
|---|---|---|---|
| Gerät | Infusions- (Spritzen) Pumpe | | |
| Typ | | Serien-Nr. | |
| Hersteller | | | |
| Zubehör | ☐ Infusionsbesteck ☐ Infusionsspritze | | |
| | | | |
| **Datum** | | Prüfer | |

| Prüfintervall | 6M | 12M | 24M | 36M |
|---|---|---|---|---|
| Anwend.-Teil | **B** | **BF** | **CF** | **keiner** |
| Schutzklasse | **I** | **II** | **III** | **Batterie** |
| Feuchteschutz | **IPX 1** | **IP** | | |
| Ex.-Schutz | **0** | **AP** | **AP G** | |

| Prüfpunkt | | | n. z. | o. k. | M 1 | M 2 | M 3 | Ergebnis | Bemerkungen |
|---|---|---|---|---|---|---|---|---|---|
| **Sichtprüfung außen** | | | | | | | | | |
| Gebrauchsanweisung *(zusätzliche Prüfpunkte notwendig)* | | | | | | | | | |
| Überprüfungshinweise in Gebrauchsanweisung | | | | | | | | | |
| Typenschild | | | | | | | | | |
| Hersteller | | | | | | | | | |
| CE- Kennzeichnung *(ggf. mit Kenn- Nr.)* | | | | | | | | | |
| Warnhinweise | | | | | | | | | |
| Gerät *(Standfestigkeit, Beschädigung, Betriebsart)* | | | | | | | | | |
| Gehäuse *(Berührschutz, Verschmutzung, Feuchtigkeitsschutz)* | | | | | | | | | |
| Netzstecker *(Adernendhülsen, nacheil.SL, Isolation, Zugentlastung)* | | | | | | | | | |
| Netzleitungsisolation | | | | | | | | | |
| Knickschutz | | r > 1,5D | | | | | | | |
| Zugentlastung | | | | | | | | | |
| Sicherungshalter (*Kennzeichnung, Berührschutz)* | | | | | | | | | |
| Sicherung(en) *(Anzahl, Nennwert)* | | SKI: allpolig<br>SKII: einpolig | | | | | | | |
| Netzschalter *(Kennzeichnung, Schaltrichtung, Leuchte)* | | | | | | | | | |
| **Netzspannungs- Kontrollleuchte** (zusätzlich) | | | | | | | | | |
| **Besteck- Anschluss** *(AT-Typ, GA- Hinweis, Flowrichtig)* | | | | | | | | | |
| **Luftblasendetektor** *(Verschmutzung durch Infusat?)* | | | | | | | | | |
| Einstellteile *(Kennzeichnung, Skalenposition, Betätigungsschutz)* | | | | | | | | | |
| **Infusionsbesteck** *(für Gerät zugelassen?)* | | | | | | | | | |
| **Sicherheitstechnische Parameter** | | | | | | | | | |
| $R_{SL}$ ☐ *(Netzstecker-Gehäuse)* ☐ *(Gerätestecker-Gehäuse)* | | ≤ 0,2Ω | | | | | | **mΩ** | |
| Erdableitstrom *(NC)* | | < 5mA | | | | | | **µA** | |
| Erdableitstrom *(SFC)* | | < 10mA | | | | | | **µA** | |
| Berührungsstrom *(NC)* | | < 100µA | | | | | | **µA** | |
| Berührungsstrom *(SFC)* | | < 500µA | | | | | | **µA** | |
| Patientenableitstrom *(NC)* | B,BF,CF:≤ 10µA DC | B,BF:≤ 100µA AC<br>CF: ≤ 10µA AC | | | | | | **µA** | |
| Patientenableitstrom *(SFC)* | B,BF,CF: ≤ 50µA DC | B,BF:≤ 500µA AC<br>CF: ≤ 50µA AC | | | | | | **µA** | |
| Patientenableitstrom U-AWT *(SFC)* | | B,BF:≤ 5mA<br>CF: ≤ 50µA | | | | | | **µA** | |
| Patientenhilfsstrom *(NC)* | B,BF,CF:≤ 10µA DC | B,BF:≤ 100µA AC<br>CF: ≤ 10µA AC | | | | | | **µA** | |
| Patientenhilfsstrom *(SFC)* | B,BF,CF: ≤ 50µA DC | B,BF:≤ 500µA AC<br>CF: ≤ 50µA AC | | | | | | **µA** | |
| **Funktion** | | | | | | | | | |
| **Netzspannungs- Kontroll- Leuchte** | | | | | | | | | |
| Netzkontroll- Leuchte | | | | | | | | | |
| Ausgangsaktivierungs-Kontroll- Leuchte | | | | | | | | | |
| Akku- Ladekontroll- Leuchte | | | | | | | | | |
| Akku- (Unterspannungs)- Kontroll- Leuchte | | | | | | | | | |
| **keine unbeabsichtigte Verstellung des Kolbens** *(bei ISP)* | | | | | | | | | |
| **Spannungsausfall- Alarm** *(Netzstecker ziehen, schnell aus/ein)* | | | | | | | | | |
| **Tropfensensorpositions- Alarm** *(bei TIP)* | | Δα > 20° | | | | | | | |
| **Okklusions- Alarm** *(Abknicken des Schlauches)* | | bei $Flow_{max}$ | | | | | | | |
| **Maximal**-(Okklusions-)**Druck** *(bei $Flow_{max}$)* | | | | | | | | | |
| **Förderrate** *(5-min-Mittelwert nach 5min)* | | TIP: 20 T/min<br>VIP:25ml/h<br>ISP: 5ml/h | | | | | | | |
| **Bolusgabe** *(5 mal $B_{max}$ bei Testförderrate)* | | TIP: 20 T/min<br>VIP:10 bzw.100ml/h<br>ISP: 5ml/h | | | | | | | |
| **Manipulations- Alarm** *(Besteckwechsel, Ziehen an Leitung)* | | | | | | | | | |
| **Luftblasen- Alarm** *(nicht für ISP)* | | | | | | | | | |
| **Freifluß nur durch 2 unabhängige Aktionen** | | | | | | | | | |
| **Freifluß- Alarm** *(schnelles Infus.-Behälter-Absenken um 50 cm)* | | | | | | | | | |
| **Infusionsende- Voralarm *(bei ISP)*** | | | | | | | | | |
| **Infusionsende- Alarm** | | | | | | | | | |
| **Standby- Alarm** *(nach 1h)* ***(bei AIP)*** | | | | | | | | | |
| **Gerätezustand** | | | | | | | | Unterschrift: | |

**o.k.**...Gerät in Ordnung, **n.z.**....nicht zutreffend, Mängelstufen: **M1**…tolerierbar, **M2**...mittelfristig zu beheben, **M3**...gefährlich

STAATLICH AKKREDITIERTE
**EUROPAPRÜFSTELLE FÜR MEDIZINPRODUKTE**
**am Institut für Health Care Engineering**

| | |
|---|---|
| **Referenz-Nr.** | |
| **Kunde** | |
| **Med. Bereich** | |
| Inventar-Nr. | |

## Wiederkehrende Prüfung
# Infusionspumpe nach EN 62353

| | | | |
|---|---|---|---|
| Gerät | Infusions- (Spritzen) Pumpe | | |
| Typ | | Serien-Nr. | |
| Hersteller | | | |
| Zubehör | ☐ Infusionsbesteck ☐ Infusionsspritze | | |
| | | | |
| **Datum** | | Prüfer | |

| Prüfintervall | 6M | 12M | 24M | 36M |
|---|---|---|---|---|
| Anwend.-Teil | **B** | **BF** | **CF** | **keiner** |
| Schutzklasse | **I** | **II** | **III** | **Batterie** |
| Feuchteschutz | **IPX 1** | **IP** | | |
| Ex.-Schutz | **0** | **AP** | **AP G** | |

| Prüfpunkt | | n. z. | o. k. | M 1 | M 2 | M 3 | Ergebnis | Bemerkungen |
|---|---|---|---|---|---|---|---|---|
| **Sichtprüfung außen** | | | | | | | | |
| Gebrauchsanweisung *(zusätzliche Prüfpunkte notwendig)* | | | | | | | | |
| Überprüfungshinweise in Gebrauchsanweisung | | | | | | | | |
| Typenschild | | | | | | | | |
| Hersteller | | | | | | | | |
| CE- Kennzeichnung *(ggf. mit Kenn- Nr.)* | | | | | | | | |
| Warnhinweise | | | | | | | | |
| Gerät *(Standfestigkeit, Beschädigung, Betriebsart)* | | | | | | | | |
| Gehäuse *(Berührschutz, Verschmutzung, Feuchtigkeitsschutz)* | | | | | | | | |
| Netzstecker *(Adernendhülsen, nacheil.SL, Isolation, Zugentlastung)* | | | | | | | | |
| Netzleitungsisolation | | | | | | | | |
| Knickschutz | r > 1,5D | | | | | | | |
| Zugentlastung | | | | | | | | |
| Sicherungshalter (*Kennzeichnung, Berührschutz)* | | | | | | | | |
| Sicherung(en) *(Anzahl, Nennwert)* | SKI: allpolig SKII: einpolig | | | | | | | |
| Netzschalter *(Kennzeichnung, Schaltrichtung, Leuchte)* | | | | | | | | |
| **Netzspannungs- Kontrollleuchte** (zusätzlich) | | | | | | | | |
| **Besteck- Anschluss** *(AT-Typ, GA- Hinweis, Flowrichtig)* | | | | | | | | |
| **Luftblasendetektor** *(Verschmutzung durch Infusat?)* | | | | | | | | |
| Einstellteile *(Kennzeichnung, Skalenposition, Betätigungsschutz)* | | | | | | | | |
| **Infusionsbesteck** *(für Gerät zugelassen?)* | | | | | | | | |
| | | | | | | | | |
| **Sicherheitstechnische Parameter** | | | | | | | | |
| $R_{SL}$ ☐ *(Netzstecker-Gehäuse)* ☐ *(Gerätestecker-Gehäuse)* | ≤ 0,3Ω | | | | | | **mΩ** | |
| Geräteableitstrom *(Ersatzmessung)* | SKI: max. 1mA | | | | | | **µA** | |
| Geräteableitstrom *(Ersatzmessung)* | SKII:max.0,5mA | | | | | | **µA** | |
| **Geräteableitstrom *(Direkt-/Differenzstrommess.)*** | SKI: max. 0,5mA | | | | | | **µA** | |
| **Geräteableitstrom *(Direkt-/Differenzstrommess.)*** | SKII:max.0,1mA | | | | | | **µA** | |
| Ableitstrom vom Anwendungsteil | BF: max. 5mA CF: max. 50µA | | | | | | **µA** | |
| **Funktion** | | | | | | | | |
| **Netzspannungs- Kontroll- Leuchte** | | | | | | | | |
| Netzkontroll- Leuchte | | | | | | | | |
| Ausgangsaktivierungs-Kontroll- Leuchte | | | | | | | | |
| Akku- Ladekontroll- Leuchte | | | | | | | | |
| Akku- (Unterspannungs)- Kontroll- Leuchte | | | | | | | | |
| **keine unbeabsichtigte Verstellung des Kolbens** *(bei ISP)* | | | | | | | | |
| **Spannungsausfall- Alarm** *(Netzstecker ziehen, schnell aus/ein)* | | | | | | | | |
| **Tropfensensorpositions- Alarm** *(bei TIP)* | Δα > 20° | | | | | | | |
| **Okklusions- Alarm** *(Abknicken des Schlauches)* | bei $Flow_{max}$ | | | | | | | |
| **Maximal**-(Okklusions-)**Druck** *(bei $Flow_{max}$)* | | | | | | | | |
| **Förderrate** *(5-min-Mittelwert nach 5min)* | TIP: 20 T/min VIP:25ml/h ISP: 5ml/h | | | | | | | |
| **Bolusgabe** *(5 mal $B_{max}$ bei Testförderrate)* | TIP: 20 T/min VIP:10 bzw.100ml/h ISP: 5ml/h | | | | | | | |
| **Manipulations- Alarm** *(Besteckwechsel, Ziehen an Leitung)* | | | | | | | | |
| **Luftblasen- Alarm** *(nicht für ISP)* | | | | | | | | |
| **Freifluß nur durch 2 unabhängige Aktionen** | | | | | | | | |
| **Freifluß- Alarm** *(schnelles Infus.-Behälter-Absenken um 50 cm)* | | | | | | | | |
| **Infusionsende- Voralarm *(bei ISP)*** | | | | | | | | |
| **Infusionsende- Alarm** | | | | | | | | |
| **Standby- Alarm** *(nach 1h)* ***(bei AIP)*** | | | | | | | | |
| **Gerätezustand** | | | | | | | Unterschrift: | |

**o.k.**...Gerät in Ordnung, **n.z.**....nicht zutreffend, Mängelstufen: **M1**…tolerierbar, **M2**...mittelfristig zu beheben, **M3**...gefährlich

STAATLICH AKKREDITIERTE
**EUROPAPRÜFSTELLE FÜR MEDIZINPRODUKTE**
am Institut für Health Care Engineering

| | |
|---|---|
| **Referenz-Nr.** | |
| **Kunde** | |
| **Med. Bereich** | |
| Inventar-Nr. | |

## Wiederkehrende Prüfung
# Defibrillator nach EN 62353

| | | | |
|---|---|---|---|
| Gerät | Defibrillator | | |
| Typ | | Serien-Nr. | |
| Hersteller | | | |
| Zubehör | ☐ *externe Defi- Elektr.* ☐*interne Defi- Elektr.* ☐ *EKG- Elektroden* | | |
| | ☐ *Elektrodengel* | | |
| **Datum** | | Prüfer | |

| Prüfintervall | 6M | 12M | 24M | 36M |
|---|---|---|---|---|
| Anwend.-Teil | B | BF | CF | keiner |
| Schutzklasse | I | II | III | Batterie |
| Feuchteschutz | IPX 1 | IP | | |
| Ex.-Schutz | 0 | AP | APG | |

| Prüfpunkt | | | | n.z. | o.k. | M1 | M2 | M3 | Ergebnis | Bemerkungen |
|---|---|---|---|---|---|---|---|---|---|---|
| **Sichtprüfung außen** | | | | | | | | | | |
| Gebrauchsanweisung *(zusätzliche Prüfpunkte notwendig)* | | | | | | | | | | |
| Typenschild | | | | | | | | | | |
| Hersteller | | | | | | | | | | |
| CE- Kennzeichnung *(ggf. mit Kenn- Nr.)* | | | | | | | | | | |
| **Kurzbedienungsanweisung** *(am Gerät)* | | | | | | | | | | |
| Warnhinweise | | | | | | | | | | |
| Gerät *(Standfestigkeit, Beschädigung, Betriebsart)* | | | | | | | | | | |
| Gehäuse *(Berührschutz, Verschmutzung, Feuchtigkeitsschutz)* | | | | | | | | | | |
| Netzstecker *(Adernendhülsen, nacheil.SL, Isolation, Zugentlastung)* | | | | | | | | | | |
| Netzleitungsisolation | | | | | | | | | | |
| Knickschutz | | | r > 1,5D | | | | | | | |
| Zugentlastung | | | | | | | | | | |
| Sicherungshalter (*Kennzeichnung, Berührschutz)* | | | | | | | | | | |
| Sicherung(en) *(Anzahl, Nennwert)* | | | SKI: allpolig<br>SKII: einpolig | | | | | | | |
| Netzschalter *(Kennzeichnung, Schaltrichtung, Leuchte)* | | | | | | | | | | |
| Patientenanschlüsse *(unverwechselbar)* | | | | | | | | | | |
| Kennzeichnung Pat.-Anschl. *(AT-Type, GA-Hinweis)* | | | | | | | | | | |
| Einstellteile *(Kennzeichnung, Skalenposition, Betätigungsschutz)* | | | | | | | | | | |
| **Akku *(Zustand, Alter, Ablaufdatum)*** | | | | | | | | | | |
| **Defi-Elektroden *(extern)*** | | | KS≥50mm<br>LS≥25mm | | | | | | | |
| **Defi-Elektroden *(intern)*** | | | | | | | | | | |
| **EKG-Elektroden** *(Typ CF-AWT, Ablaufdatum)* | | | | | | | | | | |
| **Elektrodengel** *(nicht eingetrocknet, Ablaufdatum)* | | | | | | | | | | |
| **Sicherheitstechnische Parameter** | | | | | | | | | | |
| $R_{SL}$ ☐ *(Netzstecker-Gehäuse)* ☐ *(Gerätestecker-Gehäuse)* | | | ≤ 0,2Ω | | | | | | **mΩ** | |
| Geräteableitstrom *(Ersatzmessung)* | | | SKI: max. 1mA | | | | | | **µA** | |
| Erdableitstrom | *(NC)* | | < 5mA | | | | | | **µA** | |
| Erdableitstrom | *(SFC)* | | < 10mA | | | | | | **µA** | |
| Berührungsstrom | *(NC)* | | < 100µA | | | | | | **µA** | |
| Berührungsstrom | *(SFC)* | | < 500µA | | | | | | **µA** | |
| Patientenableitstrom | *(NC)* | B,BF,CF:≤ 10µA DC | B,BF:≤ 100µA AC<br>CF: ≤ 10µA AC | | | | | | **µA** | |
| Patientenableitstrom | *(SFC)* | B,BF,CF: ≤ 50µA DC | B,BF:≤ 500µA AC<br>CF: ≤ 50µA AC | | | | | | **µA** | |
| Patientenableitstrom U-AWT *(SFC)* | | | B,BF:≤ 5mA<br>CF: ≤ 50µA | | | | | | **µA** | |
| Patientenhilfsstrom | *(NC)* | B,BF,CF:≤ 10µA DC | B,BF:≤ 100µA AC<br>CF: ≤ 10µA AC | | | | | | **µA** | |
| Patientenhilfsstrom | *(SFC)* | B,BF,CF: ≤ 50µA DC | B,BF:≤ 500µA AC<br>CF: ≤ 50µA AC | | | | | | **µA** | |
| **Funktion** | | | | | | | | | | |
| Netzkontroll- Leuchte | | | | | | | | | | |
| Akku- Lade-/Unterspannungs- Kontroll- Leuchte | | | | | | | | | | |
| **Zusatzaktion bei Energien > 360 J** | | | | | | | | | | |
| **Ladung/Entladung nur nach Aktivierung** | | | | | | | | | | |
| **keine Impulsabgabe während Aufladens** | | | | | | | | | | |
| **Abgabeenergie** | @ 50Ω bei $W_{max}$ | | ±3J bzw.±15% | | | | | | **J** | |
| | @ 50Ω 30s nach Aufladung bei $W_{max}$ | | >85% | | | | | | **J** | |
| | @ 50Ω bei $W_{mittel}$ | | ±3J bzw.±15% | | | | | | **J** | |
| | @ 50Ω bei $W_{min}$ | | ±3J bzw.±15% | | | | | | **J** | |
| **Ladezeit nach 5 Entl.** | bei $W_{max}$ | | < 15s | | | | | | **sec** | |
| **keine Impulsabgabe nach Ausschalten** | | | | | | | | | | |
| **Synchronbetrieb- Kontrollleuchte** | | | | | | | | | | |
| **Auslösung durch EKG- R- Zacke** | | | | | | | | | | |
| **keine Entladung im Synchronbetrieb** | | | | | | | | | | |
| **EKG- Signal nicht gleichzeitig von Defi- und EKG- Elektroden** | | | | | | | | | | |
| **Monitorausfallzeit** nach Entladung @$W_{max}$ | | | < 10s | | | | | | **sec** | |
| **Gerätezustand** | | | | | | | | | Unterschrift: | |

**o.k.**...Gerät in Ordnung, **n.z.**....nicht zutreffend, Mängelstufen: **M1**…tolerierbar, **M2**...mittelfristig zu beheben, **M3**...gefährlich

STAATLICH AKKREDITIERTE
**EUROPAPRÜFSTELLE FÜR MEDIZINPRODUKTE**
am Institut für Health Care Engineering

| | |
|---|---|
| **Referenz-Nr.** | |
| **Kunde** | |
| **Med. Bereich** | |
| Inventar-Nr. | |

## Wiederkehrende Prüfung
# Defibrillator nach EN 62353

| | | | |
|---|---|---|---|
| Gerät | Defibrillator | | |
| Typ | | Serien-Nr. | |
| Hersteller | | | |
| Zubehör | ☐ *externe Defi- Elektr.* ☐ *interne Defi- Elektr.* ☐ *EKG- Elektroden* | | |
| | ☐ *Elektrodengel* | | |
| **Datum** | | Prüfer | |

| Prüfintervall | 6M | 12M | 24M | 36M |
|---|---|---|---|---|
| Anwend.-Teil | B | BF | CF | keiner |
| Schutzklasse | I | II | III | Batterie |
| Feuchteschutz | IPX 1 | IP | | |
| Ex.-Schutz | 0 | AP | AP G | |

| Prüfpunkt | | n. z. | o. k. | M 1 | M 2 | M 3 | Ergebnis | Bemerkungen |
|---|---|---|---|---|---|---|---|---|
| **Sichtprüfung außen** | | | | | | | | |
| Gebrauchsanweisung *(zusätzliche Prüfpunkte notwendig)* | | | | | | | | |
| Typenschild | | | | | | | | |
| Hersteller | | | | | | | | |
| CE- Kennzeichnung *(ggf. mit Kenn- Nr.)* | | | | | | | | |
| **Kurzbedienungsanweisung** *(am Gerät)* | | | | | | | | |
| Warnhinweise | | | | | | | | |
| Gerät *(Standfestigkeit, Beschädigung, Betriebsart)* | | | | | | | | |
| Gehäuse *(Berührschutz, Verschmutzung, Feuchtigkeitsschutz)* | | | | | | | | |
| Netzstecker *(Adernendhülsen, nacheil.SL, Isolation, Zugentlastung)* | | | | | | | | |
| Netzleitungsisolation | | | | | | | | |
| Knickschutz | r > 1,5D | | | | | | | |
| Zugentlastung | | | | | | | | |
| Sicherungshalter (*Kennzeichnung, Berührschutz)* | | | | | | | | |
| Sicherung(en) *(Anzahl, Nennwert)* | SKI: allpolig SKII: einpolig | | | | | | | |
| Netzschalter *(Kennzeichnung, Schaltrichtung, Leuchte)* | | | | | | | | |
| Patientenanschlüsse *(unverwechselbar)* | | | | | | | | |
| Kennzeichnung Pat.-Anschl. *(AT-Type, GA-Hinweis)* | | | | | | | | |
| Einstellteile *(Kennzeichnung, Skalenposition, Betätigungsschutz)* | | | | | | | | |
| **Akku *(Zustand, Alter, Ablaufdatum)*** | | | | | | | | |
| **Defi-Elektroden *(extern)*** | KS≥50mm LS≥25mm | | | | | | | |
| **Defi-Elektroden *(intern)*** | | | | | | | | |
| **EKG-Elektroden** *(Typ CF-AWT, Ablaufdatum)* | | | | | | | | |
| **Elektrodengel** *(nicht eingetrocknet, Ablaufdatum)* | | | | | | | | |
| | | | | | | | | |
| **Sicherheitstechnische Parameter** | | | | | | | | |
| $R_{SL}$ ☐ *(Netzstecker-Gehäuse)* ☐ *(Gerätestecker-Gehäuse)* | ≤ 0,3Ω | | | | | | **mΩ** | |
| Geräteableitstrom *(Ersatzmessung)* | SKI: max. 1mA | | | | | | **µA** | |
| Geräteableitstrom *(Ersatzmessung)* | SKII: max. 0,5mA | | | | | | **µA** | |
| **Geräteableitstrom *(Direkt-/Differenzstrommess.)*** | SKI: max. 0,5mA | | | | | | **µA** | |
| **Geräteableitstrom *(Direkt-/Differenzstrommess.)*** | SKII: max. 0,1mA | | | | | | **µA** | |
| Ableitstrom vom Anwendungsteil | BF: max. 5mA CF: max. 50µA | | | | | | **µA** | |
| **Funktion** | | | | | | | | |
| Netzkontroll- Leuchte | | | | | | | | |
| Akku- Lade-/Unterspannungs- Kontroll- Leuchte | | | | | | | | |
| **Zusatzaktion bei Energien > 360 J** | | | | | | | | |
| **Ladung/Entladung nur nach Aktivierung** | | | | | | | | |
| **keine Impulsabgabe während Aufladens** | | | | | | | | |
| **Abgabeenergie** | @ 50Ω bei $W_{max}$ | ±3J bzw.±15% | | | | | **J** | |
| | @ 50Ω 30s nach Aufladung bei $W_{max}$ | >85% | | | | | **J** | |
| | @ 50Ω bei $W_{mittel}$ | ±3J bzw.±15% | | | | | **J** | |
| | @ 50Ω bei $W_{min}$ | ±3J bzw.±15% | | | | | **J** | |
| **Ladezeit nach 5 Entl.** | bei $W_{max}$ | < 15s | | | | | **sec** | |
| **keine Impulsabgabe nach Ausschalten** | | | | | | | | |
| **Synchronbetrieb- Kontrollleuchte** | | | | | | | | |
| **Auslösung durch EKG- R- Zacke** | | | | | | | | |
| **keine Entladung im Synchronbetrieb** | | | | | | | | |
| **EKG- Signal nicht gleichzeitig von Defi- und EKG- Elektroden** | | | | | | | | |
| **Monitorausfallzeit** nach Entladung @$W_{max}$ | < 10s | | | | | | **sec** | |
| **Gerätezustand** | | | | | | | Unterschrift: | |

**o.k.**...Gerät in Ordnung, **n.z.**....nicht zutreffend, Mängelstufen: **M1**…tolerierbar, **M2**...mittelfristig zu beheben, **M3**...gefährlich

STAATLICH AKKREDITIERTE
**EUROPAPRÜFSTELLE FÜR MEDIZINPRODUKTE**
am Institut für Health Care Engineering

| | |
|---|---|
| **Referenz-Nr.** | |
| **Kunde** | |
| **Med. Bereich** | |
| Inventar-Nr. | |

## Wiederkehrende Prüfung
# Pflege-/Krankenbetten nach EN 62353

| | | | |
|---|---|---|---|
| Gerät | Pflege-/Krankenbett | | |
| Typ | | Serien-Nr. | |
| Hersteller | | | |
| Zubehör | | | |
| | | | |
| **Datum** | | Prüfer | |

| Prüfintervall | 6M | 12M | 24M | 36M |
|---|---|---|---|---|
| Anwend.-Teil | B | BF | CF | keiner |
| Schutzklasse | I | II | III | Batterie |
| Feuchteschutz | IPX 4 | IP | | |
| Ex.-Schutz | 0 | AP | AP G | |

| Prüfpunkt | | | n. z. | o. k. | M 1 | M 2 | M 3 | Ergebnis | Bemerkungen |
|---|---|---|---|---|---|---|---|---|---|
| **Sichtprüfung außen** | | | | | | | | | |
| Gebrauchsanweisung *(zusätzliche Prüfpunkte notwendig)* | | | | | | | | | |
| Typenschild | | | | | | | | | |
| Hersteller | | | | | | | | | |
| CE- Kennzeichnung *(ggf. mit Kenn- Nr.)* | | | | | | | | | |
| Warnhinweise | | | | | | | | | |
| Gerät *(Standfestigkeit, Beschädigung, Betriebsart)* | | | | | | | | | |
| Gehäuse *(Berührschutz, Verschmutzung, Feuchtigkeitsschutz)* | | | | | | | | | |
| Netzstecker *(Adernendhülsen, nacheil.SL, Isolation, Zugentlastung)* | | | | | | | | | |
| Netzleitungsisolation | | | | | | | | | |
| Knickschutz | | r > 1,5D | | | | | | | |
| Zugentlastung | | | | | | | | | |
| Transporthalterung für Netzkabel | | | | | | | | | |
| Sicherungshalter (*Kennzeichnung, Berührschutz)* | | | | | | | | | |
| Sicherung(en) *(Anzahl, Nennwert)* | | SKI: allpolig<br>SKII: einpolig | | | | | | | |
| Netzschalter *(Kennzeichnung, Schaltrichtung, Leuchte)* | | | | | | | | | |
| Klemm- Quetschstellen | | | | | | | | | |
| Fußschalter gegen unbeabs. Betätigen geschützt *(Schutzbügel, deakt.)* | | | | | | | | | |
| Kontrolle von Patientensicherungssystemen, Gurtsystemen | | | | | | | | | |
| Gurte: Verwendung nur mit durchgehenden Seitengitter | | | | | | | | | |
| Kontrolle der Arretierungen | | | | | | | | | |
| Seitengitter: mögl. Gefährdung, Defekt | | | | | | | | | |
| Feuchtigkeitsschutz | | | | | | | | | |
| Einstellteile *(Kennzeichnung, Skalenposition, Betätigungsschutz)* | | | | | | | | | |
| Vorrichtung zum Deaktivieren der Bedieneinheit | | | | | | | | | |
| Kontrolle der Gelenke *(Gefährdung durch herabstürzen)* | | | | | | | | | |
| Zubehör *(vollzählig, mech. i.O.)* | | | | | | | | | |
| **Sicherheitstechnische Parameter** | | | | | | | | | |
| $R_{SL}$ ☐ *(Netzstecker-Gehäuse)* ☐ *(Gerätestecker-Gehäuse)* | | ≤ 0,2Ω | | | | | | **mΩ** | |
| Erdableitstrom | *(NC)* | ≤ 5mA | | | | | | **µA** | |
| Erdableitstrom | *(SFC)* | ≤ 10mA | | | | | | **µA** | |
| Berührungsstrom | *(NC)* | ≤ 100µA | | | | | | **µA** | |
| Berührungsstrom | *(SFC)* | ≤ 500µA | | | | | | **µA** | |
| Patientenableitstrom | *(NC)* | B,BF,CF:≤ 10µA DC | B,BF:≤ 100µA AC<br>CF: ≤ 10µA AC | | | | | **µA** | |
| Patientenableitstrom | *(SFC)* | B,BF,CF: ≤ 50µA DC | B,BF:≤ 500µA AC<br>CF: ≤ 50µA AC | | | | | **µA** | |
| Patientenableitstrom U-AWT *(SFC)* | | | B,BF:≤ 5mA<br>CF: ≤ 50µA | | | | | **µA** | |
| Patientenhilfsstrom | *(NC)* | B,BF,CF:≤ 10µA DC | B,BF:≤ 100µA AC<br>CF: ≤ 10µA AC | | | | | **µA** | |
| Patientenhilfsstrom | *(SFC)* | B,BF,CF: ≤ 50µA DC | B,BF:≤ 500µA AC<br>CF: ≤ 50µA AC | | | | | **µA** | |
| **Funktion** | | | | | | | | | |
| Netzkontroll- Leuchte | | | | | | | | | |
| Akku- Ladekontroll- Leuchte | | | | | | | | | |
| Akku- (Unterspannungs)- Kontroll- Leuchte | | | | | | | | | |
| Zustand n. Spannungswiederkehr/Funktionsprüfung | | | | | | | | | |
| Alle auswählbaren Bewegungsmöglichkeiten überprüfen | | | | | | | | | |
| Bedienung über Taster *(hold-to-run)* | | | | | | | | | |
| **Gerätezustand** | | | | | | | | Unterschrift: | |

**o.k.**...Gerät in Ordnung, **n.z.**...nicht zutreffend, Mängelstufen: **M1**…tolerierbar, **M2**...mittelfristig zu beheben, **M3**...gefährlich

STAATLICH AKKREDITIERTE
**EUROPAPRÜFSTELLE FÜR MEDIZINPRODUKTE**
am Institut für Health Care Engineering

| | |
|---|---|
| **Referenz-Nr.** | |
| **Kunde** | |
| **Med. Bereich** | |
| Inventar-Nr. | |

## Wiederkehrende Prüfung
# Pflege-/Krankenbetten nach EN 62353

| | | | |
|---|---|---|---|
| Gerät | Pflege-/Krankenbett | | |
| Typ | | Serien-Nr. | |
| Hersteller | | | |
| Zubehör | | | |
| | | | |
| **Datum** | | Prüfer | |

| Prüfintervall | 6M | 12M | 24M | 36M |
|---|---|---|---|---|
| Anwend.-Teil | **B** | **BF** | **CF** | **keiner** |
| Schutzklasse | **I** | **II** | **III** | **Batterie** |
| Feuchteschutz | **IPX 4** | **IP** | | |
| Ex.-Schutz | **0** | **AP** | **AP G** | |

| Prüfpunkt | | n. z. | o. k. | M 1 | M 2 | M 3 | Ergebnis | Bemerkungen |
|---|---|---|---|---|---|---|---|---|
| **Sichtprüfung außen** | | | | | | | | |
| Gebrauchsanweisung *(zusätzliche Prüfpunkte notwendig)* | | | | | | | | |
| Typenschild | | | | | | | | |
| Hersteller | | | | | | | | |
| CE- Kennzeichnung *(ggf. mit Kenn- Nr.)* | | | | | | | | |
| Warnhinweise | | | | | | | | |
| Gerät *(Standfestigkeit, Beschädigung, Betriebsart)* | | | | | | | | |
| Gehäuse *(Berührschutz, Verschmutzung, Feuchtigkeitsschutz)* | | | | | | | | |
| Netzstecker *(Adernendhülsen, nacheil.SL, Isolation, Zugentlastung)* | | | | | | | | |
| Netzleitungsisolation | | | | | | | | |
| Knickschutz | r > 1,5D | | | | | | | |
| Zugentlastung | | | | | | | | |
| Transporthalterung für Netzkabel | | | | | | | | |
| Sicherungshalter (*Kennzeichnung, Berührschutz)* | | | | | | | | |
| Sicherung(en) *(Anzahl, Nennwert)* | SKI: allpolig<br>SKII: einpolig | | | | | | | |
| Netzschalter *(Kennzeichnung, Schaltrichtung, Leuchte)* | | | | | | | | |
| Klemm- Quetschstellen | | | | | | | | |
| Fußschalter gegen unbeabs. Betätigen geschützt *(Schutzbügel, deakt.)* | | | | | | | | |
| Kontrolle von Patientensicherungssystemen, Gurtsystemen | | | | | | | | |
| Gurte: Verwendung nur mit durchgehenden Seitengitter | | | | | | | | |
| Kontrolle der Arretierungen | | | | | | | | |
| Seitengitter: mögl. Gefährdung, Defekt | | | | | | | | |
| Feuchtigkeitsschutz | | | | | | | | |
| Einstellteile *(Kennzeichnung, Skalenposition, Betätigungsschutz)* | | | | | | | | |
| Vorrichtung zum Deaktivieren der Bedieneinheit | | | | | | | | |
| Kontrolle der Gelenke *(Gefährdung durch herabstürzen)* | | | | | | | | |
| Zubehör *(vollzählig, mech. i.O.)* | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| **Sicherheitstechnische Parameter** | | | | | | | | |
| $R_{SL}$ ☐ *(Netzstecker-Gehäuse)* ☐ *(Gerätestecker-Gehäuse)* | ≤ 0,3Ω | | | | | | **mΩ** | |
| Geräteableitstrom *(Ersatzmessung)* | SKI: max. 1mA | | | | | | **µA** | |
| Geräteableitstrom *(Ersatzmessung)* | SKII:max.0,5mA | | | | | | **µA** | |
| **Geräteableitstrom *(Direkt-/Differenzstrommess.)*** | SKI: max. 0,5mA | | | | | | **µA** | |
| **Geräteableitstrom *(Direkt-/Differenzstrommess.)*** | SKII:max.0,1mA | | | | | | **µA** | |
| Ableitstrom vom Anwendungsteil | BF: max. 5mA<br>CF: max. 50µA | | | | | | **µA** | |
| **Funktion** | | | | | | | | |
| Netzkontroll- Leuchte | | | | | | | | |
| Akku- Ladekontroll- Leuchte | | | | | | | | |
| Akku- (Unterspannungs)- Kontroll- Leuchte | | | | | | | | |
| Zustand n. Spannungswiederkehr/Funktionsprüfung | | | | | | | | |
| Alle auswählbaren Bewegungsmöglichkeiten überprüfen | | | | | | | | |
| Bedienung über Taster *(hold-to-run)* | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| **Gerätezustand** | | | | | | | Unterschrift: | |

**o.k.**...Gerät in Ordnung, **n.z.**...nicht zutreffend, Mängelstufen: **M1**…tolerierbar, **M2**...mittelfristig zu beheben, **M3**...gefährlich

# Appendix C.

# User manual

Hunor Kovács

# Benutzerhandbuch für die Software zur Dokumentation der wiederkehrenden Prüfung von Medizinprodukten nach der Medizinproduktebetreiberverordnung

Health Care Engineering Projekt


**Institut für Health Care Engineering**
Technische Universität Graz
Stremayrgasse 16/II, A - 8010 Graz

Leiter: Univ.-Prof. Dipl.-Ing. Dr.techn. Baumgartner Christian

Betreuer: Univ.-Prof. Dipl.-Ing. Dr.techn. Baumgartner Christian

Begutachter: Univ.-Prof. Dipl.-Ing. Dr.techn. Baumgartner Christian

Graz, Oktober 2019

# Inhaltsverzeichnis

# Abbildunsverzeichnis

# 1 Allgemeine Buttons

Bis auf die einzelnen Menüs besitzen sämtliche Screens eine Titel- und eine Navigationsleiste, welche jeweils über eigene Buttons verfügen. Diese werden in diesem Kapitel näher erläutert.

## 1.1 Titelleiste

### 1.1.1 Hinzufügen-Button

*Abbildung 1: Hinzufügen-Button*

Wird beim Anlegen neuer Prüfer, Kunden, Geräte, Prüfungen oder Geräteprüfungen verwendet.

### 1.1.2 Entfernen- bzw. Reaktivieren-Button

*Abbildung 2: Entfernen- bzw. Reaktivieren-Button*

Dient dem Entfernen von ausgewählten Prüfern, Kunden, Geräten, Prüfungen oder Geräteprüfungen. Weiters können damit bereits abgeschlossene Geräteprüfungen in noch nicht abgeschlossene Prüfungen umgewandelt und wieder editierbar gemacht werden (siehe Kapitel 4.2.1 und 4.2.2).

### 1.1.3 Suchen-Button

*Abbildung 3: Suchen-Button*

Schaltet die Suchmaske frei. Diese kann zum Durchsuchen aller Elemente auf dem aktuellen Screen verwendet werden.

## 1.2 Navigationsbuttons

### 1.2.1 Zurück-Button

*Abbildung 4: Zurück-Button*

Navigiert zum vorangehenden Screen oder dient zum Verlassen der Suchmaske.

### 1.2.2 Hauptmenü-Button

*Abbildung 5: Hauptmenü-Button*

Bewirkt die sofortige Rückkehr zum Hauptmenü.

### 1.2.3 Bestätigen-Button

*Abbildung 6: Bestätigen-Button*

Dient der Bestätigung eines Elements oder mehrerer Elemente oder dem Voranschreiten auf den nachfolgenden Screen.

# 2 Hauptmenü

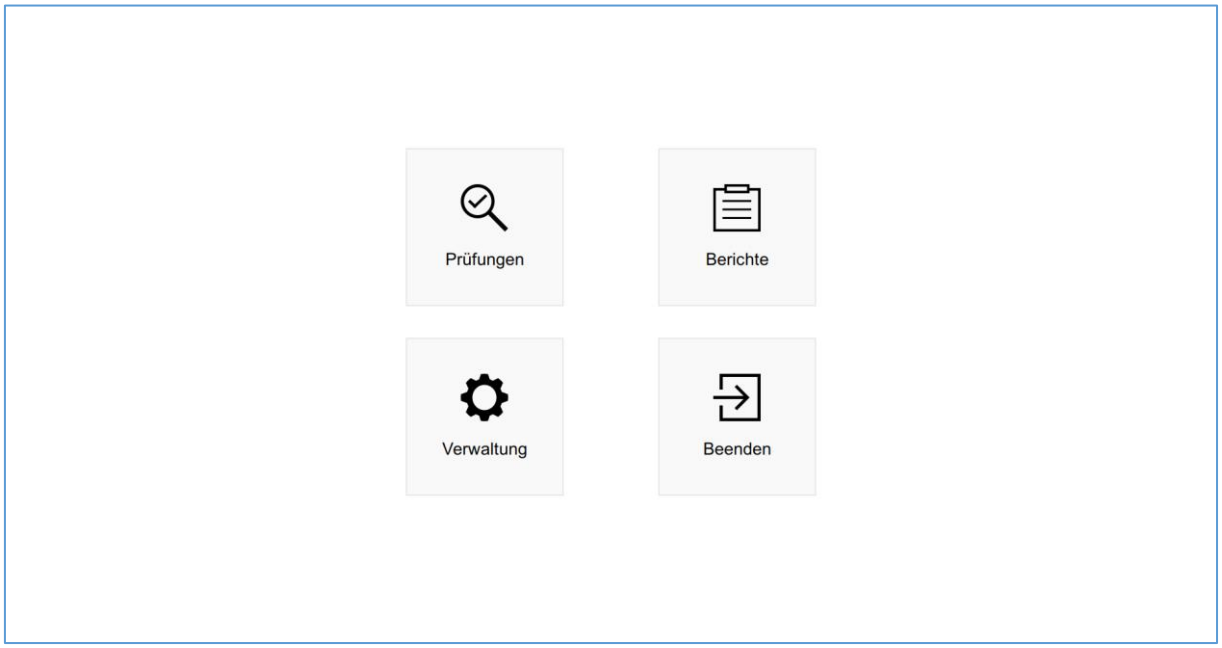


*Abbildung 7: Hauptmenü*

Das Hauptmenü hat, wie in Abbildung 7 ersichtlich, vier Buttons über die Sie einzelne Abschnitte bzw. Funktionalitäten der Software erreichen können:

- Prüfungen
- Berichte
- Verwaltung
- Beenden

Die nachfolgenden Kapitel beschäftigen sich näher mit den einzelnen Abschnitten und deren Funktionsumfang.

# 3 Verwaltung

In diesem Bereich können Sie die Verwaltung von Prüfern und Kunden sowie die Aktualisierung der Geräteprüfungsvorlagen vornehmen.


*Abbildung 8: Verwaltungsmenü*

Wie in Abbildung 8 dargestellt, haben Sie hier drei verschiedene Möglichkeiten:

- Prüfer verwalten
- Kunden verwalten
- Geräteprüfungsvorlagen aktualisieren

## 3.1 Prüfer verwalten

### 3.1.1 Prüfer hinzufügen

Wenn Sie den Hinzufügen-Button betätigen, können Sie einen neuen Prüfer anlegen. Es erscheint eine Eingabeaufforderung, welche der Eingabe des Namens dient.

### 3.1.2 Prüfer entfernen

Wird ein bereits angelegter Prüfer ausgewählt, kann dieser durch die Betätigung des Entfernen-Buttons gelöscht werden.

## 3.2 Kunden verwalten

### 3.2.1 Kunden hinzufügen

Anhand des Hinzufügen-Buttons können Sie einen neuen Kunden anlegen. Es erscheint folgende Eingabeaufforderung:

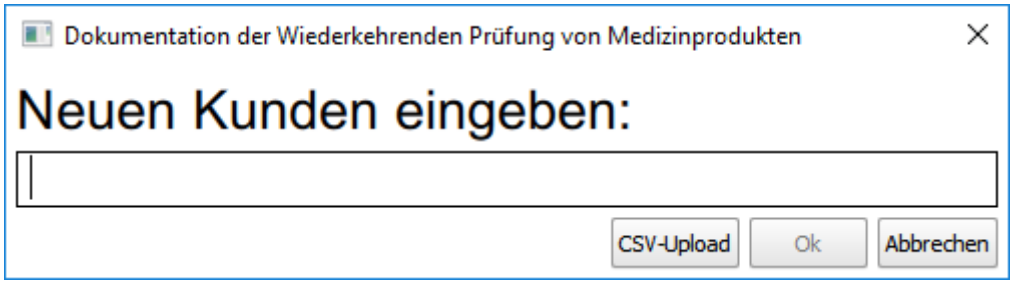


*Abbildung 9: Eingabeaufforderung für einen neuen Kunden*

Wie in Abbildung 9 ersichtlich, gibt es hier eine Texteingabe für den Namen. Neben den Buttons „Ok“ und „Abbrechen“ ist dort auch der „CSV-Upload-Button zu finden, welcher zum Hochladen der Geräteliste des Kunden dient.
Bitte beachten Sie, dass der Speicherort der Geräteliste beim Hinzufügen softwareintern gesichert wird. Außerdem greift die Software während der Laufzeit auf die Geräteliste zu.

### 3.2.2 Kunden entfernen

Durch die Auswahl eines bereits angelegten Kunden können Sie diesen durch die Betätigung des Entfernen-Buttons löschen.

### 3.2.3 Geräteliste eines Kunden anzeigen

Wenn Sie einen Kunden selektiert haben, können Sie durch das Betätigen des Bestätigen-Buttons dessen Gerätemenü öffnen.

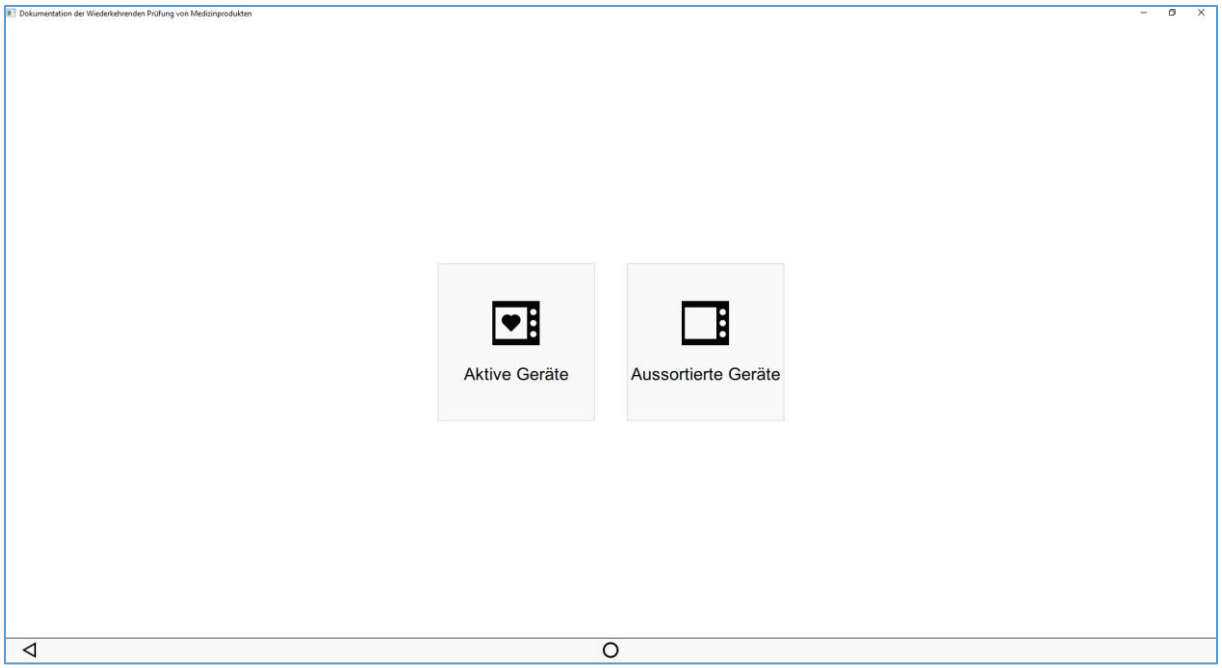


*Abbildung 10: Gerätemenü*

Wie in Abbildung 10 ersichtlich, können durch das Gerätemenü entweder die aktiven oder die aussortierten Geräte des Kunden angezeigt werden. Die Verwaltung der Geräte von Kunden ist dem Kapitel 3.2.4 zu entnehmen.

### 3.2.4 Geräte eines Kunden verwalten

#### *3.2.4.1 Gerät hinzufügen*

Im Bereich „Aktive Geräte“ (siehe Kapitel 3.2.3) können Sie durch die Betätigung des Hinzufügen-Buttons ein neues Gerät anlegen. Es erscheint folgende Eingabeaufforderung:

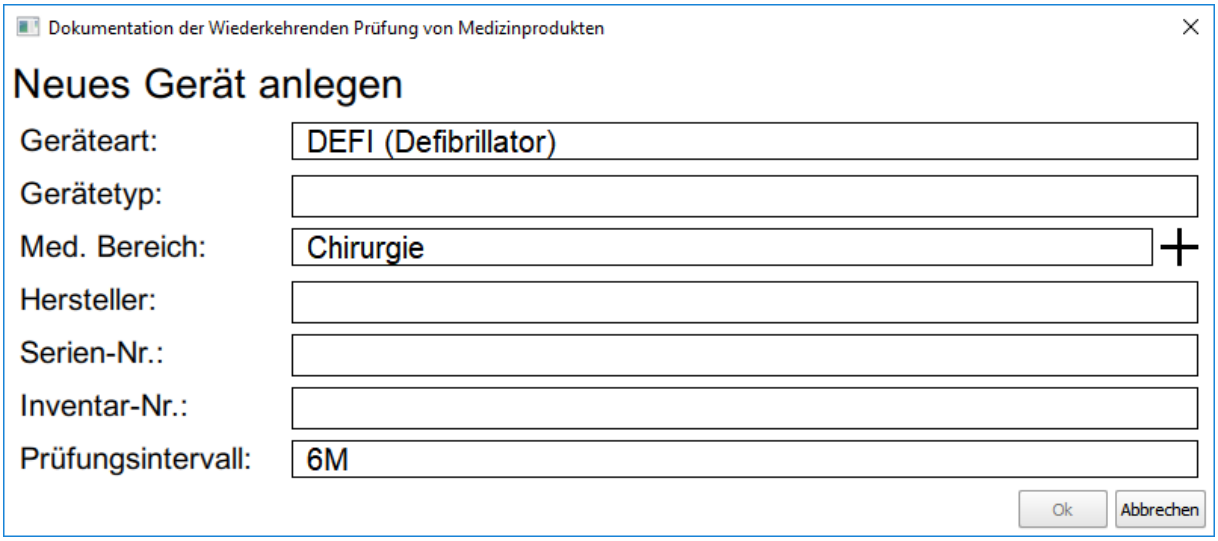


*Abbildung 11: Neues Gerät anlegen*

Abbildung 11 zeigt die Maske zum Hinzufügen neuer Geräte. Sie müssen dabei sämtliche Felder ausfüllen, um das neue Gerät erstellen zu können. Beim Erstellen von Prüfungen werden diese Felder von den abgeleiteten Geräteprüfungen übernommen. In allen Textfeldern muss zumindest ein Zeichen eingegeben werden, leere Felder sind nicht erlaubt.

In den Zeilen „Geräteart“ und „Prüfungsintervall“ kann aus vordefinierten Werten gewählten werden.

Im Feld „Med. Bereich“ können Sie aus den bereits existierenden medizinischen Bereichen selektieren - oder durch Drücken auf den Hinzufügen-Button einen neuen med. Bereich hinzufügen.

#### *3.2.4.2 Gerät aussortieren oder reaktivieren*

Dies erfolgt durch die Auswahl des Gerätes und anschließend durch das Drücken des Entfernen-Buttons. Der Effekt ist je nach Bereich (siehe Kapitel 3.2.3) unterschiedlich. In „Aktive Geräte“ wird das ausgewählte Gerät aussortiert, im Bereich „Aussortierte Geräte“ wird es reaktiviert.

Bitte beachten Sie, dass nur aktive Geräte beim Anlegen von Prüfungen ausgewählt werden können.

## 3.3 Geräteprüfungsvorlagen aktualisieren

Hierbei werden die hinterlegten Vorlagen (Microsoft Word .DOCX-Format) für die Geräteprüfungen in softwareinterne Strukturen umgewandelt und gespeichert. Dies sollten Sie bei jeder Änderung der .DOCX-Vorlagen ausführen, da auf diese Vorlagen beim Export von Geräteprüfungen zugegriffen wird.

Bitte beachten Sie, dass der Aktualisierungsvorgang nicht mehr abgebrochen werden kann, sobald er gestartet ist. Außerdem ist die Software während dieser Zeit nicht anderweitig nutzbar.

# 4 Prüfungen

In diesem Kapitel wird der Abschnitt „Prüfungen“ genauer beschrieben.
Beim Betreten des Bereichs werden alle bereits erfassten, aber noch nicht abgeschlossenen Prüfungen angezeigt.

## 4.1 Prüfungen verwalten

### 4.1.1 Prüfung hinzufügen

Durch das Betätigen des Hinzufügen-Buttons kann eine neue Prüfung angelegt werden. Wählen Sie zunächst jenen Kunden, welchem die Prüfung zugeordnet ist und lassen Sie sich anschließend eine Liste aller aktiven Geräte (siehe Kapitel 3.2.4) des Kunden anzeigen. Hier können Sie nun sämtliche Geräte auswählen, welche geprüft werden sollen. Schließlich müssen Sie noch die Bezeichnung für die Prüfung angeben.

### 4.1.2 Prüfung entfernen

Wird eine bereits angelegte Prüfung ausgewählt, können Sie diese durch das Anklicken des Entfernen-Buttons löschen.

## 4.2 Geräteprüfungen verwalten

Wird eine Prüfung ausgewählt und anschließend der Bestätigen-Button betätigt, so gelangen Sie zum Geräteprüfungsmenü.

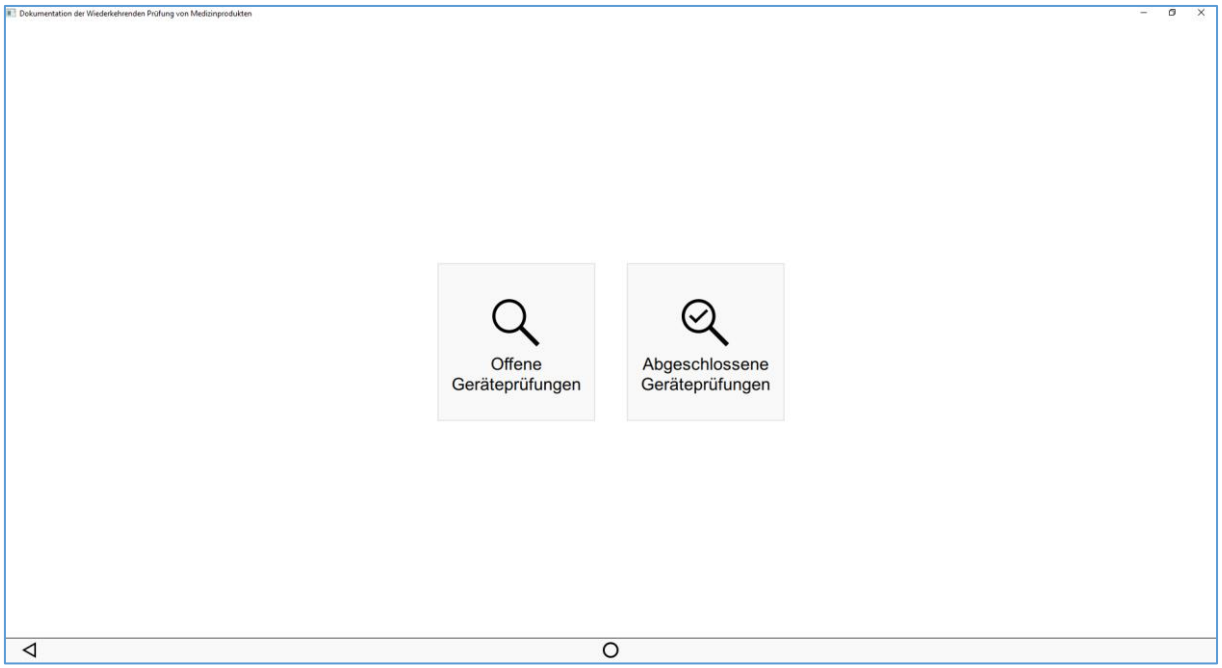


*Abbildung 12: Geräteprüfungsmenü*

Wie in Abbildung 12 dargestellt, kann hier zwischen der Anzeige von offenen oder bereits abgeschlossenen Geräteprüfungen gewählt werden.

### 4.2.1 Offene Geräteprüfungen

In dieser Sektion werden sämtliche offenen Geräteprüfungen angezeigt. Geräteprüfungen gelten als abgeschlossen, wenn Sie sämtliche Felder ordnungsgemäß ausgefüllt haben.

#### *4.2.1.1 Geräteprüfungen hinzufügen*

Durch Auswählen des Hinzufügen-Buttons kann eine neue Geräteprüfung angelegt werden. Hierbei können Sie - wie beim Erstellen von neuen Prüfungen (siehe Kapitel 4.1.1) - die zu prüfenden Geräte aus der Geräteliste des entsprechenden Kunden wählen.

#### *4.2.1.2 Geräteprüfung entfernen*

Bei der Auswahl einer bereits angelegten Prüfung kann diese durch Betätigen des Entfernen-Buttons gelöscht werden.

### *4.2.1.3 Geräteprüfung bearbeiten*

Bei der Bearbeitung von Geräteprüfungen gibt es wiederum vier Teilbereiche, auf die in den folgenden Unterkapiteln näher eingegangen wird.

#### 4.2.1.3.1 Allgemeine Informationen

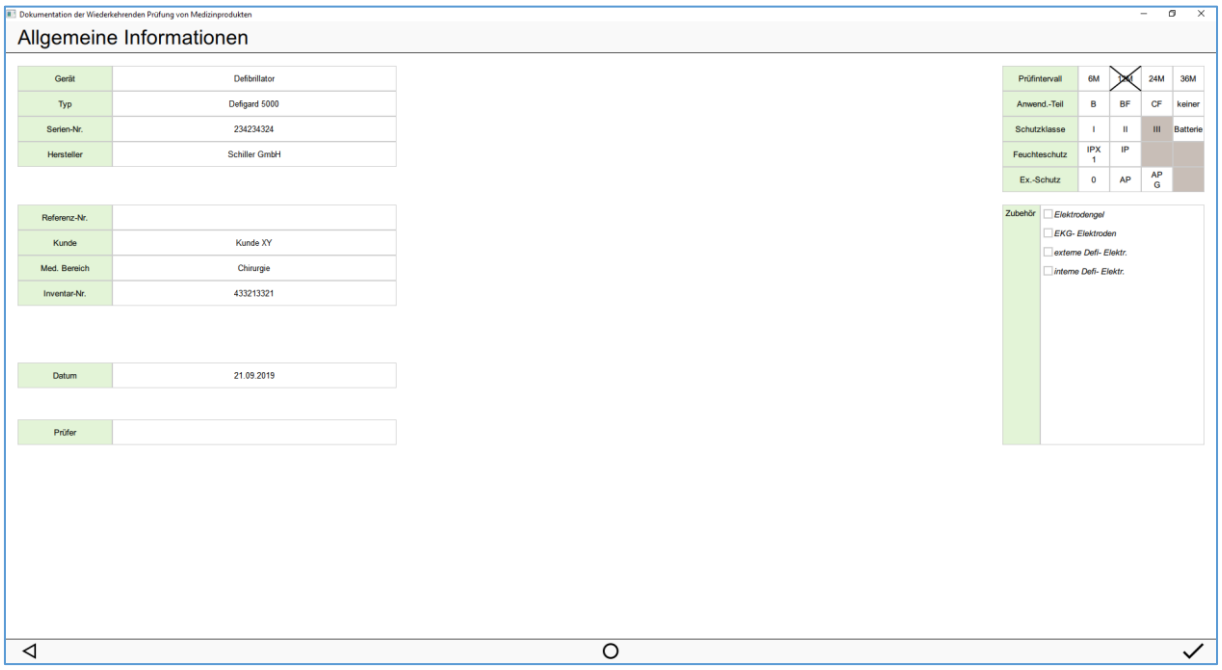


*Abbildung 13: Geräteprüfung „Allgemeine Informationen“*

Wie in Abbildung 13 ersichtlich, befinden sich auf der linken Seite verschiedene Felder, die einzeln ausgefüllt werden müssen:

- Gerät
- Typ
- Serien-Nr.
- Hersteller
- Referenz-Nr.
- Kunde
- Med. Bereich
- Inventar-Nr.
- Datum
- Prüfer

„Kunde“ ist die einzig nicht editierbare Zeile in dieser Spalte, da die Geräteprüfung bereits einem bestimmten Kunden zugewiesen ist.

Das „Datum“-Feld wird beim ersten Betreten der Geräteprüfung automatisch gesetzt.

Bei „Prüfer“ können Sie aus den bereits angelegten Prüfern (siehe Kapitel 3.1) wählen.

Auf der rechten Seite sind bei den Zeilen

- Prüfintervall
- Anwend.-Teil
- Schutzklasse
- Feuchteschutz
- Ex.-Schutz

die zutreffenden Elemente anzukreuzen.

Darunter befindet sich eine ankreuzbare Liste für das etwaige Zubehör des Gerätes.

Falls beim Anklicken des Bestätigen-Buttons unvollständige ausgefüllte Komponenten vorhanden sind, erscheint ein entsprechender Hinweis.

#### 4.2.1.3.2 Sichtprüfung außen und Funktion

Auf diesen beiden Seiten können Sie die Prüfpunkte editieren.

Bei den Mängelstufen 1-3 muss beim Ergebnis etwas eingegeben werden.

Ist im Ergebnisfeld eine Einheit vorhanden, so muss auch im „o.k.“-Fall ein Ergebnis vorhanden sein.

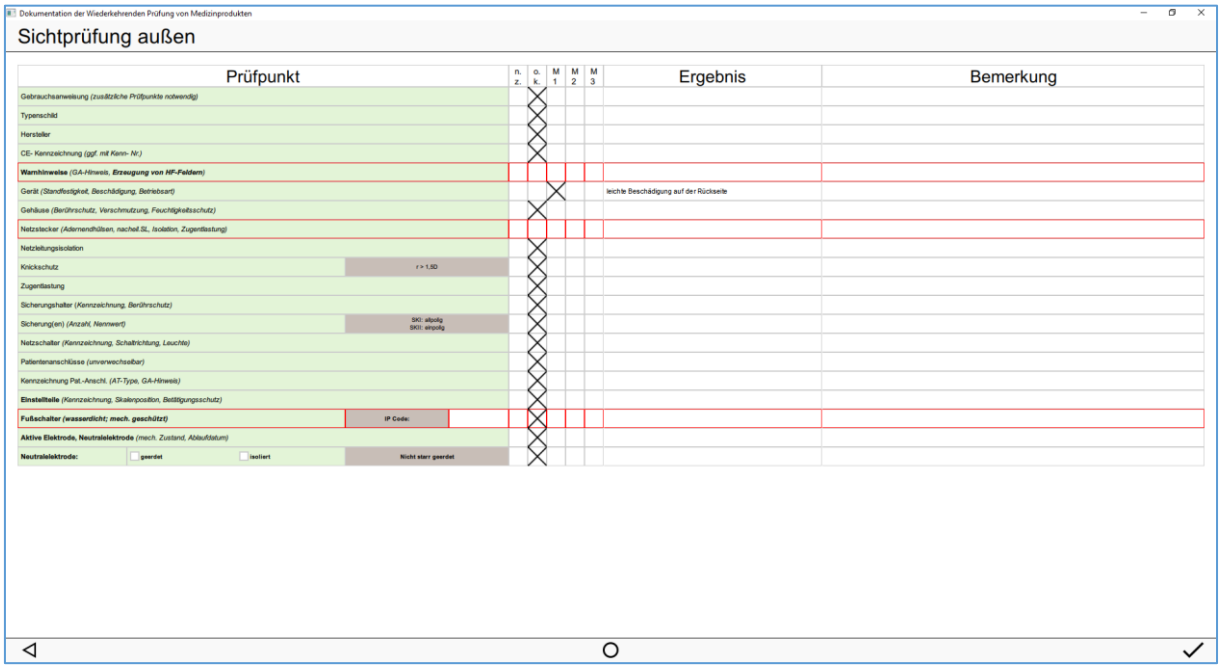


*Abbildung 14 : Unausgefüllte Prüfpunkte „Sichtprüfung außen“*

Wie in Abbildung 14 dargestellt, wird beim Betätigen des Bestätigen-Buttons auf nicht ordnungsgemäß ausgefüllte Prüfpunkte hingewiesen (rot hervorgehoben).

#### 4.2.1.3.3 Normauswahl, sicherheitstechnische Parameter und Abschluss

Zum Fortfahren müssen Sie zunächst die Norm setzen, auf deren Grundlage die durchzuführenden Prüfpunkte der sicherheitstechnischen Parameter ausgewählt werden. Hierfür gibt es zwei Optionen:

- EN 60601
- EN 62353

Sobald die Norm gesetzt ist, können Sie dies nicht mehr rückgängig machen.
Bei Betätigung des Bestätigen-Buttons werden nicht ordnungsgemäß ausgefüllte Prüfpunkte rot hervorgehoben.

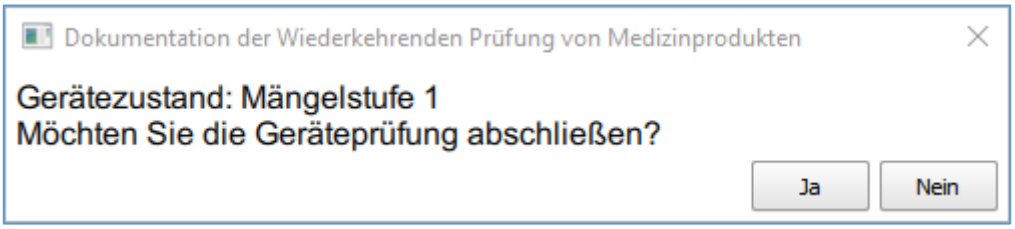


*Abbildung 15: Gerätezustand und Abschluss der Geräteprüfung*

Wurden alle Prüfpunkte der sicherheitstechnischen Parameter und auch alle vorangehenden Seiten der Geräteprüfung korrekt ausgefüllt, so wird, wie in Abbildung 15 dargestellt, der Gerätezustand angezeigt. Außerdem können Sie an dieser Stelle die Geräteprüfung abschließen.

### 4.2.2 Abgeschlossene Geräteprüfungen

In dieser Sektion werden sämtliche abgeschlossenen Geräteprüfungen angezeigt. Geräteprüfungen gelten als abgeschlossen, wenn Sie sämtliche Felder ordnungsgemäß ausgefüllt haben.
Im Gegensatz zu den offenen Geräteprüfungen (siehe Kapitel 4.2.1) können Sie abgeschlossene Geräteprüfungen nur ansehen, jedoch nicht editieren.

#### *4.2.2.1 Geräteprüfungen editierbar machen*

Durch Anklicken des Reaktivieren-Buttons können bereits abgeschlossenen Geräteprüfungen wieder editierbar gemacht werden. Hierbei erscheint die gewählte Geräteprüfung wieder unter „offene Geräteprüfungen“ (siehe Kapitel 4.2.1). Sämtliche bisherige Eingaben bleiben erhalten und die Geräteprüfung muss wieder manuell abgeschlossen werden.

## 4.3 Prüfung abschließen

Eine Prüfung kann abgeschlossen werden, wenn Sie alle Geräteprüfungen in der Sektion „Offene Geräteprüfungen“ (siehe Kapitel 4.2.1) abgeschlossen haben.

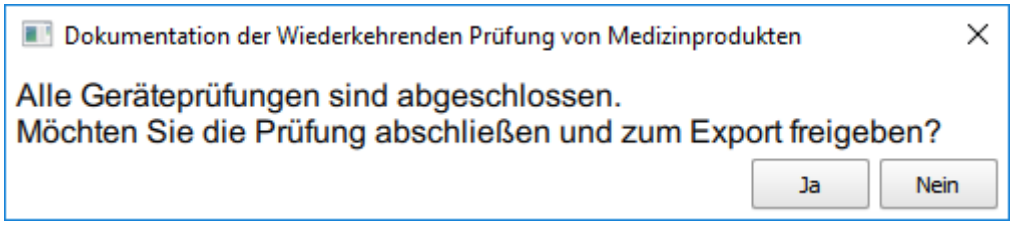


*Abbildung 16: Abschluss einer Prüfung*

Wird die letzte vorhandene Geräteprüfung abgeschlossen, erscheint in dieser Sektion eine Eingabeaufforderung zum Abschluss der Prüfung, wie in Abbildung 16 ersichtlich. Bei Bestätigung wird die Prüfung unwiderruflich abgeschlossen und steht im Programmabschnitt „Berichte“ (siehe Kapitel 5) zum Exportieren bereit.

# 5 Berichte

Dieses Kapitel befasst sich mit dem Export von abgeschlossenen Prüfungen bzw. deren Geräteprüfungen in das Microsoft Word .DOCX-Format.

## 5.1 Kundenauswahl

Suchen Sie zunächst einen Kunden aus der Liste aus und betätigen Sie den Bestätigen-Button.

## 5.2 Auswahl der Prüfung

Hier finden Sie alle abgeschlossenen Prüfungen des zuvor gewählten Kunden. Wählen Sie nun eine Prüfung aus und bestätigen Sie diesen Schritt mit dem entsprechenden Button.

## 5.3 Auswahl der zu exportierenden Geräteprüfungen

Dieser Bereich listet sämtliche Geräteprüfungen der zuvor gewählten Prüfung aus.
Sie können durch Setzten des Kreuzes bei „Alle auswählen“ sämtliche Geräteprüfungen aus- und wieder abwählen. Ansonsten findet die Auswahl durch das Ankreuzen der einzelnen Geräteprüfungen statt.
Durch den Bestätigen-Button kann die Auswahl gespeichert werden, anschließend startet der Exportvorgang. Bitte beachten Sie, dass während der Erstellung der Berichte die Software nicht anderweitig benutzt werden kann. Nach dem Abschluss des Prozesses gelangen Sie zurück zum Hauptmenü.

# 6 Beenden

Hier können Sie die Software herunterfahren und schließen.

# Anhang A: Format der Geräteliste

Die Geräteliste muss im CSV Datenformat hochgeladen werden und hat wie folgt auszusehen:

## Beispiel 1 (Microsoft Excel):

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | Geraet | Med. Bereich | Typ | Hersteller | Seriennr. | Inventarnr. | Pruefintervall | Aussortiert |
| 2 | DEFI | Chirurgie | Defigard 5000 | Schiller GmbH | 234234324 | 433213321 | 12M | false |
| 3 | DEFI | Chirurgie | Defigard 5000 | Schiller GmbH | 924234324 | 344213321 | 12M | true |
| 4 | HFCG | Chirurgie | Erbe VIO 200 S | Erbe Elektromedizin Ges.m.b.H | 623123213 | 578542452 | 12M | false |
| 5 | LASG | Chirurgie | BEACON Advanced Energy Laser System | OmniGuide, Inc. | 342543543 | 478678654 | 12M | false |
| 6 | INFP | Kardiologie | Perfusor Space | B.Braun Melsungen AG | 834214234 | 645434534 | 12M | false |
| 7 | PKRB | Kardiologie | Economic II | Burmeier GmbH & Co. KG | 423123213 | 754789879 | 24M | false |
| 8 | EMGE | Kardiologie | Corpuls 3 | GS Elektromedizinische GerÃ¤te G. Stemple GmbH | 755523213 | 343452542 | 24M | false |
| 9 | REIZ | Kardiologie | EM 80 | Beurer GmbH | 874223213 | 664645354 | 12M | false |

## Beispiel 2 (Windows Texteditor):

```
Geraet;Med. Bereich;Typ;Hersteller;Seriennr.;Inventarnr.;Pruefintervall;Aussortiert
DEFI;Chirurgie;Defigard 5000;Schiller GmbH;234234324;433213321;12M;false
DEFI;Chirurgie;Defigard 5000;Schiller GmbH;924234324;344213321;12M;true
HFCG;Chirurgie;Erbe VIO 200 S;Erbe Elektromedizin Ges.m.b.H;623123213;578542452;12M;false
LASG;Chirurgie;BEACON Advanced Energy Laser System;OmniGuide, Inc.;342543543;478678654;12M;false
INFP;Kardiologie;Perfusor Space;B.Braun Melsungen AG;834214234;645434534;12M;false
PKRB;Kardiologie;Economic II;Burmeier GmbH & Co. KG;423123213;754789879;24M;false
EMGE;Kardiologie;Corpuls 3;GS Elektromedizinische Geräte G. Stemple GmbH;755523213;343452542;24M;false
REIZ;Kardiologie;EM 80;Beurer GmbH;874223213;664645354;12M;false
```

## Legende:

- **Geraet:** Kurzbezeichnung des medizinischen Geräts mit folgenden Optionen:
  - REIZ (Reizstromgerät)
  - DEFI (Defibrillator)
  - EMGE (Elektromedizinisches Gerät)
  - HFCG (HF-Chirurgiegerät)
  - INFP (Infusionspumpe)
  - LASG (Lasergerät)
  - PKRB (Pflege-/Krankenbett)
- **Med. Bereich:** Bezeichnung des medizinischen Bereichs, in dem das medizinische Gerät zum Einsatz kommt.
- **Typ:** Typenbezeichnung des medizinischen Geräts
- **Hersteller:** Hersteller des medizinischen Geräts
- **Seriennr.:** Seriennummer des medizinischen Geräts
- **Inventarnr.:** Inventarnummer des medizinischen Geräts
- **Pruefintervall:** Prüfintervall des medizinischen Geräts in Monaten mit folgenden Optionen:
  - 6M
  - 12M
  - 24M
  - 36M

- **Aussortiert:** Angabe, ob das medizinische Geräte im Einsatz oder aussortiert ist, mit folgenden Optionen:
  - true (= aussortiert)
  - false (= im Einsatz)

# Anhang B: Pfadangaben

## Default-Pfad Gerätelisten

Zur Hinterlegung der kundenspezifischen Gerätelisten im .CSV-Datenformat:

- Build-Verzeichnis/config/clients_devices_csv/

## Default-Pfad Geräteprüfungsvorlagen

Pfad zur Hinterlegung der Geräteprüfungsvorlagen im .DOCX-Datenformat:

- Build-Verzeichnis/config/deviceinspectiontemplates/