FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Design and Development of a Clinical Annotation and Visualization Tool

Inês Castro Teiga

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Hélder Filipe Pinto de Oliveira, PhD

Co-Supervisor: Joana Vale Amaro de Sousa, MSc

Co-Supervisor: Tânia Maria Pereira Lopes, PhD

July 31, 2023

Abstract

New medical image analysis (MIA) algorithms for clinical evaluation and treatment are constantly being developed and optimized, attending to the needs of people in the healthcare sector. However, this is a very complex process that implies data visualization and annotation, model training and human-computer interaction through a graphical user interface (GUI). Consequently, it becomes essential to create software tools capable of supporting all these functionalities and methods, which not only help developers, so that they can concentrate on implementing the algorithms based on machine and deep learning, but also favor health professionals.

A well-designed, practical and intuitive GUI is highly desired by the clinical users, more specifically radiologists, as their profession demands urgent and quick responses. Moreover, without a GUI-based software tool, many algorithms are published simply as source code or command-lines that are not suited for users with no training in computer engineering.

Therefore, the objective of this study was to plan and develop an interface that allows users to visualize, annotate, and navigate through medical images, specifically DICOM files, in a clinical environment. The specific requirements included the ability to perform manual annotation using a brush tool, display and annotate 3D images, reopen images to modify annotations, run user-developed scripts for automatic annotations, adapt the graphical user interface, and perform semantic annotation with side surveys.

The study followed a well-defined workplan, that involved designing, implementing, and evaluating the application. Throughout the development process, specific libraries and technologies were used to meet the predefined requirements and rigorous testing and evaluations were conducted to ensure the functionality and usability of the application.

The results of the study showed that the application successfully implemented most of the desired functions. However, certain significant issues and limitations were identified during the testing process. Although these issues do not pose an immediate threat to the application's functionality, they need to be addressed to further enhance its performance. Additionally, the usability of the application was evaluated through surveys administered to actual potential users.

This study successfully designed and developed an interface to assist physicians. The application demonstrated promising results, by providing a practical and intuitive graphical user interface.

Keywords: visualization tool, clinical annotation tool, DICOM format, GUI, software engineering

Resumo

Novos algoritmos de análise de imagens médicas para diagonostico e tratamento de doenças estão constantemente a ser desenvolvidos e otimizados, atendendo às necessidades das pessoas no setor da saúde. No entanto, este é um processo muito complexo, que envolve visualização e anotação de dados, técnicas de aprendizagem computacional e interação humano-computador, por meio de uma interface gráfica do utilizador. Consequentemente, torna-se crucial a criação de ferramentas de software capazes de suportar todas estas funcionalidades e métodos, que não só auxiliam os desenvolvedores de algoritmos baseados em aprendizagem computacional, para que estes se possam concentrar nas partes importantes da sua implementação, como também favorecem os profissionais de saúde.

Uma interface gráfica do utilizador bem projetada, prática e intuitiva é altamente desejada pelos usuários clínicos, mais especificamente os radiologistas, pois as suas profissões exigem respostas urgentes e rápidas. Além disso, sem uma interface gráfica, muitos algoritmos são publicados simplesmente como código-fonte ou linhas de comando, que não se revelam adequadas à atualização por parte de um utilizador sem formação na área da engenharia informática.

O objetivo deste estudo foi planear e desenvolver uma interface que permita aos utilizadores visualizar, anotar e navegar por imagens médicas, especificamente ficheiros DICOM, num ambiente clínico. Os requisitos incluem a capacidade de realizar anotações manuais utilizando uma ferramenta de pincel, exibir e anotar imagens 3D, reabrir imagens para modificar as anotações, executar scripts desenvolvidos pelo utilizador para anotações automáticas, possuir uma interface gráfica do utilizador adaptável e realizar anotação semântica com inquéritos adicionais.

O estudo seguiu um plano de trabalho bem definido, que envolveu o design, implementação e avaliação da aplicação. Ao longo do processo de desenvolvimento, foram utilizadas bibliotecas e tecnologias específicas para cumprir os requisitos predefinidos, e foram realizados testes rigorosos e avaliações para garantir a funcionalidade e usabilidade da aplicação.

Os resultados do estudo demonstraram que a aplicação implementou com sucesso a maioria das funções desejadas. No entanto, foram identificados alguns problemas e limitações significativas durante o processo de testes. Embora esses problemas não representem uma ameaça imediata à funcionalidade da aplicação, é necessário resolvê-los para melhorar ainda mais o seu desempenho. Além disso, a usabilidade da aplicação foi avaliada por meio de inquéritos administrados a potenciais utilizadores.

Este estudo conseguiu projetar e desenvolver com sucesso uma interface para auxiliar os médicos. A aplicação demonstrou resultados promissores, fornecendo uma interface gráfica de utilizador prática e intuitiva.

Palavras-chave: ferramenta de visualização, ferramenta de anotação clínica, formato DICOM, interface gráfica do utilizador, engenharia de software

Acknowledgments

This work is financed by National Funds through the FCT - Fundação para a Ciência e a Tecnologia, I.P. (Portuguese Foundation for Science and Technology) within the project LUCCA, with reference 2022.03488.PTDC.

The collaboration with Dr. Silvia Dias and Dr. Miguel Castro, radiologists from Centro Hospitalar Universitário de São João, as well as members from the INESC TEC VCMI team, also contributed greatly to the success of this work through usability evaluation and clinical insights.

"De nenhum fruto queiras só metade."

Miguel Torga

Contents

List of Figures

List of Tables

Abbreviations

2D	Two Dimensions
3D	Three Dimensions
AI	Artificial Intelligence
CLI	Command Line Interfaces
CT	Computerized Tomography
DBT	Digital Breast Tomosynthesis
DICOM	Digital Imaging and Communications in Medicine
GUI	Graphical User Interface
HCI	Human-Computer Interaction
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
ITK	Insight Toolkit
JaRRViS	Java-based Remote Viewing and Reporting System
MDL	Micro Development Language
MIA	Medical Image Analysis
ML	Meta Language
MRI	Magnetic Resonance Imaging
MVC	Model-View-Controller
OO	Object-oriented
OS	Operating System
PIL	Python Imaging Library
PL	Programming language
SDLC	Software Development Life Cycle
TIFF	Tagged Image File Format
UI	User Interface
US	Ultrasound
VR	Virtual Reality
XML	Extensible Markup Language

Chapter 1

Introduction

A software tool is essential to assist the medical evaluation and care, as it grants visualization and annotation of various types of clinical examinations, in a quick and trustworthy way. Ideally, the application to be developed should be able to open, display and save images as Digital Imaging and Communications in Medicine (DICOM) formatted images, as it is the most used format in the clinical environment [7]. Simultaneously, it needs to improve the intelligibility and clearness of the represented details in medical images of diverse modalities, like computerized tomography (CT), magnetic resonance imaging (MRI) and ultrasound (US) [8]. Moreover, this software tool should not only allow the performance of manual image annotation, but also support a plugin interface to run scripts [9].

Manual annotation can be noticeably useful for generating a high-quality dataset, however, it is often toilsome, exhausting and time-consuming. Both algorithm developers and health professionals are interested in simple, flexible and time-sparing tools, so the need for automatic and semi-automatic functionalities increased. Medical image analysis and processing steadily benefit from the extensive recent research on artificial intelligence (AI), machine learning and specially deep learning. However, to effectively and expertly incorporate several image data is necessary to recognise complex health conditions, so the doctors are capable of providing the finest prescription to patients [10]. The integration of a scripting plugin come out as remarkably convenient, as it allows developers and users to write a customized script to operate in the program order during a simulation.

1.1 Motivation

Visualization and annotation tools adapted for the clinical user are tremendously significant for disease diagnosis and treatment, contributing for the well-being of the patient [11]. Furthermore, the development of the algorithms to assist the annotation, especially the ones based in machine learning, can be quite convoluted, which makes a simple and convenient GUI particularly interesting for developers. Addressing this subject, many software tools are endeavouring to satisfy these

requirements [9], yet there is still a lot of room for improvement, which makes the research for new tools vastly compelling .

1.2 Objectives and Contributions

The main objective of this dissertation is to plan and develop an interface that allows user visualization, annotation and navigation through medical images, in particular DICOM files, in a clinical environment. Surveys are commonly useful to receive feedback from possible users and should be accessible throughout the course of this Thesis to verify if the tool is appealing and fulfils the requirements presented in Table 1.1.

Table 1.1: Parameters to evaluate clinical visualization and annotation tools.

Requirements	Definition
3D Annotation	Ability of displaying and annotating 3D images
Re-annotation	Possibility of re-opening an image to change the annotation
Algorithm	Capacity of running user-developed scripts
Configurable GUI	Capability of adapting the graphical user interface
Manual Annotation	Possibility of performing graphical and semantic manual annotations

During the elaboration of this dissertation the following contributions in the engineering and medical departments were produced:

• Construction of an accessible and clean graphical user interface (GUI) that supports the human-computer interaction, appropriated for clinicians;

• Implementation of distinctive and advantageous functionalities for manual annotation, like brushes to draw and erase, copy and paste annotations, and options to automatically close the drawn lines;

• Development of a scripting plugin interface to run developed algorithms and be able to adjust the results after.

• Incorporation of adaptable surveys to assist the annotations with linguistic notes.

1.3 Document Structure

Besides introduction, this thesis has 6 more chapters.

Chapter 2 presents a concise study on the medical domain knowledge concerning image analysis and visualization and annotation tools.

Chapter 3 is dedicated to the literature review related to the technological advances concerning programming languages, in particular high level object oriented languages, and user interface design, development and evaluation.

Chapter 4 reports the methodologies, tools and technologies to be implemented and used during the dissertation.

Chapter 5 describes the design and development of the application, explaining each component that composes it and the overall architecture and workflows.

Chapter 6 states the preliminary work and the results of tests and evaluations conducted, followed by a discussion of the strengths and weaknesses of the application.

Chapter 7 condensates some final remarks on the consummated work, and defines the future work.

Chapter 2

Background

This Chapter exposes the fundamental aspects of medical image and some relevant tools to process them, with the aim of understanding the scientific and engineering area where the work is contextualized. Section 2.1 exhibits a study on medical image analysis, focusing CT, MRI and US. The existing tool focused on this sort of medical image analysis are indicated in Section 2.2. Lastly, in Section 2.3, a critical evaluation of the presented tools is proposed.

2.1 Medical Image Analysis

Since Wilhelm Conrad Roentgen accidentally discovered X-rays in 1895, the medical imaging field has evolved into an immense scientific discipline. Medical image analysis is the technique of gleaning meaningful information from medical images, generally using computation methods, to improve the understandability of the illustrated content [12]. The most important tasks for medical image processing are:

- *Image enhancement* – the elimination of image deformities, including background inhomogeneities and noise;
- *Image segmentation* – the recognition and delineation of anatomical structures, namely tumor lesions and organs;
- *Image registration* – the dimensional transformation of an image to fit a certain format. This process is imperative when merging visualization of images from distinct modalities;
- *Quantification* – the determination of physiological characteristics of an anatomical structure, like tissue composition and geometrical characteristics, like diameter and curvature;
- *Visualization* – two-dimensional and three-dimensional synthesis of image data and virtual models of anatomical structures;
- *Computer-aided detection* – the discovery and definition of pathological lacerations.

2.1.1 Imaging Modalities

For the past two decades, the processing of medical image has experienced quick advances with the support of computers.This development has led to various new modalities for medical image analysis of 2D images, 3D volumes and multidimensional figures [13]. The most commonly used nowadays are computerized tomography (CT), magnetic resonance imaging (MRI) and ultrasound (US).

CT is defined by a much greater resolution on high-density tissues, compared to the original imaging modality X-ray, which is used mostly for preliminary medical examination. MRI appears to be more convenient on soft tissue imaging than CT and X-ray, since it has no ionizing radiation during the imagining acquirement procedure. However, it demands a longer time acquiring the images, and has restrictions for patients who wear metal medical instruments, like cardiac pacemakers, as it may have harmful effects on them. Contrasting with the other presented imaging modalities, US is safe, low-cost and broadly applicable to many different situations. Moreover, patients do not have to stay still in certain positions, which permits radiologists to have a real-time vision of the analysed body areas [8].

By virtue of these imaging modalities, the clinical teams can evaluate a vast quantity of anatomical structures in a relative innocuous non-invasive way. Clinical visualization and annotation tools are then essential to assist the health professionals with interpreting medical images and enabling human and computer manipulation.

2.2 Related Tools

Nowadays, there are various clinical tools that stand out for offering different annotation functionalities, that can be manual, semi-automatic and automatic [11]. Testing and analysing them is important to understand the possible functionalities that can be executed and evaluate their utility.

2.2.1 Itk-snap

Itk-snap is a free, open-source and multi-platform software application used to segment structures in 3D medical images. This tool is dedicated to semi-automatic segmentation using active contour mechanisms, manual delineation and image navigation.

The main advantages of itk-snap cover:

- Manual segmentation in three orthogonal planes simultaneously;
- Support for manipulation of numerous images in the same workplace, colour and time-variant images;
- 3D cut-plane tool for the agile post-processing of segmentation results.

However, it lacks the flexibility of shifting to new techniques when the ones offered come as incomplete or inadequate. The Itk-snap GUI is depicted in Figure 2.1.

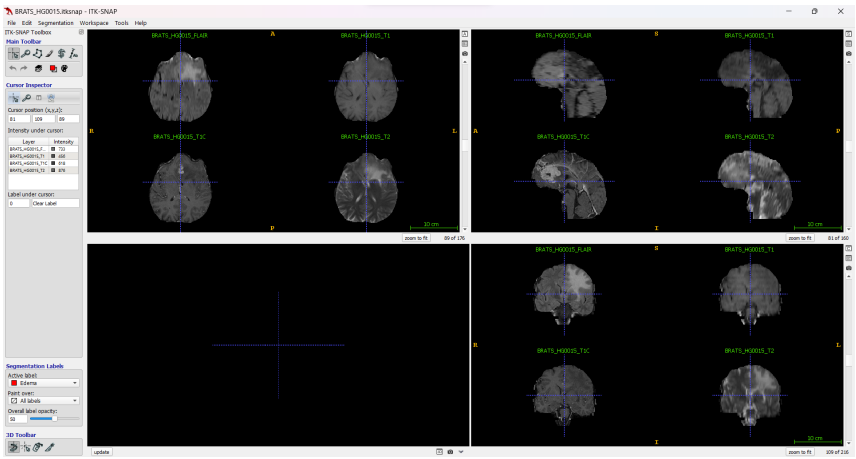


Figure 2.1: Screenshot of the Itk-snap interface captured during testing of the downloaded application [1].

2.2.2 Seg3D

Seg3D is another free, open-source and cross-platform semi-automatic segmentation tool, that also provides powerful image processing algorithms. The Seg3D GUI is depicted in Figure 2.2.

Some of the core advantages are:

- 3D interface with multiple volumes managed at the same time as layers;
- Incorporated algorithms from the Insight Toolkit (ITK), which is a system for medical image processing;
- Real-time demonstration of ITK filtering output;

Contrastingly, it is still missing some basic features associated with image visualization, as it does not allow zooming in and out. Besides, Seg3D has issues with file importation, as it opens the images neglecting their order, which generates anatomical disproportion and incorrect annotation.

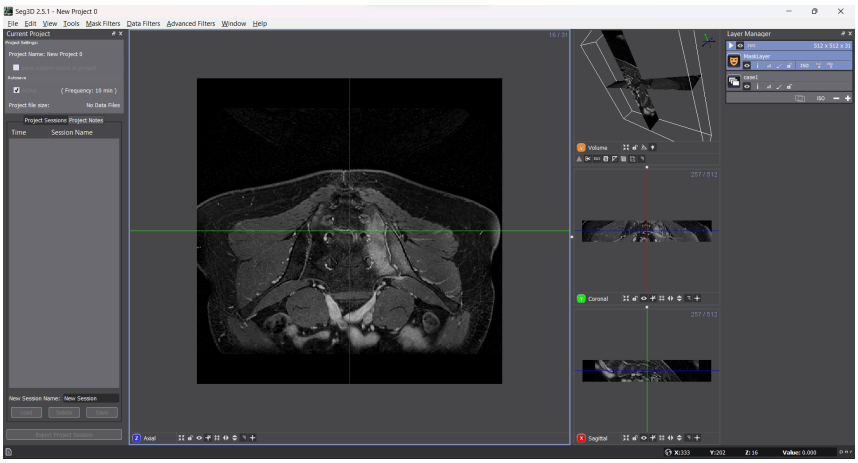


Figure 2.2: Screenshot of the Seg3D interface captured during testing of the downloaded application [2].

2.2.3 MeVisLab

MeVisLab is one of the most impressive multi-platform tools for medical image processing. It combines an Integrated Development Environment (IDE) for visual programming and a very complete text editor MATE for Python scripting, enabling debugging and other advanced functionalities.

Some of the most prominent features are:

- 6D image processing: x, y, z, color, time and user dimensions;
- Scripting support in Python and Micro Development Language (MDL);
- Error handling: configurable exception usage, diagnosis modules and automatic module tester.

Despite being equipped with a large image processing framework, MeVisLab does not seem adequate for manual annotation. Additionally, the non-trial version is not free. The MeVisLab GUI is depicted in Figure 2.3.

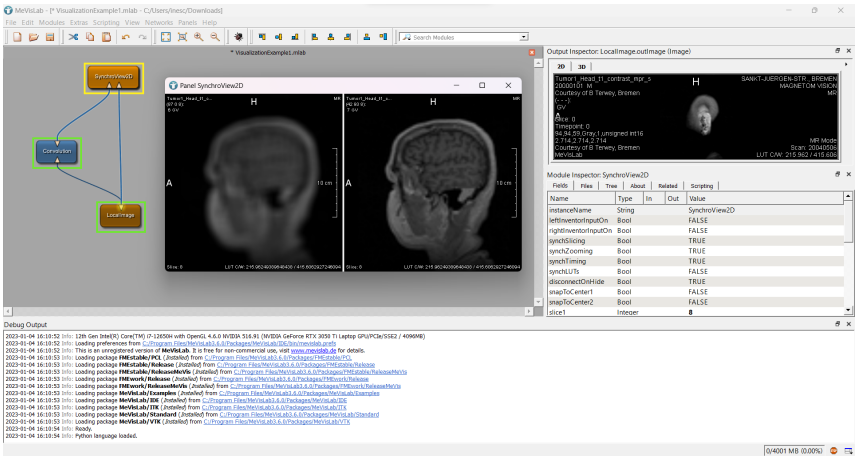


Figure 2.3: Screenshot of the MeVisLab interface captured during testing of the downloaded application [3].

2.2.4 Fiji

Fiji is a distribution of the free, open-source and multi-platform software ImageJ, which incorporates many convenient plugins added by specialists. The core application is rather poor in terms of image-processing methods provided. Instead, it relies on his extensibility to remain relevant.

- Automated and reproducible workflows through scripting;
- Active and helpful user community;

On the other hand, Fiji is not suited for 3D data annotation. The Fiji GUI is depicted in Figure 2.4.

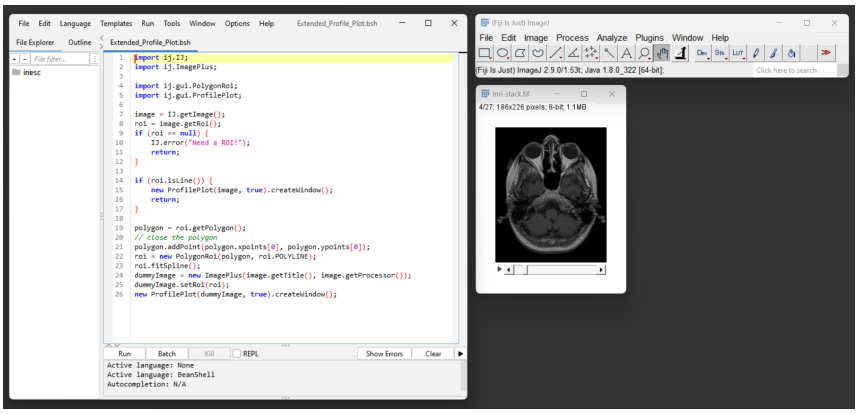


Figure 2.4: Screenshot of the Fiji interface captured during testing of the downloaded application [4].

2.3 Evaluation of Related Tools

2.3.1 Itk-snap

Itk-snap has an eminently clear and intuitive interface for 3D manual annotation in several medical image formats, including DICOM, NIfTI and Mayo Analyze. Workplaces (more than one image) can only be saved in the XML format. Favors semantic annotation and customisation of the GUI. Moreover, there are several tutorials and examples available, that make this tool very appealing to beginners. Scripting, however, is not useful, as it allows algorithms only in their own syntax.

2.3.2 Seg3D

Seg3D reads DICOM files and any image file format supported by the ITK. Authorizes 3D annotation, scripting in Python and has an adaptable GUI, yet the semantic annotation is not very valuable. Highly advantageous for annotation with semi-automatic and automatic methods, by running scripts in Python. However, not helpful with the manual annotation and lacks tutorials and examples.

2.3.3 MeVisLab

MeVisLab allows 3D and even 4D efficient annotation, saved as pairs of DICOM and Tagged Image File Format (TIFF) image files, scripting in Python and MDL and semantic annotation. This tool is the most complete and sophisticated in terms of functionalities and guides accessible, yet it does not permit to adjust the GUI as it is not open source.

2.3.4 Fiji

Fiji manages images mainly in TIFF, but can read some kinds of DICOM files. Includes a configurable GUI and indulges semantic annotation and scripting in various languages, but primarily in

Python. The interface is comparatively modest as it does not allow visualization and manipulation of 3D images nor opens more than one image at a time.

2.3.5 Summary of Evaluation of Related Tools

The summary of the evaluation after testing each of the tools considering the proposed requirements is presented in Table 6.3.

Table 2.1: Summary of the evaluation of the related tools.

Requirements	Itk-Snap	Seg3D	MeVisLab	Fiji
3D Annotation	●	●	●	○
Re-annotation	◉	●	●	◉
Algorithm	○	●	●	●
Configurable GUI	●	●	○	●
Manual Annotation	●	○	●	●

● Implemented	◉ Half Implemented	○ Inutile	○ Missing

Chapter 3

Literature Review

In this Chapter the fundamental models and mechanics that will support the decisions and development of this project are analyzed. Section 3.1 suggests different approaches to define programming languages. The requirements to produce an excellent program according to ISO/IEC 9126-1:2001 Standard are exposed in Section 3.2. In Section 3.3, the categories of programming languages are presented. Section 3.4 lists some high level programming languages that were considered and the strengths and weaknesses of each, comparing them. Finally, in Section 3.5 the characteristics of a user friendly GUI are covered, along with test methods.

3.1 Definition of Programming Language

Programming languages (PL) are means for telling computers what to do. It consists in a relationship of dominance, with the programmer controlling a computer, and the computer obeying. However, this two actors do not speak the same language, so perhaps IDEs are like human translators, promoting communication between them.

Programming languages are a set of characters and formal rules to combine them, that the programmer exploits to specify a computer physical and logical behavior. However, this definition is not necessarily the correct one, as it can be defined from various perspectives, related to the way a person designs, evolves and uses the programming language. Some of this views are compiled and explained by Amy J. Ko [14].

It is also possible to view PL as a kind of natural language, considering that they are influenced by some linguistic ideas, reason it is named "language". Another view of PL is as a kind of documentation. The source code is not usually what is executed, but a compiled translated version of it, implying that the true job of a PL is to assist programmers to think and store the development of this logical process. Most modern web developers view PL as a glue between libraries, frameworks, platforms, services and APIs. The developers or engineers that are tasked with building large systems with multiple functions use languages as exceptional connectors between functions. Others view PL as an interface between humans and computers, seeing that both have properties

such as usability, learnability and feedback. A more inspiring view frames PL as notations for describing the future. Blackwell defines programming as an activity in which "the user is not directly manipulating observable things, but specifying behaviour to occur at some future time" [15]. In many situations, programs are expositions of contracted demands. In this view, PL are the mechanisms through which a contract is fulfilled. Although the need for PL knowledge is exponentially growing, as nearly all new products and services are in digital form, it is still limited to a small minority of humanity. For that, PL can also be pictured as a form of power.

3.2 Software Quality Model

ISO/IEC 9126-1:2001 Standard establishes a collection of non-functional requirements that help evaluate the external and internal quality of a program. These requirements are allocated into six different categories: functionality, reliability, usability, efficiency, maintainability and portability, each having subcharacteristics, which influences the quality characteristic [16]. It is essential that a PL is able to not only provide constructs that guide the programmer in the develop of a software programs that respects the below fundamentals, but also complicate the implementation of systems that infringe them to the greatest extent.

3.2.1 Functionality

Ability to provide functions which meet certain needs, under certain conditions. The subcharacteristics are:

- *Suitability* – Ability to provide an appropriate group of functions for a particular task;
- *Accuracy* – Ability to provide the accurate and precise results or effects;
- *Interoperability* – Ability to reach out to one or more specified systems;
- *Security* – Ability to conserve information and data from unauthorised entities and enable access to authorised ones.

3.2.2 Reliability

Ability to sustain a particular level of performance under particular conditions.

- *Maturity* – Ability to avoid failure caused by deficiencies in the program;
- *Fault tolerance* – Ability to sustain a particular level of performance in case of software deficiencies;
- *Recoverability* – Ability to recover data and re-establish a particular level of performance in case of failure.

3.2.3 Usability

Ability to be understood, learned, used and attractive to the user.

- *Understandability* – Ability to empower the user to understand whether the software is appropriate for particular tasks and how it can be used;
- *Learnability* – Ability to facilitate the user to learn the program's function;
- *Operability* – Ability to enable the user to control the program;
- *Attractiveness* – Ability to be pleasing and engaging to the user.

3.2.4 Efficiency

Ability to maintain a suitable performance, corresponding to a certain amount of resources used, under certain conditions.

- *Time behaviour* – Ability to provide suitable processing and response times;
- *Resource utilization* – Ability to operate a suitable amount and type of resources.

3.2.5 Maintainability

Ability to be modified, such as corrections, improvements and adaptations to the requirements, environment and functional specifications.

- *Analysability* – Ability to be investigated for flaws that may cause failure;
- *Changeability* – Ability to permit a singular change to be made;
- *Stability* – Ability to avert unexpected outcomes from the modifications;
- *Testability* – Ability to test the modification so they can be validated.

3.2.6 Portability

Ability to be transported from one environment to another.

- *Adaptability* – Ability to be adapted for various environments without a lot of effort;
- *Intallability* – Ability to be installed in a specific environment;
- *Co-existence* Ability to share resources with other independent software in the same environment;
- *Replayceability* – Ability to be used in place of another software product in the same environment.

3.3 Classification of Programming Languages

3.3.1 Declarative and imperative languages

The many existing programming languages can be allocated into families based on different principals. The model of computation is one of them [17], dividing PL into declarative and imperative languages. While declarative concentrate in commanding the computer to do a specified assignment, imperative concentrate in explaining the computer how to do it.

Within the declarative category, there are several significant subclasses:

- *Functional languages* – apply a computation model based on the recursive definition of functions. Fundamentally, a program is treated as a function, defined in terms of simpler functions. Some examples are Lisp [18], ML [19] and Haskell [20], which take their inspiration from the lambda calculus [21];
- *Dataflow languages* – design computation as the flow of information (tokens) among primitive functional nodes. An example is Val [22];
- *Logic or constraint-based languages* – model computation as an effort to find values that assure certain relationships, adopting a goal-directed search through a list of logical rules. The most popular is Prolog [23].

As for the imperative languages, these are divided into the following subclasses:

- *von Neumann languages* – the most familiar and commonly used, where the fundamental means of computation consist in the modification of variables. They include Fortran [24], Basic, Pascal and C [25];
- *Object-oriented (OO) languages* – comparatively recent, most originate from von Neumann languages. However, they have a much more organized and appropriated model of both memory and computation. Rather than having a monolithic processor operating on a monolithic memory, object-oriented languages have semi-independent objects that interact with each other. These objects have their own attributes and functions to manage that attributes. The purest OO language is Smalltalk [26], but C++ [27] and Java [28] are the most popular.

3.3.2 Typed and Un-typed Languages

Another way to classify the PL is based on whether they have a type system or not. In typed languages, the variables have a limited range of values they can accept, while un-typed languages do not restrain the scope of variables [29].

Typed languages are reported to be statically typed, when they prevent running unreliable and wrong programs through the performance of static checks. However, the type checking may be suspended until runtime, so in this case the language is claimed to be dynamically typed. Examples of object-oriented statically typed languages include Java and C++, while Smalltalk is dynamically typed.

3.3.3 Low Level and High Level Languages

Programming languages can be classified into two categories based on the proximity to machine language and human language. A PL that has a stronger connection to the machine is a low level programming language, while one that is closer to the human speech is a high level programming language.

Low level language allows internal machine communication through instructions, which have unique opcodes (Operation Codes), used to specify the operation to be performed.

High level language simplifies the writing, debugging and maintenance of the instructions that the programmer lends to the machine. The machine absorbs the code in machine language through translators or compilers. A high level language facilitates the work of the programmer, as it uses notations that make it easier to understand the task to be executed, reducing the time of programming and the chance of occurring errors, avoiding frustration. There are numerous high level languages in constant evolution and renovation. Each has pros and cons, so it's up to the programmer to decide which one suits the purpose better.

3.4 High Level Programming Languages

Multiple programming languages have been matured and released over the years, therefore it became difficult to find the right programming language for a certain job. A possible solution is to analyse the pros and cons of the most popular programming languages and compare them.

3.4.1 Java

Java is one of the most popular programming languages, even in the clinical area, with various applications such as Java-based Remote Viewing and Reporting System (JaRRViS) in the nuclear medicine department [30], and RetinaView, that allows manual analysis as well as automatic characterization and segmentation of multi-modality retinal images and videos [31].

Java was firstly designed for use on the Internet, envisaging benefits for both users and developers. However, it goes beyond the Web domain, since it can be useful to produce a variety of robust applications that do not depend on network features [32].

This language is extremely easy to learn and use, considering it has automatic memory management, garbage collection and type-safe references. Moreover, it is object-oriented, which facilitates the recycling of previously written portions of the program [33] and the structured arrangement of the code into objects. It has several highly extensible and convenient containers in widget toolkits namely JavaSwing and JavaFX.

Java is designed for maximum portability, following the concept "write once, run anywhere". This involves less work for the user, as the installation of a compiler for that language is not needed. It also benefits the developer, that only has to code once, with no need to "port" their applications to every platform.

However, the variety of alternatives in the development of graphical user interfaces is considerably limited compared to other programming languages, leading to embryonic GUIs and poor user experience.

3.4.2 C++

C++ is a similar alternative to Java, both support object-oriented programming and are focused on application and system development. Some examples of its employment in medicine is the Digital Breast Tomosynthesis (DBT) imaging system, that provides 3D reconstructed images to detect breast cancer [34], and ImLib3D, which is a C++ library designed for 3D medical image processing [35].

While Java provides greater diversity and flexibility, C++ proves to be a better option when you need a high-performance application, as it is much faster than Java. Furthermore, it is a reliable programming language for programming GUIs, as there are several libraries and frameworks for developing GUIs available, such as Qt platform [36].

However, this language appears to be noticeably harder to learn and use, due to the complex manual memory allocation method, especially in topics like pointers and arrays [36]. Additionally, it is platform-dependent, following the concept of "write once, compile anywhere".

3.4.3 C#

C# (C-sharp) is a general-purpose, object-oriented programming language developed by Microsoft. Since Microsoft is also the creator of Windows, this language allows the creation of secure and valuable Windows-based desktop applications. Applications written in C# use .NET, which is is a platform and programming framework for cross-platform development [37], so both technologies are often seen as inseparable.

This language is used across a diversity of platforms, mainly for visual programming purposes, including Microsoft Visual Studio, where it is used together with .NET framework for visual coding, and Unity, where it is used for game development [38]. An example of an application related to medicine is the educational game that uses virtual reality (VR) tools to teach pathologies of the cardiovascular system to medical students [39].

Both C# and C++ languages are obtained from C, therefore they find resemblances with syntax and symbols of the C language [38]. One of the most prominent features of the C language is that it can access memory directly through pointers. As C# is a much higher level language, the range of available pointer resources is still very limited. Furthermore, it is not as solid as C++ in terms of performance, is more complicated to learn and use than Java and is overly dependent on the .NET platform, which requires the payment of a license from Microsoft.

3.4.4 Python

Python is a high-level general purpose programming language. It is very easy to understand, open source and performs tasks with tremendous versatility, becoming one of the most popular

language in the world. As Python frequently uses object-oriented programming and its code is automatically compiled to byte code and executed, it is suitable for use as a scripting language and Web applications implementation language [40]. Python is mainly used for machine learning, neural networks, data science, data analysis, data mining, data visualization, web application development and testing and desktop application development and testing [38].

Assignable to high-level data types and inherent dynamic typing, Python allows programmers to convey tasks with fewer code than in languages like Java and C++. It is extremely compatible with multiple operating systems, like Linux, Windows, Mac OS, Amigo and UNIX, is extensible and embeddable, as it can easily incorporate pieces of other languages with its code [41] and has a large community that can assist in case any issue occur [38].

Additionally, a vast number of valuable libraries help the programmer solve common challenges, such as graphical user interface creation. Some of the most prominent are Tkinter, WxPython, Kivy, PyGUI and PySide [42], which provide the fundamental tools for GUI designing. There are also libraries focused on medical image analysis, like Highdicom [43] and PyOsiriX [44]. Moreover, Python has several computer vision frameworks that facilitate the adaptation and integration of algorithms on an application, including OpenCV, Fastai, IPSDK, Imutils, Pytessarct, PyTorchCV and SimpleCV [42].

However, its fluidity slows Python performance down, as the machine takes some time to understand the definitions through a lot of referencing. It may get hard to track down and fix errors due to its dynamic typing [41] .

3.4.5 Summary of High Level Programming Languages

The summary of the evaluation of some popular high level programming languages considering some specific features is presented in Table 3.2. This features are defined in Table 3.1

Table 3.1: Glossary of the evaluation features to compare programming languages.

Features	Definition
Simplicity	Ease and clarity in learning, reading and writing code
Execution time	Low amount of time spent by the system to complete a task
Portability	Ability to run the same program on different environments
Security	Ability to protect the assets from unauthorized action
Memory consumption	Low amount of memory spent by the system to complete a task
GUI creation	Ease and clarity in creating good graphical user interfaces

Table 3.2: Level of support of the features for different programming languages.

Features	Java	C++	C#	Python
Simplicity	●	○	◍	●
Execution time	◍	●	◍	○
Portability	●	●	◍	●
Security	●	○	○	●
Memory consumption	○	●	◍	◍
GUI creation	○	●	●	●

● Full support	◍ Partial support	○ Poor support

3.5 User Interface

Human-computer interaction (HCI) is the study of the interaction between humans and computers. As computer use became more common, the interest of researchers in the physical, psychological and theoretical aspects of this interaction intensified. HCI draws on many disciplines, including psychology, ergonomics, sociology, engineering, business, computer science and system design [45].

When users interact with a computer system, they do so through a user interface (UI), that include various natures of synergy, like command line interfaces (CLI) and graphical user interface (GUI). The user interface defines the first impression that the user has of an application, which is why it is its most significant part [46].

3.5.1 User Interface Design

A good UI design provides an easy, natural and engaging interaction between a person and a system, and it allows users to perform all the tasks they wish to undertake with minimum frustration. Furthermore, the user should be able to use the machine without extensive knowledge of how it works internally. However, describing a user interface as "good" or "bad" is worth little, as these are subjective words. They may be used to characterize the aesthetic attributes of the UI. Yet, the real interest is defining the quality of a user interface in relation to its usability [46].

Usability is defined in Part 11 of the ISO 9241 standard [47] as the "extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use".

- Effectiveness is the accuracy and completeness with which specified users can achieve specified goals in particular environments;

- Efficiency is the resources expended in relation to the accuracy and completeness of the goals achieved;

- Satisfaction is the comfort and acceptability of the work system to its users and other people affected by its use .

ISO 9241 portrayal of usability is much more than just the words "effectiveness", "efficiency" and "satisfaction". The expressions "specified users", "specified goals" and "specified context of use" are equally significant to define usability, mainly in the aspect of security [48]. The specified users are the users for whom the system has been designed and developed. A user-friendly GUI has the user as a focus for the design of the interface, but also looks at the wider context within which the system is expected to operate (domain, tasks and environment).

Poorly designed user interfaces may result in user dissatisfaction and even safety issues, that can be easily avoid creating evaluation methods and inserting them into the GUI development life cycle.

3.5.2 User Interface Evaluation

Evaluation is a crucial procedure in the development of a user interface, not only to find errors, but also to measure the performance of the system. Particular techniques of evaluations can be executed at particular timings and for particular reasons. Regardless, it often leads to product improvement [46].

Some of the techniques are:

- Observing how people interact with the product in a specific environment;

- Interviewing and questioning the people about their opinions towards the product;

- Making predictions about the problems the user will encounter, without having to test with real users.

Measures of effectiveness, efficiency, and satisfaction to visualization and annotation tools can be easily quantified by a model adapted from the proposed model by Kyung S Park and Chee Hwan Lim [5], providing solid information about the general usability of the application. The model presented in Figure 3.1 consists of two main phases: the expert evaluation phase and the user evaluation phase.

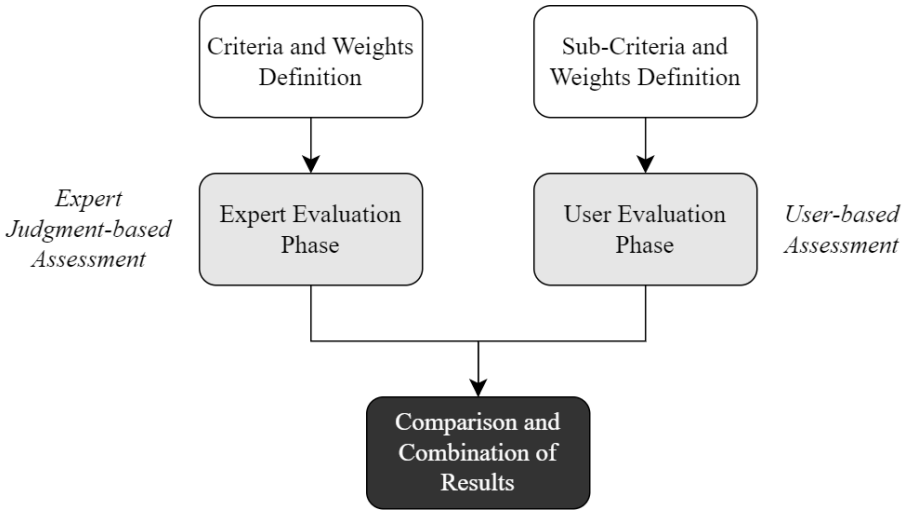


Figure 3.1: Overall procedure of the user interface evaluation model adapted from the proposed model by Kyung S Park and Chee Hwan Lim [5].

3.5.2.1 Expert Evaluation Phase

In the expert evaluation phase, engineers use their experience and user interface design guidelines to assess the application. This phase relies on expert judgment and requires clear criteria. Multiple specialists should be involved to minimize subjectivity. Usability criteria are dimensions recognized as leading to better user behavior and should be weighted depending on the importance. Some example criteria for evaluation can be as follows:

- *Suitability for the Task:* The system should meet users' needs and requirements when performing tasks.
- *User Control:* Users should have control over the interface as much as possible.
- *Flexibility:* The interface should be adaptable and customizable to suit the needs of different users.
- *Error Management:* The system should minimize user errors and provide mechanisms for error prevention and correction.
- *Compatibility:* The system should align with user conventions and expectations.
- *Self-descriptiveness:* The system should provide feedback, guidance, and support to help users understand and use it effectively.
- *Consistency:* The system's appearance and behavior should remain consistent.
- *User Workload:* The system should minimize the cognitive load on users, including memory load, and promote efficient interaction.

3.5.2.2 User Evaluation Phase

In the user evaluation phase, the focus is on assessing the application using user testing data to evaluate usability. Various quantifiable measures can be utilized, including time to complete tasks, success rate, error rate, user satisfaction, and more. These measures can be obtained through direct measurement, interviews, or questionnaires.

Usability parameters can be categorized into objective performance measures, which assess user capability in using the system, and subjective user preference measures, which evaluate user satisfaction with the system. To evaluate effectiveness, efficiency, and satisfaction, these criteria are further decomposed into sub-criteria or usability measures. An example of sub-criteria or measures are the following:

Effectiveness:

- Percentage of users successfully completing task;
- Number of user errors;
- Ratio of successful interactions to errors;
- Number of tasks completed in a given time;
- Average accuracy of completed tasks.

Efficiency:

- Time to complete a task;
- Number of references to help;
- Effort (cognitive workload);
- Tasks completed per unit time;
- Time spent using help or documentation;
- Learning time.

Satisfaction:

- Rating scale for user satisfaction;
- Proportion of users who say they would prefer using the system over that of some specified competitor;
- Proportion of user statements during a test that are positive versus critical;
- Frequency of complaints.

The choice of measures and weights depends on the project's objectives and available resources. It is important to ensure that the selected measures are independent of each other. The outcomes of the two phases are compared and/or combined using a formula that acknowledges the significance of each criteria to the specific project.

3.6 Summary of Literature Review

This chapter provides an overview of various aspects related to programming languages and user interface design and evaluation. The chapter begins by defining programming languages as a means of communication between programmers and computers, highlighting the need for effective communication. The ISO/IEC 9126-1:2001 Standard is introduced as a quality model that establishes non-functional requirements for evaluating the external and internal quality of software programs. It emphasizes the importance of programming languages in providing constructs that guide programmers in developing software programs that adhere to these fundamental quality characteristics. The chapter then explores the classification of programming languages based on different criteria. The focus shifts to high-level programming languages, including Java, C#, C++, and Python. The pros and cons of each language are discussed, with Python emerging as the language that best supports the evaluated features.

The significance of user interface design and evaluation is highlighted, with an emphasis on providing an easy, natural, and engaging interaction between users and systems. The concept of usability is defined based on ISO 9241 standard, which considers the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specific context of use.

The chapter emphasizes the importance of evaluation in the development of user interfaces to identify errors and measure system performance. Different evaluation techniques are discussed, highlighting their timing, purpose, and contribution to product improvement. The proposed model, adapted from Kyung S Park and Chee Hwan Lim, povides quantifiable information about the overall usability of an application. It offers advantages from both experts and users perspectives. It is especially beneficial when dealing with multiple criteria and when conducting usability evaluations with limited resources.

Chapter 4

Methodology

In this chapter, the guidelines conducive to the successful development of the project are revealed. Section 4.1 briefly presents a plausible workplan for the development of the application, while Section 4.2 delves into the selection and justification of the libraries, tools, and technologies implemented, outlining the rationale behind the choices and their relevance to the project's objectives. It aims to establish a solid foundation for the technological decisions made throughout the application's development.

4.1 Workplan

Currently, there are several different approaches to organizing the software development process, known as software development life cycle (SDLC) models. SDLC models outline the steps involved in creating software. The chosen model plays a significant role in determining the success of a project in terms of quality, cost, timeline, and stakeholder satisfaction. Some of the more popular models include waterfall, spiral, v-model, iterative, and incremental. Each of these models has its own benefits and drawbacks.

The iterative and incremental methods involve building a system through repetitive cycles of development, working on smaller parts of the system at a time. This approach allows software developers to incorporate what they have learned during the development of earlier parts of the system into the current version. By releasing various versions of the project throughout the development process, it is easier to gather customer feedback and make adjustments based on changing requirements, resulting in a lower cost for accommodating these changes. The proposed workplan for the dissertation is based on the incremental and iterative SDLCs, as demonstrated in Figure 4.1.

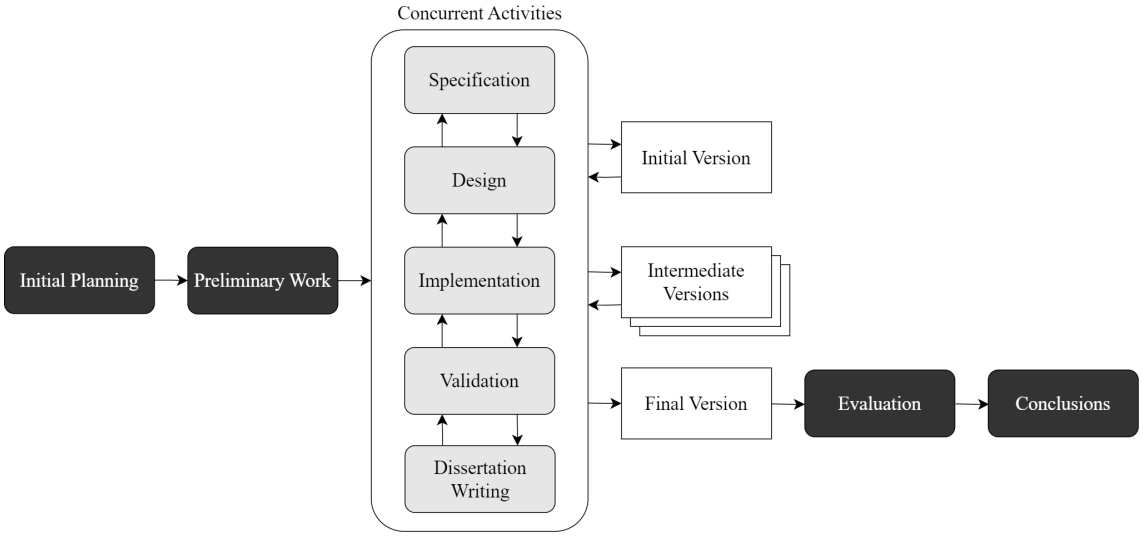


Figure 4.1: Workplan for the dissertation rooted on the incremental and iterative SDLCs.

Every phase of the cycle is essential on its own, but when all the stages are considered together, the collective impact is greater than the sum of its parts. The workplan can be divided into three main stages: the initial stage, the intermediate stage, and the final stage.

4.1.1 Initial Stage

The first stage involves conducting *Initial Planning*, which is later documented in the dissertation. The *Initial Planning* stage is where the concept for the software is developed. During this phase, the problem is identified, pertinent information is gathered to plan a solution, and all available and relevant literature is reviewed. The process begins with a broad range of ideas and narrows down to a specific plan with well defined steps. This was accomplished in the scope of the Introduction to Research curricular unit. The *Preliminary Work* is also pivotal in this stage, as it complements the data analysis, providing the necessary foundation for moving on to the next stage.

4.1.2 Intermediate Stage

The intermediate stage involves the concurrent *Specification*, *Design*, *Implementation*, *Validation* of both back-end and front-end systems, as well as the integration of *Dissertation Writing*.

The *Specification* stage involves clearly defining the capabilities of the system. This phase involves establishing the necessary services of the tool and the constraints on its operation and development. It is divided into requirements elicitation and analysis, in which the requirements are identified; requirements specification, in which the requirements are defined in detail; and requirements validation, in which the validity of the requirements is checked.

Design and implementation activities are closely related, as they both involve turning the application specification into a functioning system. During the *Design* stage, the developer creates a software structure that meets the specification, while the *Implementation* stage involves turning this structure into a program that can be executed.

The *Validation* stage is intended to demonstrate that a system meets its specifications and the user's requirements. It involves checking and reviewing processes and testing the system. Testing is the most common type of validation activity. System testing implicates running the system with test cases that are based on the specification of the actual data that will be processed by the tool. Additionally, a phase of acceptance tests is planned, during which the application will be evaluated by actual potential users, physicians.

4.1.2.1 Back-end Development

Back-end development is the process of developing the logic that retrieves and organizes data for websites and applications. The back-end is the machinery that operates behind the scenes, powering what the end user sees and interacts with, but is not directly visible to the user, like the program architecture and components.

During architectural design, the overall structure of the system is identified, as well as the main components (subsystems or modules), their relationships, and how they are distributed. Component selection and design involves searching for reusable components and, if they are not available, designing how they will operate.

Testing is a crucial activity for recognizing and fixing program faults, and should be executed dynamically throughout the back-end development, as depicted in Figure 4.2.

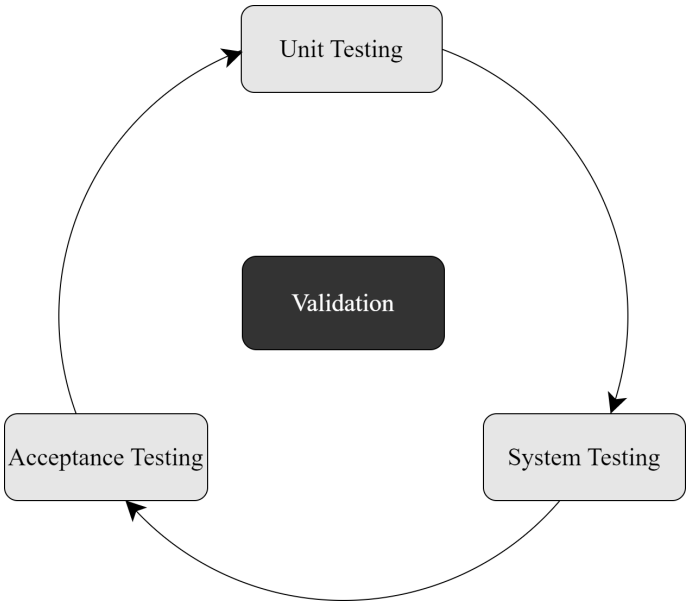


Figure 4.2: Iterative back-end validation process.

In unit testing, individual components are tested independently. These components can be functions, objects, or groups of these entities. System testing involves testing the system as a whole. It is particularly important to test emergent properties during this process. The purpose of acceptance testing is to evaluate the program using customer data and determine if it meets

the user's expectations. By testing the software in the intended environment before it is used in a real-world production, it is possible to achieve better results.

4.1.2.2 Front-end Development

To fully understand the back-end, it is also necessary to understand the front-end and how the two work together. The front-end is what the end user sees and interacts with in the interface.

In an iterative GUI life cycle, the design, prototyping and evaluation stages overlap and notify each other, as demonstrated in Figure 4.3. This model has the user in the center, following the proposition that the user should be involved in the creation of the interface for a appreciably accelerated and cost-effective process, rather than building the end product and having to make expensive alterations in case it is not gathering all the user requirements.

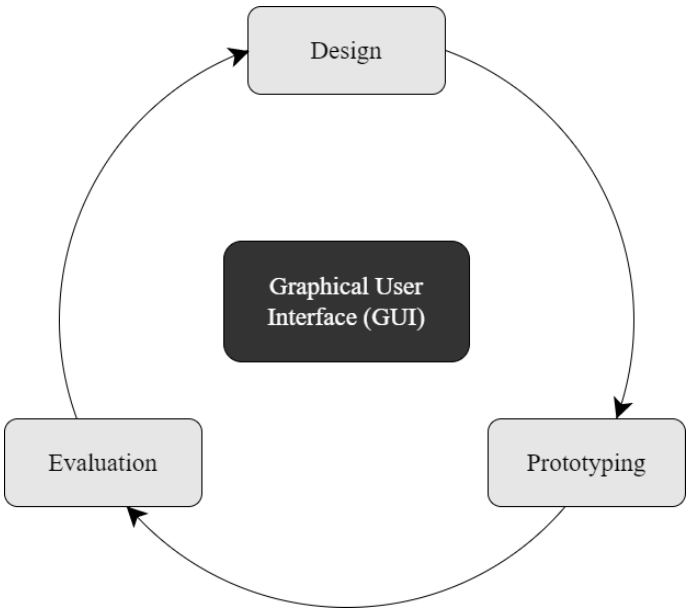


Figure 4.3: Iterative user interface design, prototyping and evaluation process.

In the design phase, laying the foundation for a pleasing and simple user interface is essential, through the establishment of the product's features and benefits, and the creation of mockups that endeavour to fulfill the needs of the users. Prototyping stage indicates choosing useful libraries and implementing some basic aspects of the design. Then, the prototype is tested and evaluated by the users, who provide information to the developer about their opinions, satisfaction, questions, and issues.

4.1.3 Final Stage

The final version is the result of successive increments to the developed work and may need time spent on refactoring to improve the software, as regular change tends to corrupt its structure.

Additionally, a final test with engineers and physicians is administered to help evaluate the application's usability, ensuring that it meets the needs and expectations of its intended users. The

favored method to collect data was surveys, followed by a casual interview for additional feedback, when needed. These surveys can be observed in Appendix A, on Figure A.1 and A.2.

After the product has passed all back-end testing phases and received feedback from users, it is ready to be used in a real-world environment. This is also the stage where the *Conclusion* is drawn, synthesizing all the important obtained results.

4.2 Description of Technologies and Libraries

This section provides an overview of the various technologies and libraries employed in the development of the application. It aims to shed light on the programming languages, frameworks, libraries, and tools that were utilized to build the application's foundation. Additionally, it explores the rationale behind the selection of each technology and discusses the significant contributions these technologies make towards the development and functionality of the application.

The programming language chosen for this application is Python, a widely adopted and versatile language known for its simplicity, readability, and extensive support for various programming paradigms. Python offers a rich ecosystem of libraries and frameworks that greatly enhance development efficiency and enable the implementation of complex functionalities. The main libraries adopted were PyDicom to manage DICOM files and PyQt5 to design the interface and enable most of the functionalities. Additionally, the JSON file type was used as an external storage of information relevant for both manual and semantic annotation.

4.2.1 PyDicom

The PyDicom library plays a crucial role in the application's development by providing extensive support for managing and manipulating DICOM files. DICOM files serve as a standard format for medical image storage and exchange, making them integral to the field of medical image analysis.

The decision to incorporate PyDicom into the application stems from its ability to effectively handle DICOM files, facilitating essential operations such as metadata extraction, pixel data manipulation, and other DICOM-related tasks. By leveraging PyDicom, the application ensures efficient and accurate processing of medical image data, enabling seamless integration into its functionality.

Specific features within the application make use of the PyDicom library. For instance, the library facilitates the opening of medical images and associated masks, allowing for the retrieval of relevant tags and metadata. PyDicom simplifies the conversion of pixel data into a suitable image format for display, enabling the application to visualize the medical images accurately. Additionally, the library leverages the "bodyPartExamined" attribute to associate the currently displayed dataset with the corresponding tags stored in JSON files. This association enables the application to retrieve and display relevant mask information, including names and associated colors. Moreover, the PyDicom library empowers the application to save masks as DICOM files. This process involves creating an empty DICOM file from scratch and populating it with essential metadata, as well as integrating the content of the displayed mask into the pixel data. By leveraging PyDicom's

functionality, the application ensures the proper preservation and compatibility of masks in the DICOM format, enabling seamless integration with other medical imaging systems.

4.2.2 PyQt5

The PyQt5 library, a Python binding for the widely-used Qt framework, serves as another pivotal component within the application. By harnessing PyQt5's comprehensive collection of classes and widgets, developers can create robust and feature-rich graphical user interfaces (GUIs). This library offers a plethora of customizable UI components and supports diverse interaction patterns, providing developers with the tools to design interfaces that are both intuitive and visually appealing.

The selection of PyQt5 was driven by its association with highly regarded applications such as Qt Creator and Qt Designer (Figure 4.4), which exemplify a "what you see is what you get" approach. These applications allow developers to easily design interfaces through a drag-and-drop interface, resulting in UI designs that can be seamlessly converted to Python code by executing the "pyuic5" script within the command prompt.

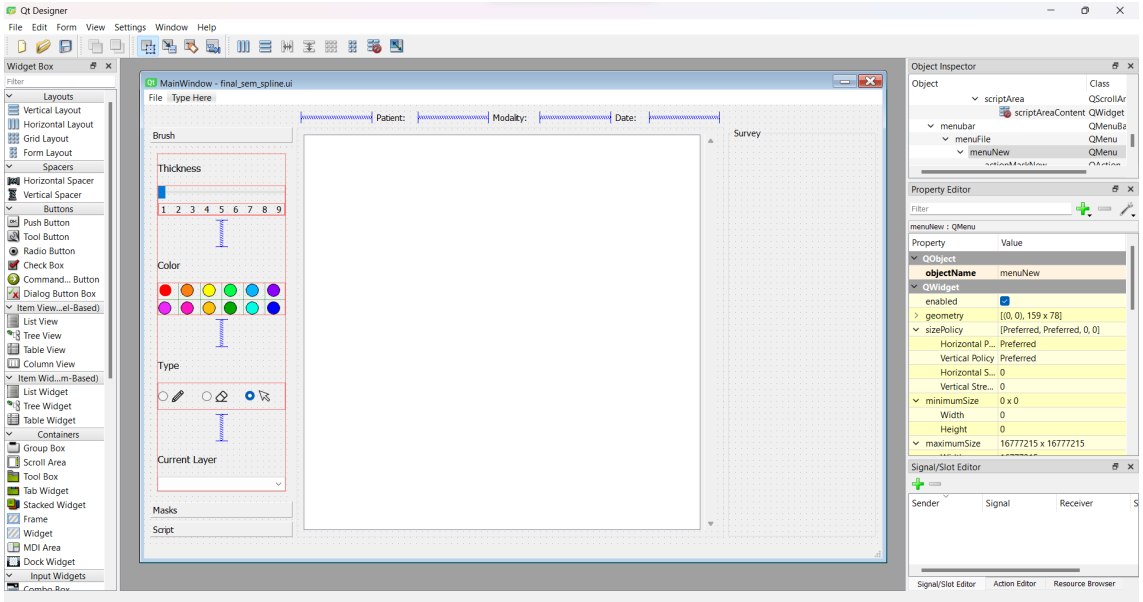


Figure 4.4: Qt Designer interface used to create the interface.

In the application, each GUI element corresponds to a Qt element or inherits from one. This tight integration with the Qt framework is particularly valuable for the brush tool and its associated drawing and erasing functionalities, which rely on the rich assortment of functions provided by Qt elements. Notably, the LayerItem class, inherited from QGraphicsRectItem, leverages the extensive capabilities of these Qt elements to enable efficient drawing and erasing operations.

4.2.3 JSON

JSON, or JavaScript Object Notation, is a widely adopted file format utilized in web applications for efficient storage and retrieval of information. It serves as a lightweight and flexible means of representing data structures in a human-readable format.

The decision to incorporate JSON files into the developed application was based on several factors. Firstly, JSON provides a straightforward approach for organizing and storing information in the form of key-value pairs, ensuring external storage of data while maintaining accessibility. This separation of data from the application's logic and control promotes modularity and facilitates data management.

Within the application, JSON files are employed for two primary purposes: tags storage and survey management. For tags, JSON files are utilized to associate names and colors, establishing meaningful connections between visual elements and their corresponding metadata, as illustrated in Figure 4.5a. Each JSON file is named after the specific body part being displayed, allowing for efficient retrieval of tag information. In the context of surveys, JSON files are employed to store both questions and answers, as depicted in Figure 4.5b. This enables the application to easily read and write survey-related data, facilitating seamless integration with survey functionality. By leveraging the inherent flexibility of JSON, the application achieves efficient and structured management of survey data, supporting dynamic interactions with the user.

```
{
  "options": [
    "pneumothorax",
    "cysts/bullae",
    "emphysema",
    "abscess",
    "tuberculosis",
    "former operation"
  ],
  "colors": [
    "red",
    "orange",
    "yellow",
    "lightGreen",
    "lightBlue",
    "purple"
  ]
}
```

(a) JSON to store name tags and associated colors for masks.

```
{
  "label1": {
    "type": "label",
    "text": "Axial Location"
  },
  "radiobutton1": {
    "type": "radiobutton",
    "text": "Central",
    "checked": false,
    "group": "group1"
  },
  "radiobutton2": {
    "type": "radiobutton",
    "text": "Peripheral",
    "checked": false,
    "group": "group1"
  }
}
```

(b) JSON for side notes storage, with both questions and answers.

Figure 4.5: Example of JSON files incorporated in the application.

4.3 Summary of Methodology

A well constructed workplan is essential in software projects, as it helps manage every phase of the project, leading to a successful outcome. The proposed workplan for this project follows an incremental and iterative development life cycle models and is divided into three stages: the initial stage, intermediate stage, and final stage.

In the initial stage the problem is identified, relevant information is gathered, and literature is reviewed and the preliminary work is carried out to complement data analysis. The intermediate stage involves concurrent activities such as specification, design, implementation, and validation of both the back-end and front-end systems. The final stage focuses on administering a final test with potential users to evaluate the application's usability. Surveys are the preferred method to assemble data. Once both developers and users are fully satisfied, the product is delivered. The conclusion is also drawn in this stage, synthesizing the important results obtained.

The specific technologies and libraries utilized in the application's development are then presented. The PyDicom library is highlighted for its extensive support in managing and manipulating DICOM files. The PyQt5 library, which provides Python bindings for the Qt framework, is emphasized for creating robust and feature-rich graphical user interfaces. Additionally, the use of JSON is mentioned, highlighting its role in organizing and storing information as key-value pairs, promoting modularity and facilitating data management.

Chapter 5

Design and Development

In this chapter, an in-depth exploration of the system architecture and workflows is introduced. Section 5.1 offers a comprehensive overview of the overall structure and organization of the application, highlighting the employed architectural patterns and design principles. Section 5.2 focuses on identifying and elucidating the main components of the application. Each component is discussed in terms of its purpose and functionality within the system, which provides insights into the core elements that contribute to the application's behavior. Finally, Section 5.3 delves into the interface design and user workflows

5.1 System Architecture

The application follows a well-structured architecture consisting of multiple hierarchical levels. At the file level, the organization of folders facilitates intuitive categorization and management of files. At the code level, classes are meticulously developed, interconnected, and segregated into separate files, adhering to the Model-View-Controller architectural pattern.

5.1.1 Architectural Patterns and Design Principals

The design pattern that has guided the development of the application is the Model-View-Controller (MVC). The MVC is highly regarded in computer science as one of the most significant patterns. Unlike many other patterns that tackle specific issues, MVC defines the architecture for object systems, applicable to both isolated subsystems and complete applications. It offers flexibility in implementation, allowing for various interpretations and adaptations.

MVC is characterized by its three core components: the model, the view, and the controller. The model represents the data and business logic, the view handles the user interface presentation, and the controller acts as the intermediary between the model and view, managing user interactions, as depicted in Figure 5.1.

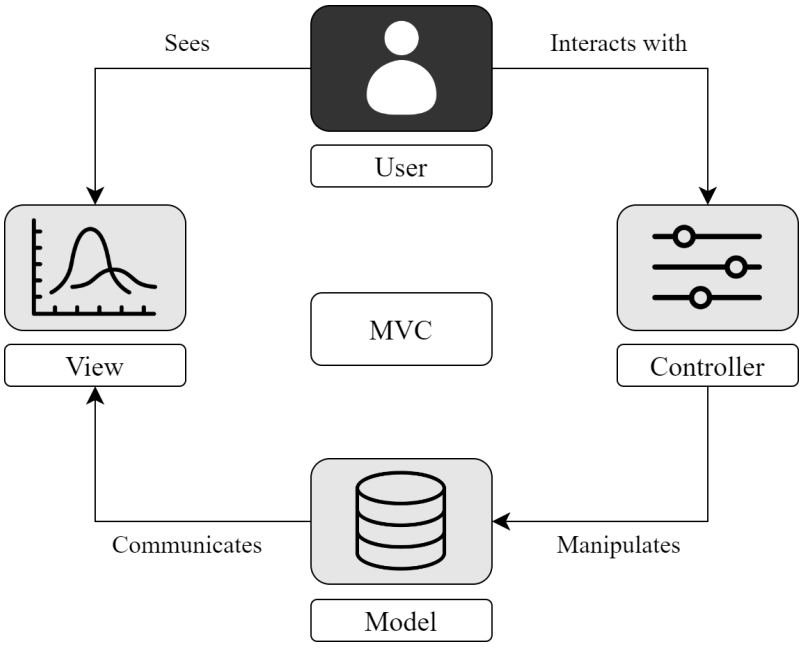


Figure 5.1: Model-View-Controller architectural pattern.

This design pattern promotes the separation of concerns (SoC), a fundamental design principle. SoC advocates dividing a system's functionalities into distinct modules, each addressing a specific aspect. By segregating data management, user interface, and user interactions, MVC enhances modularity, maintainability, and scalability.

5.1.2 Structure and Organization

UML diagrams have become increasingly important in software engineering processes, particularly in software systems. Among these diagrams, UML class diagrams have emerged as a key tool for visualizing class hierarchies, associations, aggregations, and compositions [43]. Such diagrams provide a comprehensive representation of classes, including their attributes, operations, and methods, encapsulated within a single graphical depiction. In this context, a Class Structure Diagram (or Class Hierarchy Diagram) serves as a specific type of diagram closely resembling the UML class diagram. Its primary objective is to convey the static structure of classes and their associations, while deliberately excluding the details of attributes, operations, and methods. The Class Structure Diagram of the developed application with all the classes and relationships between them is illustrated in Figure 5.2.

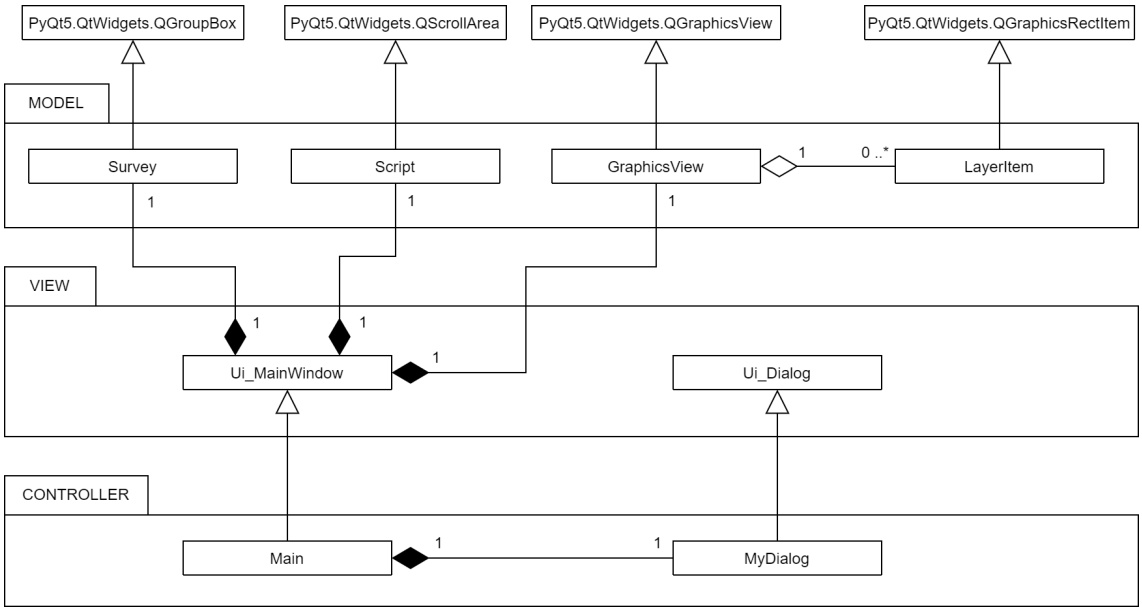


Figure 5.2: Class Structure Diagram of the application developed.

Upon analysis of the diagram depicted in Figure 5.2, it is evident that the system comprises a total of eight classes, which are organized into three separate files. The *model.py* (*Model* file) contains the classes *Survey*, *Script*, *GraphicsView*, and *LayerItem*. The *view.py* (*View* file) includes the classes *Ui_MainWindow* and *Ui_Dialog*, while the *controller.py* (*Controller* file) consists of the classes *Main* and *MyDialog*.

Furthermore, the diagram provides insights into the relationships among these classes. Some additional classes were imported from *PyQt5.QtWidgets* (*QGroupBox*, *QScrollArea*, *QGraphicsView* and *QGraphicsRectItem*) to help build the classes in the *Model* file. The *Survey* class inherits from *QGroupBox*, the *Script* class inherits from *QScrollArea*, the *GraphicsView* class inherits from *QGraphicsView*, and the *LayerItem* class inherits from *QGraphicsRectItem*.

The *Main* class serves as the child class in an inheritance relationship with *Ui_MainWindow*, while *MyDialog* inherits from *Ui_Dialog*. In addition, all classes in the *Model* file, except for *LayerItem*, exhibit a composition relationship with *Ui_MainWindow*. This means that *Survey*, *Script*, and *GraphicsView* are integrated as attributes within the *Ui_MainWindow* class, forming a whole-part relationship. Similarly, *MyDialog* is a component of the *Main* class, representing another composition relationship. It is important to note that all composition relationships depicted in the diagram are one-to-one, implying that each class is associated with a single instance of the related class.

Lastly, the *GraphicsView* class exhibits an aggregation relationship with *LayerItem*, indicating that it can contain zero or more instances of *LayerItem*.

In the application's file structure, illustrated in Figure 5.3, we can identify three primary files enclosed within the *MVC* folder: the *model.py*, the *view.py*, and the *controller.py*. Additionally, two essential folders, namely the *icons* folder and the *tags* folder, contribute to the application's functionality by providing PNG images for buttons and labels for mask categorization based on

the DICOM dataset's attribute *bodyPartExamined*, respectively. It is crucial to note that these folders must remain intact as their contents are integral to the application's proper execution. Furthermore, the file structure encompasses three additional folders that serve as supplementary components within the application. Although not essential for the application's functionality, they play a significant role in providing example or test data. The contents of these folders serve as illustrative datasets that users can choose to open and explore, allowing for experimentation or validation of the application's capabilities. Specifically, within the *case* folder, a collection of DICOM slices representing a dataset can be found. The *script* folder contains a Python executable file and a serialized PyTorch state dictionary file (PTH) that stores training data. Lastly, the *surveys* folder includes examples of surveys represented in the JSON format, providing users with a glimpse into the structure and content of such survey files.

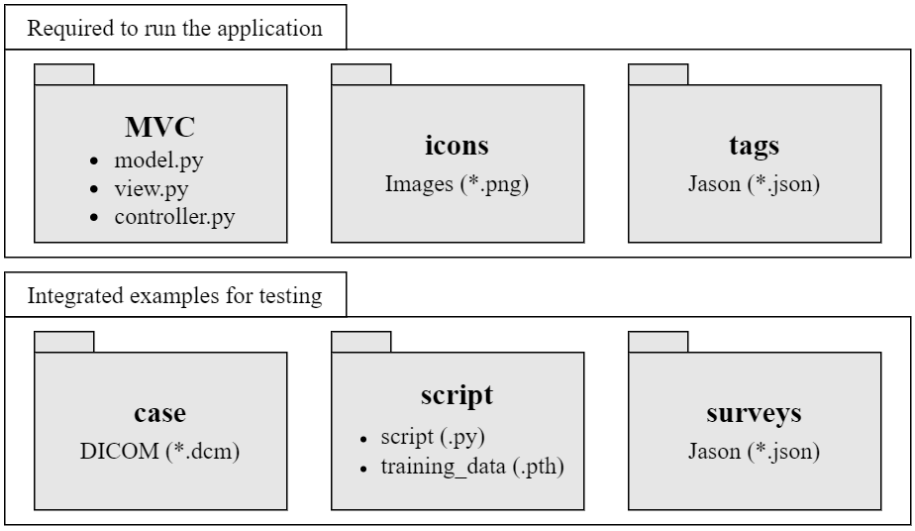


Figure 5.3: Folder organization of the application developed.

By understanding the organization and purpose of these folders and files, users can effectively navigate and utilize the application's resources, both in terms of required elements and available example data.

5.2 Components and Functionalities

The successful achievement of the objective, which entails visualizing medical images and enabling the user to carry out automated and manual annotations, necessitates the collective operation of individual components within the application. It is imperative to thoroughly examine the fundamental constituents and explain their respective roles to establish a comprehensive understanding of their contributions towards attaining the desired outcome.

The key components and their main attributes and methods are illustrated in the UML Class Diagram (Figure 5.4).

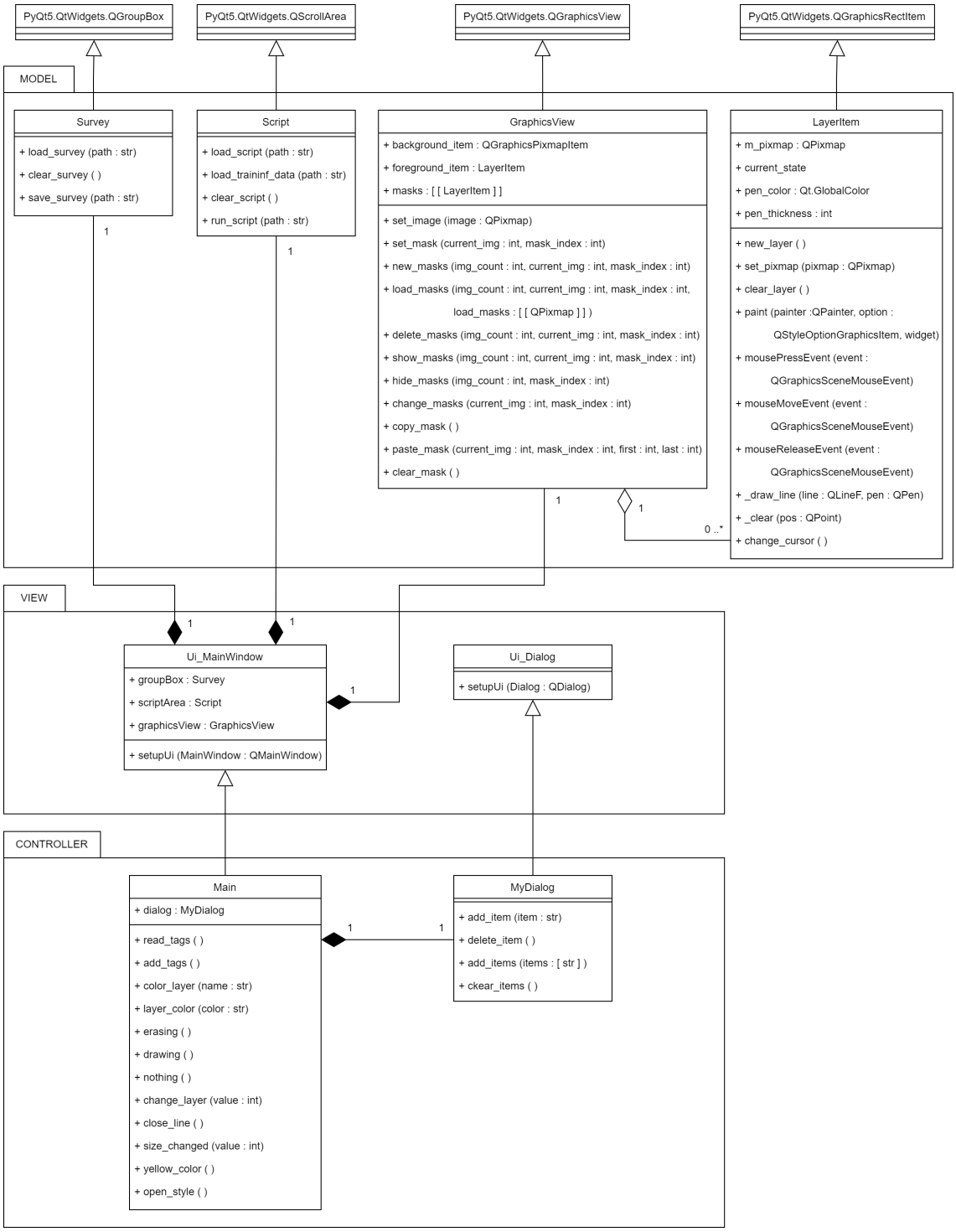


Figure 5.4: UML Class Diagram with some of the most important attributes and methods of each class that exists in the developed application.

5.2.1 Model

The model component assumes a pivotal role in the application's architecture, encapsulating the majority of its logic. Tasked with governing the behavior and functionalities of the application, it

comprises four distinct classes, with primary emphasis placed on the *GraphicsView* and *LayerItem* classes in terms of implementation. These classes orchestrate the manipulation of medical images, including the handling of associated masks and enabling manual annotation through a paint brush tool. The script functionality controls automated annotation through the execution of external scripts, while the survey component facilitates the manipulation of semantic annotations in the form of simple JSON files.

5.2.1.1 *GraphicsView* Class

The *GraphicsView* class is responsible for displaying both images and masks. It encompasses key attributes that contribute to its functionality, such as the *background_item* and *foreground_item*. The *background_item*, represented by *QGraphicsPixmapItem*, serves as the image being displayed. On the other hand, the *foreground_item*, an instance of the *LayerItem* class, handles the foreground layer or mask and manages the drawing operations. Another fundamental component of the *GraphicsView* class is the masks attribute, which is a list of lists. Each element within this structure represents a *LayerItem* and corresponds to a specific mask used in the application. These masks are stored and organized within this data structure for easy access and manipulation.

The class provides a comprehensive set of functions for manipulating the medical images and masks. Firstly, the *set_image* and *set_mask* functions are responsible for configuring the display to show the desired slice of an image and its associated masks, respectively. Additionally, the *new_masks* and *load_masks* functions enable the creation and loading of masks, accommodating different scenarios and requirements. To facilitate the user's interaction with masks, the *change_masks* function allows the selection of a specific mask to be used for drawing purposes. It alters the *foreground_item* attribute to correspond to the chosen mask, ensuring that subsequent drawing operations are performed on the desired mask.

To address further needs in manipulating masks, additional functions were introduced. The *hide_masks* and *show_masks* functions enable the toggling of visibility for a particular set of masks, effectively controlling their display on the screen. The *copy_mask* and *paste_mask* functions facilitate the duplication and pasting of mask content, allowing the user to copy a mask and apply it to a range of masks as specified. The *clear_mask* function serves the purpose of erasing the content of a specific mask, providing a straightforward method to reset a mask if needed. Finally, the *delete_masks* function removes a set of masks from the masks list, effectively deleting them from the application.

Through these functions and attributes, the *GraphicsView* class empowers the application to display images and masks seamlessly while enabling comprehensive manipulation of the masks for various tasks. The component displayed in the center of the interface as a large white box.

5.2.1.2 *LayerItem* Class

The primary attribute, *m_pixmap*, is a *QPixmap* object that stores the content of the layer. It holds the graphical information and can be manipulated during drawing and erasing operations.

Another key attribute is *current_state*, which represents the current state of the mask. It can take one of three values: *DrawState*, *EraseState*, or *MouseState* (representing a non-functional state). Additionally, *pen_color* and *pen_thickness* attributes determine the color and thickness of the pen used for drawing or erasing.

The class provides essential functions for creating empty masks (*new_layer*) when the user wants to add new masks, and assigning pixmaps to existing masks (*set_pixmap*) when loading pre-existing files. These functions ensure the proper initialization and setup of the layer.

Furthermore, the class includes functions that handle drawing and erasing operations. The paint function overrides the paint method in the *QGraphicsRectItem* to render the layer's pixmap on the scene. Mouse events such as *mousePressEvent*, *mouseMoveEvent*, and *mouseReleaseEvent* are responsible for capturing and responding to user interactions with the layer. These functions take into account the position of the mouse and perform appropriate actions for drawing and erasing on the layer. The
textittextunderscore draw_line function draws a line on the layer's pixmap using the specified line and pen, while *_clear* function clears a rectangular region on the layer's pixmap centered around the provided position. To enhance the user experience, the *change_cursor* function modifies the cursor's appearance based on the current state of the layer. It adjusts the cursor to indicate the tool in use. Lastly, the *clear_layer* function clears the content of the layer by filling it with transparency.

In summary, the *LayerItem* class serves as a fundamental component within the *GraphicsView* widget, functioning as a layer or mask. It provides attributes for storing the layer's content and managing its state, along with functions for creating, manipulating, and clearing the layer.

5.2.1.3 *Script* Class

The *Script* class facilitates automatic annotation by generating masks automatically. It requires the loading of an external script, alongside the associated training data, into the application for subsequent execution.

The *load_script* function accepts a file path as an argument and loads the script from that specified path. It performs the tasks of opening the file, reading its contents, and presenting them to the user for visualization purposes. On the other hand, the *load_training_data* method also takes a file path as an argument and assigns the *training_data* attribute of the class to the provided path. The primary function of this method is to store the file path representing the training data associated with the script. Subsequently, the *run_script* function allows the execution of the loaded external script. This method dynamically imports and executes the script module. In the case where the script module includes a function named 'run', this function is invoked with the path to the folder containing DICOM images and the training data as parameters. The method returns the outcome of the script execution in the form of an array.

Lastly, the *clear_script* function is responsible for the removal of the loaded script and training data. It eliminates all labels from the layout, sets the script and *training_data* attributes to None, thereby effectively resetting the state of the class.

5.2.1.4 *Survey* Class

The *Survey* class serves as a distinct component within the application, separate from the graphics view, allowing users to record notes about the dataset.

The *load_survey* function retrieves survey data from a specified JSON file path and dynamically generates appropriate widgets based on the data. These widgets are then added to the interface for display. The *save_survey* function is responsible for persisting the survey data. It updates the checked status of check boxes and radio buttons based on the current state of the corresponding widgets within the layout. The modified survey data is then serialized and written to a JSON file specified by the provided path. Lastly, the *clear_survey* method facilitates the removal of all widgets from the layout, effectively resetting the survey. This process includes clearing group information and resetting the label attribute, preparing the *Survey* class for a fresh state.

5.2.2 View

The *Ui_MainWindow* class is responsible for defining the structural layout and functional characteristics of the diverse user interface (UI) components within the primary application window. The *Ui_Dialog* is a useful complementary window to get user input.

5.2.2.1 *Ui_MainWindow* Class

The Ui_MainWindow class is responsible for instantiating and configuring the UI components that constitute the primary window of the application.

The primary UI components encompass the *graphicsView*, an instance of the *GraphicsView* class, which facilitates the visual presentation and manual annotation of medical images. This component offers features for user engagement, such as drawing and erasing annotations. The *groupBox*, derived from the *Survey* class, acts as a container for users to take notes related to the displayed image. Furthermore, the *toolBox* attribute denotes a *QToolBox* widget, encompassing three distinct pages: *Brush*, *Mask*, and *Script*. Each page caters to a specific annotation tool or functionality. The brush page enables manual annotation employing a brush tool, while the mask page streamlines manipulations concerning masks. The script page encompasses the *scriptArea* attribute, an instance of the *Script* class, likely enabling script-oriented annotation capabilities.

The *Ui_MainWindow* class additionally configures supplementary UI elements, exemplified by *QLabel* instances like patient name, modality and date. These labels serve the purpose of conveying specific details linked to the exhibited medical image, with appropriate styles and properties.

Its implementation enables the effective display of medical images, facilitates the capture of user annotations, and provides a comprehensive array of tools for both manual and automatic annotation tasks.

5.2.2.2 *Ui_Dialog* Class

The *Ui_Dialog* class is responsible for constructing and configuring the UI components for a custom dialog window.

The main UI component of this class is the *dialogCombo*, a *QComboBox* widget that allows the user to select the tag for the mask from a list of options. It also includes standard buttons for accepting or canceling the dialog.

5.2.3 Controller

In the software architecture, the controller component includes the *Main* class, which serves as the main executable. The *Main* class facilitates the manipulation of the underlying model and establishes appropriate connections between the view and the relevant functions. Additionally, within the controller, there is the *MyDialog* class, which defines a specialized dialog window.

5.2.3.1 *Main* Class

The *Main* class assumes the central control role within the application, inheriting the attributes of the *Ui_MainWindow* class that define the user interface elements such as buttons and widgets. By doing so, it establishes a linkage between user interactions and specific functions within the model. Aside from the attributes inherited from the interface, the most relevant is the *dialog*, which represents an object of the *MyDialog* class

Regarding the functions, many of them serve as connectors between user actions and corresponding methods within the model. Noteworthy functions include *read_tags* and *add_tags*, which analyze the *bodyPartExaminated* attribute of the imported DICOM file. They extract associated tags from a JSON file within the application's files and display them in an interface's widget and in the *QComboBox* located in the *dialog* instance. The *color_layer* and *layer_color* functions share a similar purpose, where the chosen color affects the name of the mask, and vice versa. Other functions essential for drawing purposes include erasing, drawing, and nothing, which toggle the state of *LayerItem* items.

Additionally, there are functions addressing specific tools or requirements. For example, *close_line* automatically closes a line when drawing, *change_layer* modifies the layer being drawn upon, and so forth. Simple functions exist for altering button icons, pen color, and pen size during drawing and erasing operations.

In summary, each operation available within the application is accompanied by a corresponding function that controls and connects the view to the model.

5.2.3.2 *MyDialog* Class

The purpose of the *MyDialog* class is to define a custom dialog window that extends the functionality of the *QDialog* class from the PyQt library. It provides methods to manipulate a *QComboBox*x widget (*dialogCombo*) that is defined in the associated user interface (*UI_Dialog*).

The *add_item* method adds a single item to the *dialogCombo*, while the *add_items* method adds multiple items. The *delete_item* method removes the currently selected item from the *dialogCombo*. The *clear_items* method removes all items from the *dialogCombo*.

Overall, this class provides a convenient interface to interact with the *QComboBox* widget in the dialog window, allowing items to be added, deleted, or cleared. It encapsulates the logic and provides a higher-level abstraction for working with the dialog's UI components.

5.3 User Interactions

User interface design assumes a vital role by directly shaping the user's experience and interaction with the system. Its objective lies in constructing an intuitive and visually appealing environment that fosters smooth and efficient user interactions. To accomplish this, specific user interaction patterns have been implemented, tailored to enhance usability and streamline user tasks within targeted workflows.

5.3.1 User Interface Design

The application's user interface encompasses various groups of controls and significant areas, which are depicted in Figure 5.5.

The primary and largest area is denoted as the *Display and Annotation Area* or *GraphicsView* (in green). Within this area, medical images are presented individually, allowing users to perform manual annotations using mouse interactions. Adjacent to the view, positioned on the right side, is the *Navigation Slider* (in pink), which imparts a sense of depth or spatial dimension to the displayed two-dimensional images. The *Tool Box* (in red) provides options and tools related to manual annotation. The *Brush Page* within the *Tool Box* allows users to configure brush settings such as thickness and current state (draw, erase, or none) and tools realted to the change of layers. Additional pages within the *Tool Box* encompass mask manipulation tools and an automatic annotation plugin.The *Menu Bar* (in yellow), offers a comprehensive range of functionalities, including importing DICOM sets, loading or creating new masks, importing surveys, and saving both masks and surveys. The *Dataset Information* section (in purple), consists of labels that convey specific details associated with the displayed medical image. Lastly, the *Survey Area* (in light blue), serves as a designated space for users to take notes. It comprises a group of labels and radio buttons, imported from a JSON file. Furthermore, the *Tool Box* may present two additional pages. The Masks Page (Figure 5.6) enables users to manipulate existing masks, providing options to change the layer, set visibility, enable automatic line closure, copy and paste, clear, and delete masks. The *Script Page* (Figure 5.7) facilitates the selection and opening of external scripts, script execution, and a clean-up feature for the display area. A further explanation of the interface components and how to use them to complete tasks is presented on Appendix B.

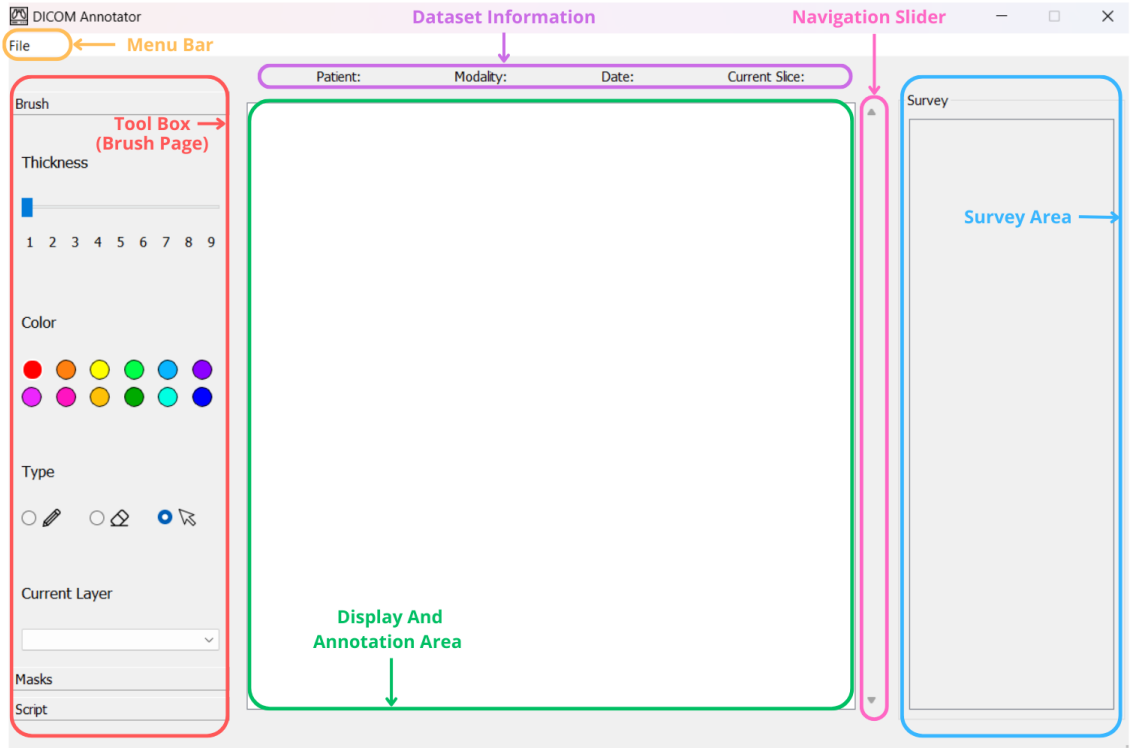


Figure 5.5: User interface of the developed application with relevant controls highlighted.

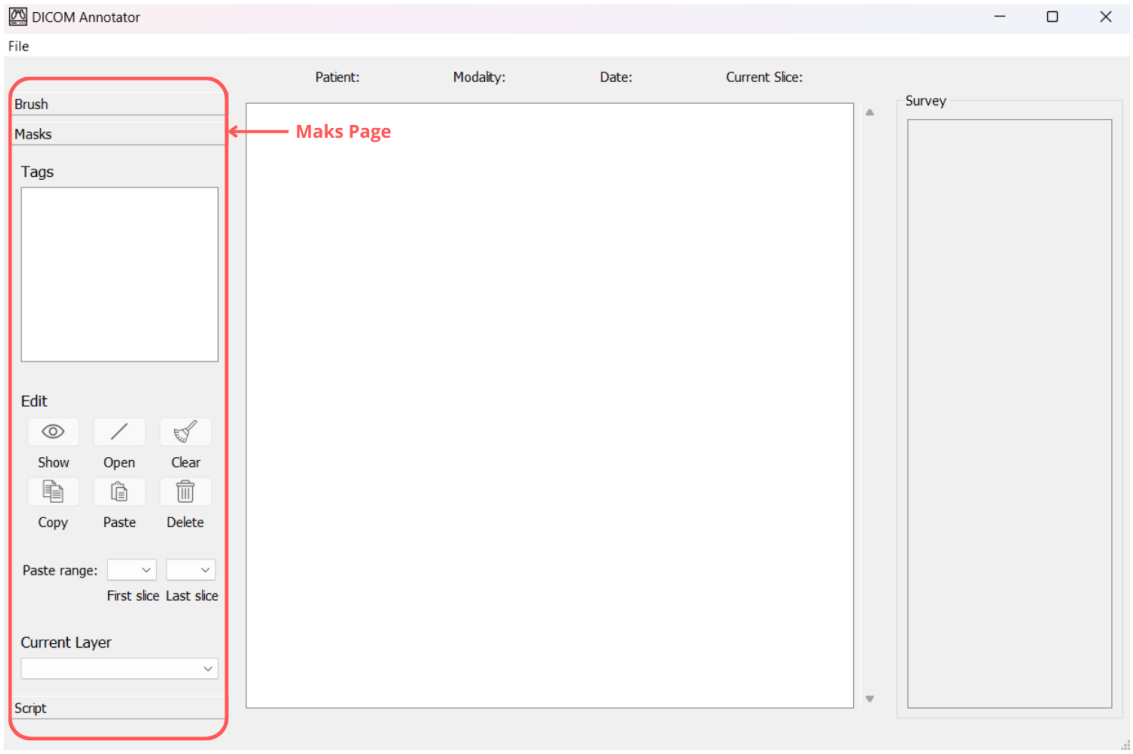


Figure 5.6: User interface of the developed application with *Masks Page* from *Tool Box* pointed out.

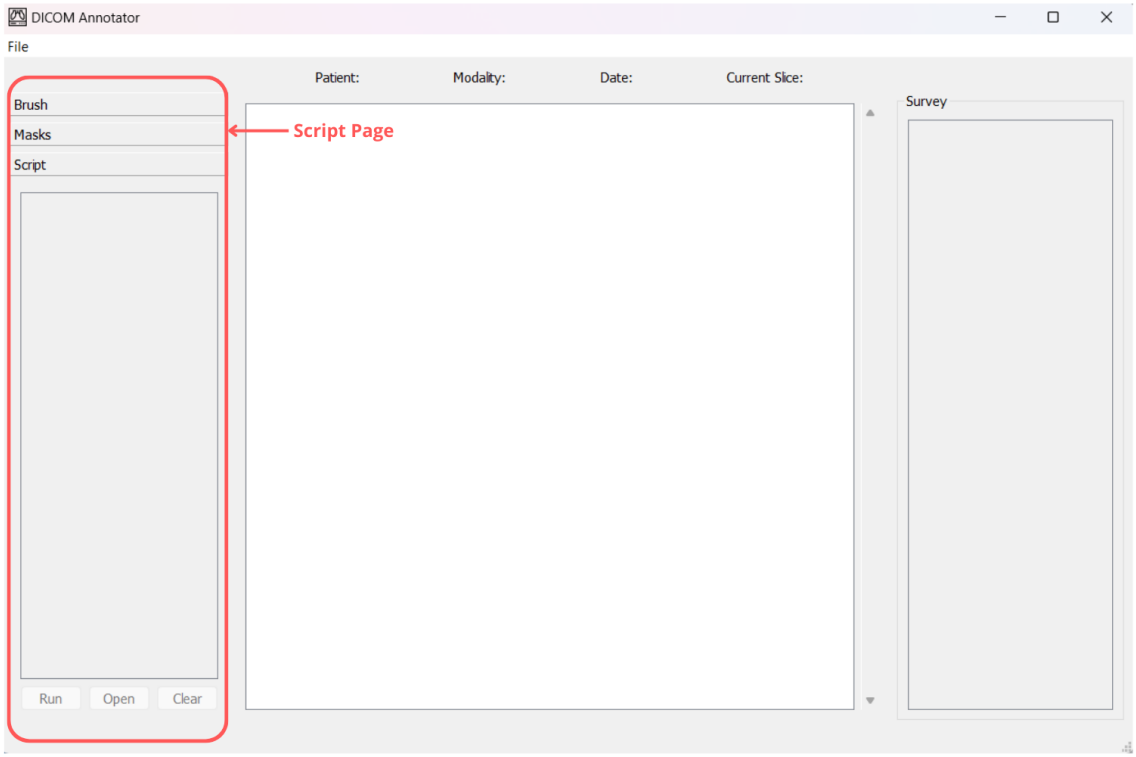


Figure 5.7: User interface of the developed application with *Script Page* from *Tool Box* pointed out.

5.3.2 User Interaction Patterns

Within the application, well-defined user interaction patterns have been applied to guarantee a smooth and productive user experience (UX). These patterns allow users to effectively accomplish their tasks by following very well structured sequences. The tasks may include navigate the application, understand its purpose or achieve objectives.

An important user interaction pattern integrated in the application and easily spotted by having a separated class focused on it's implementation is the **Dialog Window** pattern. Dialog windows provide a dedicated space for specific tasks, allowing users to interact with a conducted process without interrupting the main application. In the developed application, a custom dialog window that incorporates a *QComboBox* widget has been designed to select layer tags, as depicted in Figure 5.8. This pattern simplifies the process and makes the action of choosing a name mandatory to the flow of mask creation or loading.

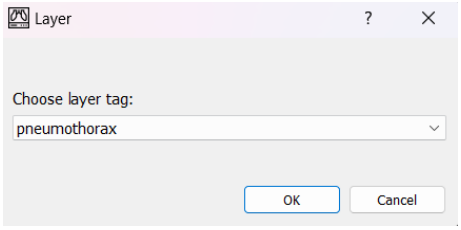
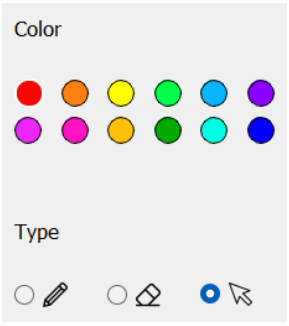


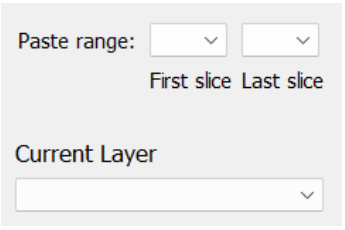
Figure 5.8: Dialog window that includes a *QComboBox* widget designed to select a layer tag from the available options.

Another prominent employed pattern is the **Selection and Manipulation**, that enables users to interact with the medical images displayed in a specific area, the *GraphicsView* component, refereed previously as *Display and Annotation Area*. In this component, users can select precise coordinates to apply annotations and perform various manipulations to its content. This pattern empowers users to have control over the visual representation and analysis of the masks that hover the medical images, enhancing their ability to interpret and interact within a personal way.

Furthermore, the application embraces the **Form and Input Validation** pattern. This user interaction pattern is broadly used in the application to ensure the data is accurate and valid. Various input widgets, such as radio buttons (Figure 5.9a), combo boxes (Figure 5.9b) and sliders (Figure 5.9c), have been strategically placed to guide users in providing personalised data, but keeping it appropriated to the specific action. This pattern helps prevent user errors and improves data quality.



(a) *Color* and *Type* radio buttons.



(b) *Paste Range* combo boxes and *Current Layer* combo box.

(c) *Thickness* slider.

Figure 5.9: Examples of the *Form and Input Validation* pattern employment in the interface.

By implementing these patterns, the application strives to provide a user-friendly interface that supports efficient and intuitive interactions. The thoughtful design and consideration of user needs contribute to an enhanced user experience and ultimately result in increased productivity and satisfaction.

5.3.3 User Workflows

User workflows refer to the sequential routes that users traverse within a digital interface to accomplish specific tasks. These workflows outline the journey undertaken by users, encompassing their interactions, system processes triggered as a result, and the corresponding outputs presented to the users. Figure 5.10 portrays the user workflows pertaining to the tool. Notably, the central focus of the tool revolves around the presentation of both medical images and the accompanying annotation overlay. By allowing users to independently manipulate the masks, the tool facilitates an improved visual experience of their contents, thereby potentially enhancing the effectiveness of the annotation task.

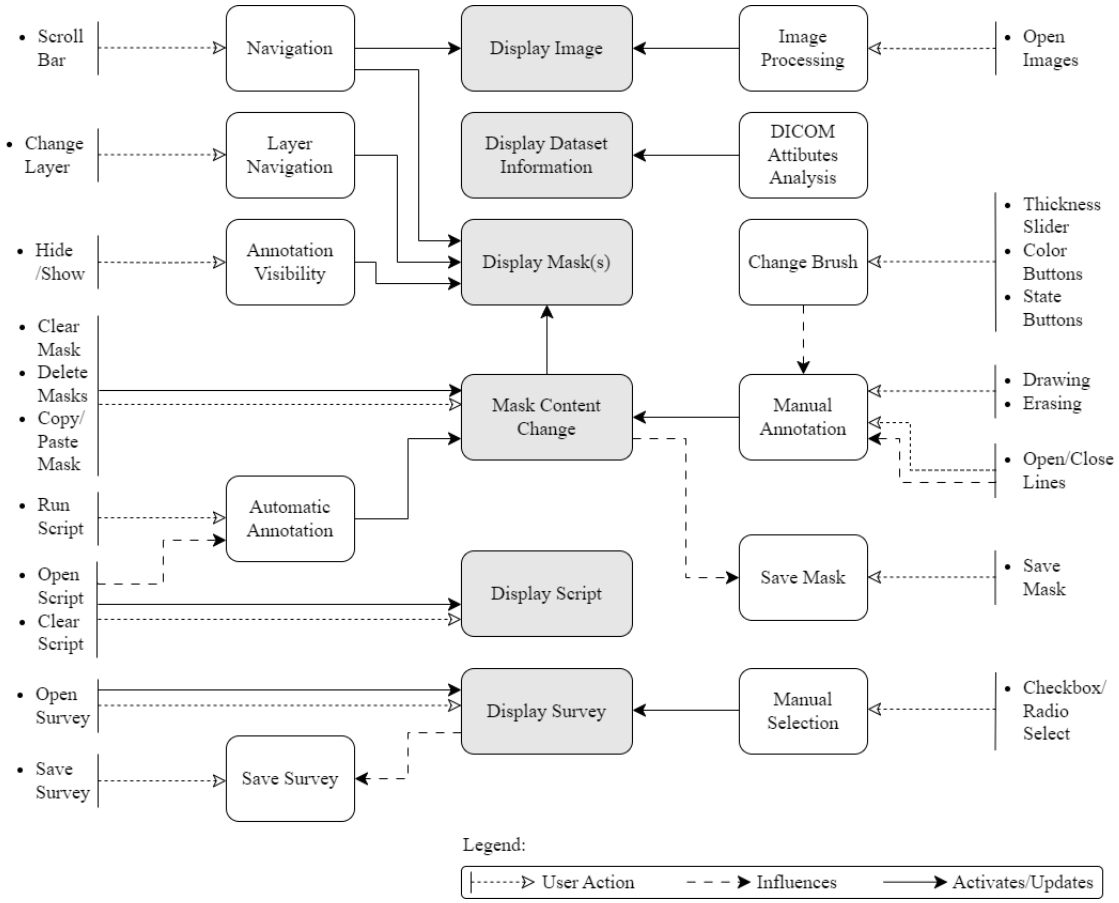


Figure 5.10: User workflows of the developed application.

5.4 Summary of Design and Development

By examining the system architecture, with a further look to the components and respective attributes and functionalities, and the user workflows, this chapter provides a complete understanding of the application. It highlights the organization of the system, the role of each components and the thoughtful decisions for user interaction contributing to a user-friendly and efficient interface.

The system architecture is examined at both file level and class level, revealing a well-organized system that adhere to the MVC design pattern. This architectural choice improves modularity, maintainability and scalability. Each component further inspected plays a significant role in allowing the user to complete the desired tasks.

Given that the application developed is a graphical user interface, the chapter emphasizes the importance of clear design and well-defined user workflows. The user interactions are carefully considered to ensure the application is user-friendly and intuitive. Various User Interaction Patterns are employed to enhance the user experience and facilitate smooth navigation and task execution.

Chapter 6

Results and Discussion

This chapter presents the results of the tests and usability evaluations performed on the application. Section 6.1 depicts some possible preparatory assignments and respective accomplished. Section 6.2 provides an overview of the tests performed and the results obtained, emphasizing any issues encountered. In Section 6.4, the usability evaluation methods used to assess the application's usability are described and the findings are presented and discussed. Section 6.3 includes a comparison between the actual results and the initial expectations for the application. In Section 6.5, the limitations and constraints faced during the development process are acknowledged.

6.1 Preliminary Work

The preliminary works plays a crucial role in the successful implementation and consummation of a project. In the development of a tool with extremely complex functionalities performing together in a very well designed interface, defining a preliminary work is essential to ensure time-effectiveness, good decision-making, and quality from the beginning to the end of the project.

An attainable introductory work is to develop a rudimentary application that is capable of opening a DICOM file, lets the user make a simple annotation with a brush and then proceed to save it as a new image, as demonstrated in Figure 6.1.

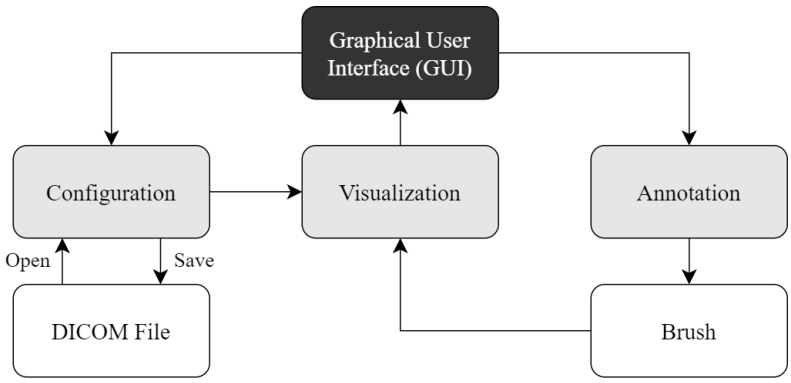


Figure 6.1: Preliminary work to create a basic visualization and annotation tool for DICOM files.

First, it is necessary to dictate the programming language to be used. Since the application to be developed is a desktop application, the following high level languages stand out: JAVA, C++, C# and Python. After taking a look into them, it is possible to choose the one that better suits the need for a clear and intuitive GUI and the demand for computer vision incorporation: Python.

Python is highly convenient for medical image processing, as it has multiple open source libraries that facilitate the DICOM file management, such as Matplotlib and Pydicom. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Pydicom clears the way for reading DICOM files into natural pythonic structures for easy manipulation. Moreover, modified datasets can be written again to DICOM format files, which is ideal as the goal is to save annotations made in the image by the user.

Afterwards, the focus should be on granting the visualization of the image to be annotated, defining the user interface construction, by defining an initial mockup, as depicted in Figure 6.2

Figure 6.2: Initial mockup of the application's GUI built on Canva.

6.1.1 Preliminary Work Results

The preliminary work plan includes handling DICOM files, beginning with opening the image. Pydicom is a crucial framework for this task in Python, as it enables easy loading of the image (using the dcmread method) and retrieval of important information such as the image data, dimensions, and context (patient name, medical image modality, and date).

Code was then written using Matplotlib to display the opened DICOM files in three different manners: a single image with a 2D grid (Figure 6.3), multiple images side by side (Figure 6.4), and multiple images in the same window that can be scrolled through to give the illusion of depth (Figure 6.5).

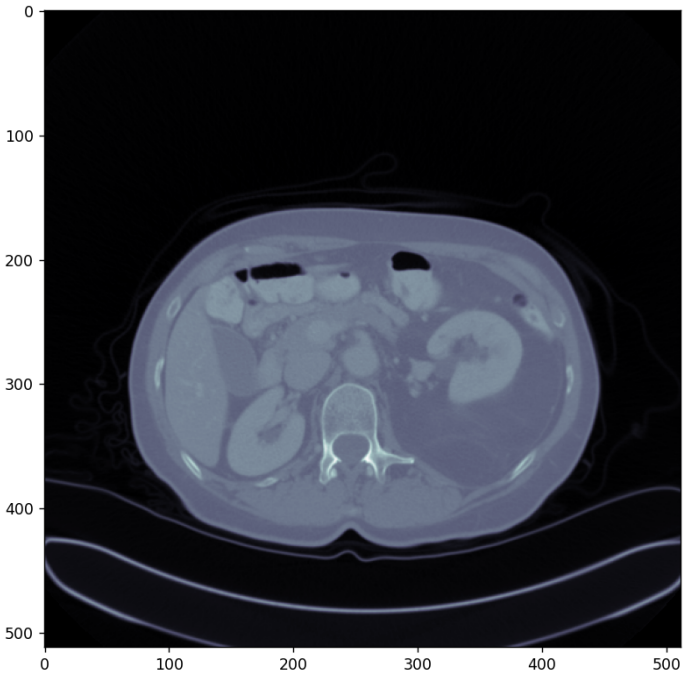


Figure 6.3: Display of a single DICOM image with a grid.

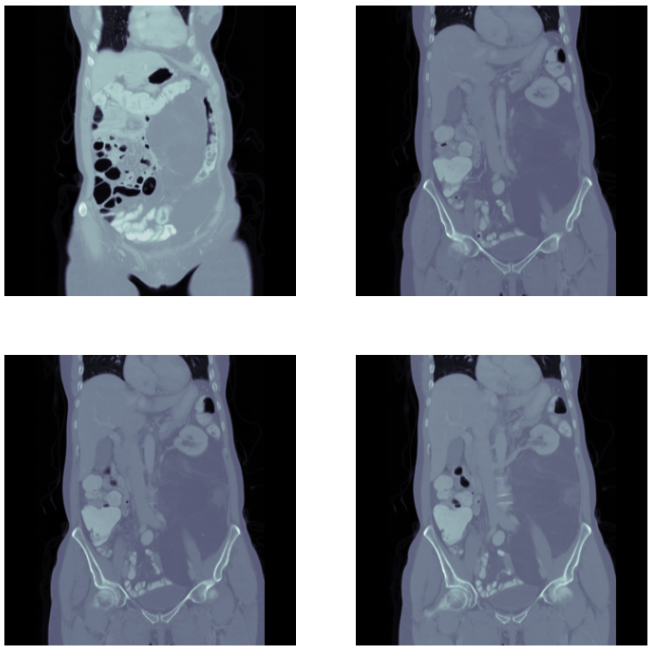
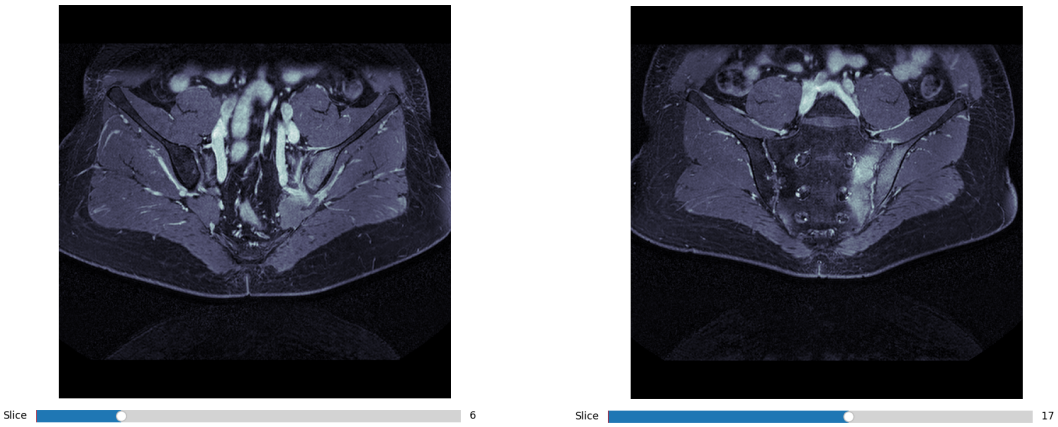


Figure 6.4: Display of multiple DICOM images side by side.



(a) Display of the slice 6.

(b) Display of the slice 17.

Figure 6.5: Display of multiple DICOM images one by one with a scroll bar.

A simple annotation of two rectangles was created, as depicted in Figure 6.6, by converting the DICOM image to a PIL image to facilitate annotation. Afterwards, the resulting annotated image is converted back to a DICOM image and saved.

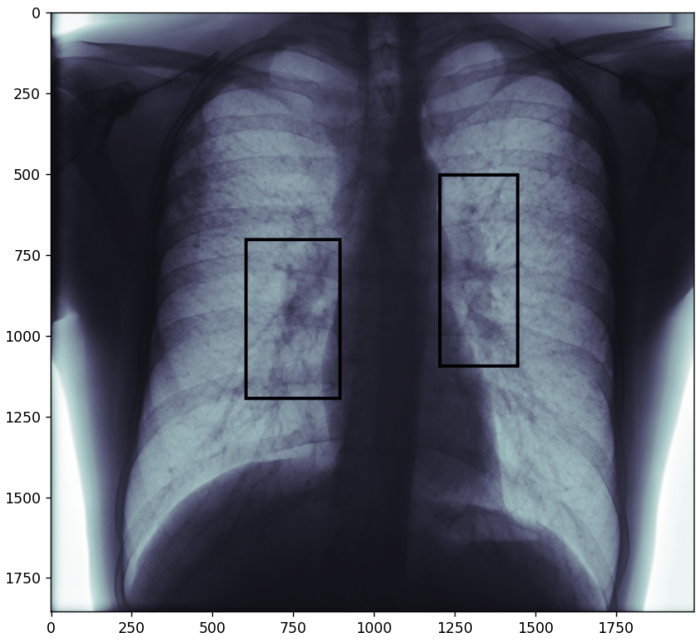


Figure 6.6: Display of annotated DICOM image with two rectangles.

6.2 Tests Conducted and Results Obtained

During the development of the application, various tests were performed to ensure the functionality and quality of the software. These tests included manual testing, where the application was

executed, and different tools and features were used to verify the effectiveness of the implemented functions.

6.2.1 Test Types

The application was tested by executing it and interacting with its GUI to simulate actual usage scenarios. During the manual testing process, in particular cases, variables were printed in the terminal to check the output of specific code lines when they were not directly visible on the interface. This allowed for supplementary verification and ensured the correctness of the application's internal operations.

6.2.2 Test Scenarios, Inputs, and Expected Outcomes

In this section, some specific test scenarios along with their corresponding inputs and expected outcomes are outlined. These tests cover various functionalities of the application and help evaluate its performance.

The tests documented include a mix of both basic and more complex tests. Basic tests helps to ensure that the fundamental features of the application behave as desired. On the other hand, incorporating more complex tests makes it possible to uncover issues and validate the performance and robustness of the application. Therefore, a combination of both primary and more intricate experiments is implemented.

Testing image loading

- Scenario – Load a DICOM dataset into the application.
- Input – Click on the *Open -> Image* button in the *Menu Bar*. Select a folder containing a DICOM dataset using the file selection dialog.
- Expected Outcome – The image should be displayed in the *GraphicsView* (or *Display and Annotation Area*), properly scaled and centered. The tag list with the names and colors of the possible masks to be added are also displayed in the *Tags TableView* present in the *Masks Page* of the *Tool Box*.
- Alternative Inputs – Select a folder containing different file types;

  – Select a folder with DICOM images of various sizes, resolutions and modalities (different datasets).

Testing mask creation

- Scenario – Create new masks for the loaded images.
- Input – Click on the *New -> Mask* button in the *Menu Bar*.

- Expected Outcome – A dialog window should be displayed to let the user select the tag and the new mask should be added to the *GraphicsView*, allowing the user to draw on it.
- Alternative Inputs – Attempt to create a mask without selecting a tag.

Testing mask loading

- Scenario – Load existing masks for the loaded images.
- Input – Click on the *Open -> Mask* button in the user interface's *Menu Bar*. Select a folder using the file selection dialog.
- Expected Outcome – A dialog window should be displayed to let the user select the mask, and the selected mask should be added to the *GraphicsView*, allowing the user to draw on it.
- Alternative Inputs – Select a folder that does not contain valid mask files.

  – Select a folder in which the number of masks does not match the number of slices.

Testing mask drawing and erasing

- Scenario – Draw on the mask using different brush sizes and then erase it using different eraser sizes.
- Input – Select a medium brush size, click on the pencil button to switch to the draw state, and draw directly on the mask with mouse press and movement. Erase the drawing by selecting the eraser button and erasing the drawing on the mask.
- Expected Outcome – The mask should be updated accordingly, reflecting the drawn strokes with the selected brush size and color and erasing the corresponding portions of the mask. The erasing action should remove the drawn strokes from the mask, leaving behind a clean area.
- Alternative Inputs – Use the minimum and maximum brush sizes available.

  – Apply the eraser on a mask with overlapping strokes.

Testing image and mask navigation

- Scenario – Navigate though different images.
- Input – Scroll on the *Navigation Slider* or on the scroll wheel of the mouse.
- Expected Outcome – The *GraphicsView* should display the image corresponding to the scroll bar's value and the associated masks, if any.
- Alternative Inputs – Rapidly scroll back and forth on the Navigation Slider.

  – Attempt to navigate to an image beyond the limits of the scroll bar.

Testing mask switching

- Scenario – Switch between different masks for the current image on display.
- Input – Select a different mask by clicking on the color button associated with the desired layer or choosing the desired layer's name on the *Current Layer* combo box with the list of masks. Erase something on the mask.
- Expected Outcome – The *GraphicsView* should bring the selected mask to the front. This can be verified by erasing something that was previously drawn on that layer.
- Alternative Inputs – Switch between masks rapidly.

Testing mask visibility management

- Scenario – Switch mask's visibility between visible and hidden.
- Input – Click on the "Visible/Hidden" button in the Masks page of the Tool Box.
- Expected Outcome – The application should toggle the visibility of the masks in the *GraphicsView*, setting the masks associated to the tag selected on the current layer combo box visible if it was previously hidden and vice versa. It should also change the button's style to match the current state of visibility (green with an eye if showing, and red with a crossed eye if hiding).
- Alternative Inputs – Hide all masks and then show them again.

  – Try to draw on an invisible mask.

Testing mask copy and paste

- Scenario – Copy a mask's content and paste it on other masks.
- Input – Select a mask and click the "Copy" button. Then, select the target masks on the range combo boxes or leave them blank to paste only on the current mask and click the "Paste" button.
- Expected Outcome – The content of the source mask should be copied and pasted onto the target masks, replicating the annotations.
- Alternative Inputs – Select the first slice higher than the last slice in the range to paste.

Testing automatically closed annotation option

- Scenario – Annotate with the option of automatically close the lines on and off.
- Input – Click the "Closed/Open" button in the Masks page of the Tool Box to choose either normal annotation or automatically closed annotation mode. Draw on the mask with each option selected.

- Expected Outcome – In normal annotation mode (open button's style on display), the user should be able to freely draw lines without them automatically closing. In automatically closed annotation mode (closed button's style on display), lines should automatically close when the user finishes drawing, connecting the first and last points. It should keep the previously selected option for the specific masks even after changing the mask or the image.

Testing mask clearing

- Scenario – Clearing a mask.
- Input – Select a mask and click the *Clear* button on the *Masks Page* of the *Tool Box*.
- Expected Outcome – The selected mask should be cleared, removing all existing annotations from the mask, whitout deleting it.

Testing mask deleting

- Scenario – Deleting a mask.
- Input – Select a mask and click the *Delete* button on the *Masks Page* of the *Tool Box*.
- Expected Outcome – The selected mask should be deleted and removed from the *GraphicsView* and mask management structure.
- Alternative Inputs – Delete all masks.

Testing automatic annotation

- Scenario – Running an external script to obtain automatic annotations.
- Input – Click on the *Open* button in the *Script Page* of the *Tool Box* to trigger a folder selection dialog and select the desired folder that contains the script file and the training data file. Click on the *Run* button on the left side of the *Open* button.
- Expected Outcome – The script should run, which may take a while, and then display the dialog window to select the favored tag for the new set of automatically annotated masks. Afterwards, the masks are displayed on the *GraphicsView* as layers.
- Alternative Inputs – Load a folder that encloses different file types.

  – Load a folder that encloses a different number of files.

  – Try to run a script that does not contain the "run" function.

Testing mask saving

- Scenario – Saving masks as DICOM files.

- Input – Click on the "Save -> Mask" button in the Menu Bar and write the name for the folder on the displayed dialog window.

- Expected Outcome – The application should save the annotated masks in the DICOM format in a folder named by the user, ensuring the masks can be accessed and loaded after.

Testing survey management

- Scenario – Open, edit and save surveys.

- Input – Click the *Open -> Survey* in the *Menu Bar* and select a JSON file from the file selection dialog. Perform some annotations by selecting check boxes or other widgets displayed in the *Survey Area*. Click on the *Save -> Survey* and choose a name for the new JSON file with the added annotations.

- Expected Outcome – The survey must be displayed in the Survey Are after being loaded, allowing the user to interact with it. Then it should be saved back to a JSON file with the given name.

6.2.3 Results Obtained and Discussion

The application must accomplish successfully the basic input tests, ensuring that the basic functionalities are working as intended. Then it is vital to experiment with edge cases to analyze whether the application is capable of handling them and expose the limitations.

Image loading: The folder with DICOM files from the same dataset within it is loaded and displayed correctly in the *GraphicsView*, as depicted in Figure 6.7. The test's response to some boundary parameters, such as a folder containing various file types or files from diverse datasets is also positive. The images are not displayed, yet the application prints some error messages (Figure 6.8) for the user to understand there is an issue without shutting the application down, which correspond to the best possible response.

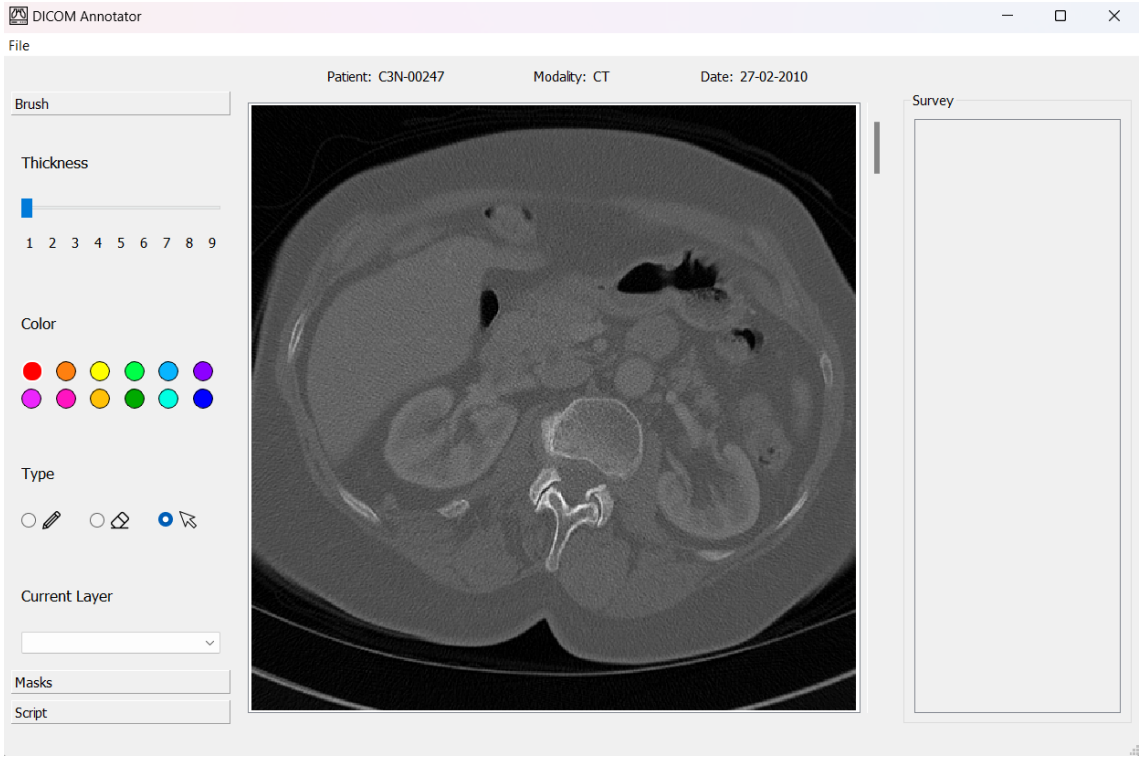


Figure 6.7: Image loading test success, by displaying the image in the *Display and Annotation Area* and the information about the dataset in the *Dataset information*.

(a) Error message when the user selects a folder with various file types.

(b) Error message when the user selects a folder with DICOM images of various datasets.

Figure 6.8: Error messages when testing image loading with invalid inputs.

Mask creation: The test verifies the correct creation of a layer that is then added to the *GraphicsView*, letting the user draw on it. It additionally tests the tag choice implemented in the dialog window, that perform as desired, by not allowing the user to create a mask without previously selecting a tag, as it is clear in Figure 6.9.

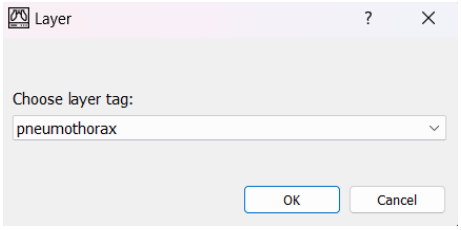
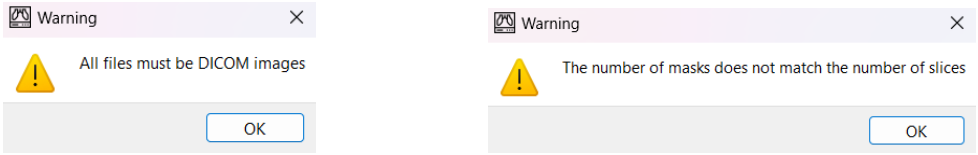


Figure 6.9: Tag selection in mask creation test.

Mask loading: The test of loading a set of masks, previously created in the developed application for the loaded DICOM dataset, has passed successfully. In this case, the files are all in the DICOM format and its number matches the number of slices of the DICOM dataset, which makes it relevant to test the opposite cases too. Similarly to what occurs in the image loading, the invalid inputs in mask loading generates error messages (Figure 6.18) that elucidate the user on the specific problem with the submitted data.



(a) Error message when the user selects a folder with various file types.

(b) Error message when the user selects a folder with a number of files different from the.

Figure 6.10: Error messages when loading invalid files as masks.

Mask drawing and erasing: One of the cores objectives of the application is to support manual annotation, which makes the profound testing of the drawing and erasing tools mandatory. The brush draws effectively in all available thicknesses, as depicted in Figure 6.11 and the eraser is capable of erasing all painted lines, even when overlapping.

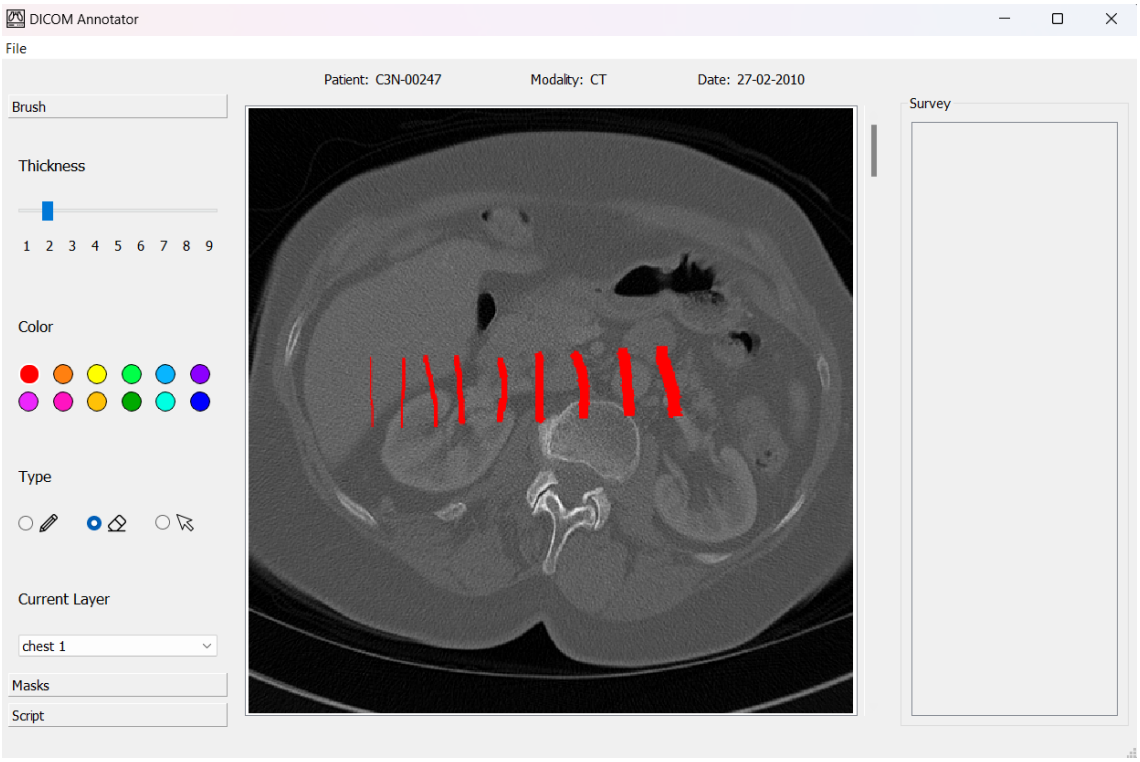
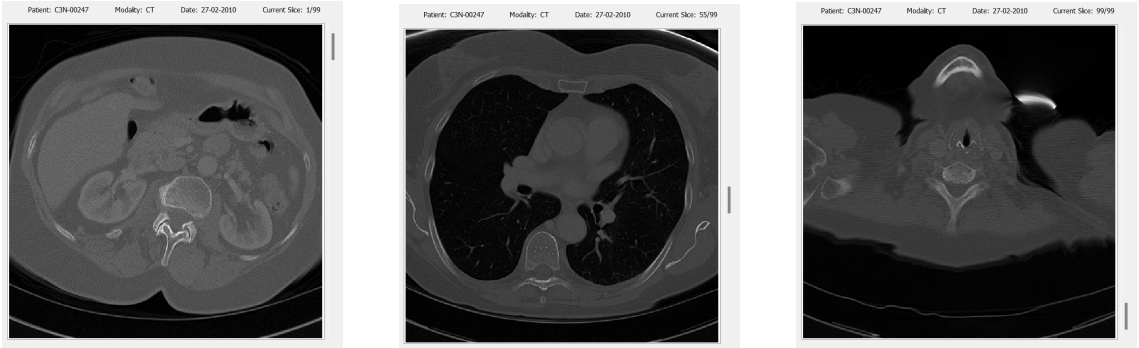


Figure 6.11: Brush drawing effectively in all available thicknesses.

Image and mask navigation: The navigation though the various loaded images runs smoothly, despite the velocity of the scroll and the quantity of slices, as ilustrated in Figure 6.12. Moreover, the scroll bar does not authorize the user to select off-limits values, which ensures that the navigation will always perform as expected.

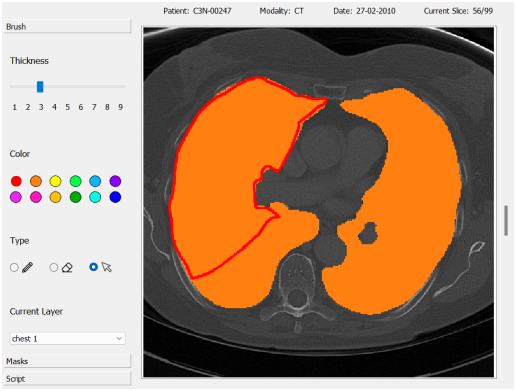


(a) Slice 1 out of 99. (b) Slice 55 out of 99. (c) Slice 99 out of 99.

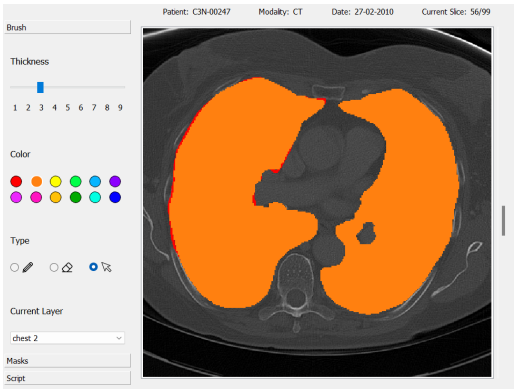
Figure 6.12: Navigation through the slices of the loaded dataset.

Mask switching: The layer switching can be achieved in diverse ways, which makes the testing complex and critical. The application should not only perform the changing of the mask that is being drawn on, but also change the pen color to the one that matches the mask and the name displayed on the "current layer" combo boxes, as depicted in Figure 6.13. The application delivers

the desired results when the user clicks the color buttons or selects the mask's name on the combo box. This is perceived as the erasing functionality affects only the annotations of the "current layer" chosen by the user, which implies that erases lines previously drawn in that mask and does not erase the content of the other masks on display.



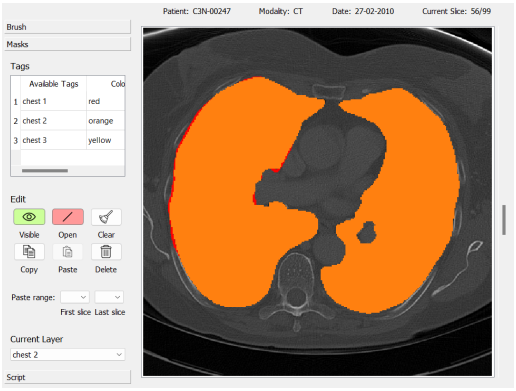
(a) Display of the *Current Layer* as *chest 1*.



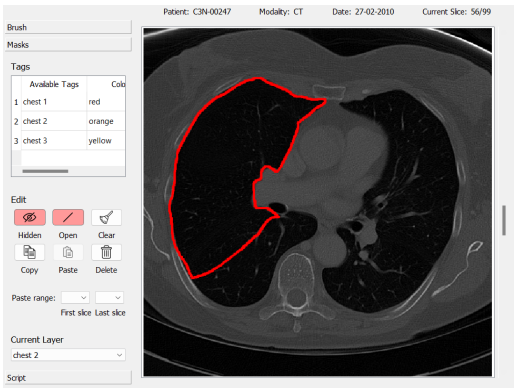
(b) Display of the *Current Layer* as *chest 2*.

Figure 6.13: Navigation through the layers of a slice.

Mask visibility management: It should be possible to toggle the visibility of the masks, to simplify the annotation and visualization processes. Although the tests to hide and show a single mask or all masks simultaneously were successful (Figure 6.14), this functionality still presents some dangerous flaws. When the user tries to draw in the invisible layer, the annotation will be placed in the layer that was right behind it, instead of not allowing the action. However, it is important to note that the visibility state is exposed in the GUI and the user can easily switch it to be able to draw correctly.



(a) Layer *chest 2* showing on *Display and Annotation Area*.

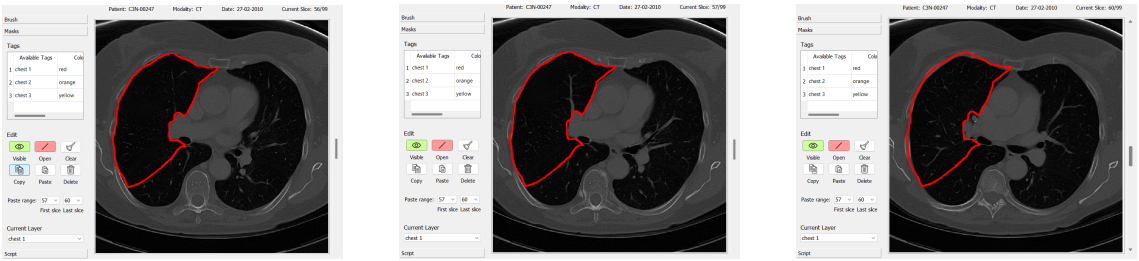


(b) Layer *chest 2* hiding on *Display and Annotation Area*.

Figure 6.14: Mask visibility test success, by showing and hiding the *chest 2* layer.

Mask copy and paste: This function is rather helpful when dealing with large DICOM

datasets, as it facilitates the manual annotation. In spite of that, it is noticeably in need of smoothing out the rough edges. The function fulfill its intended purpose, as illustrated in Figure 6.15, even when submitting invalid values in the range to paste, as it prints an explanatory error message. However, it has a small bug of a "ghost" mask when applying the copied masks to a range that does not contain the current image on display. This glitch does not influence the overall performance as it disappears once the image is switched in the scroll bar.



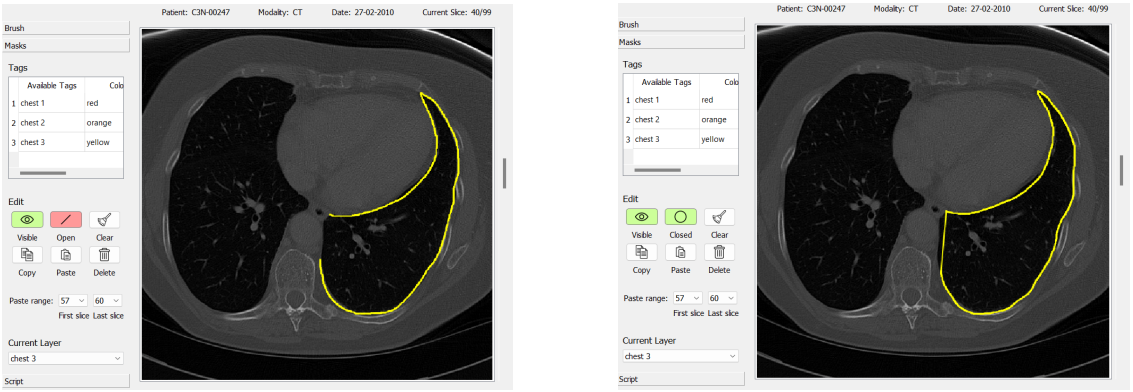
(a) Copy annotation from *chest 1* layer on slice 56.

(b) Successfully pasted the annotation on slice 57.

(c) Successfully pasted the annotation on slice 60.

Figure 6.15: Copy and paste an annotation into several masks.

Automatically closed annotation option: Annotations on medical images are frequently consummated in closed geometric or abstract forms, which makes it pertinent to have a function that automatically closes the lines the user draws. The application answers to this need in a satisfactory way, as it fulfills the intended purpose (Figure 6.16), even if in a eminently rudimentary approach. It allows the user to toggle the option and it keeps the associated selected option to the mask even after switching it.



(a) Open mode or normal annotation mode.

(b) Automatically close line mode.

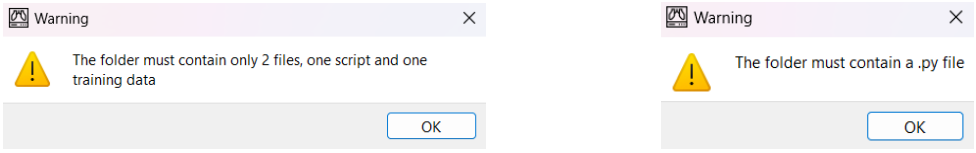
Figure 6.16: Automatically close line and normal annotation modes tested on a similar manual annotation.

Mask clearing: The application successfully replies to the *Clear* button clicking by removing all content on the selected mask, without deleting it from the mask management system.

Mask deleting: The test's response to deleting a single mask set or all masks simultaneously is positive. Once the masks are deleted, it is not possible to access them as they are deleted from the mask storage structure and removed from the *GraphicsView*, not leaving any trace of ever existing. They are then added back to the tag options in the dialog window.

Automatic annotation: A further core purpose of the application is to permit automatic annotation. This functionality is operating according to the specifications, considering it allows the user to select an external script and run it. It merely demands the pyhton script to have an external file for training data and a function named "run" within it (has to be the only time the word appears on the script), that returns a list of numpy arrays with the content of the new masks. The result when running a Lung CT Segmentation script developed by a INESC TEC research team [6] is depicted in Figure 6.17. In case this conditions are not fulfilled, the application prints error messages.

Figure 6.17: Automatic annotation by running an external script of Lung CT Segmentation developed by a INESC TEC research team [6].



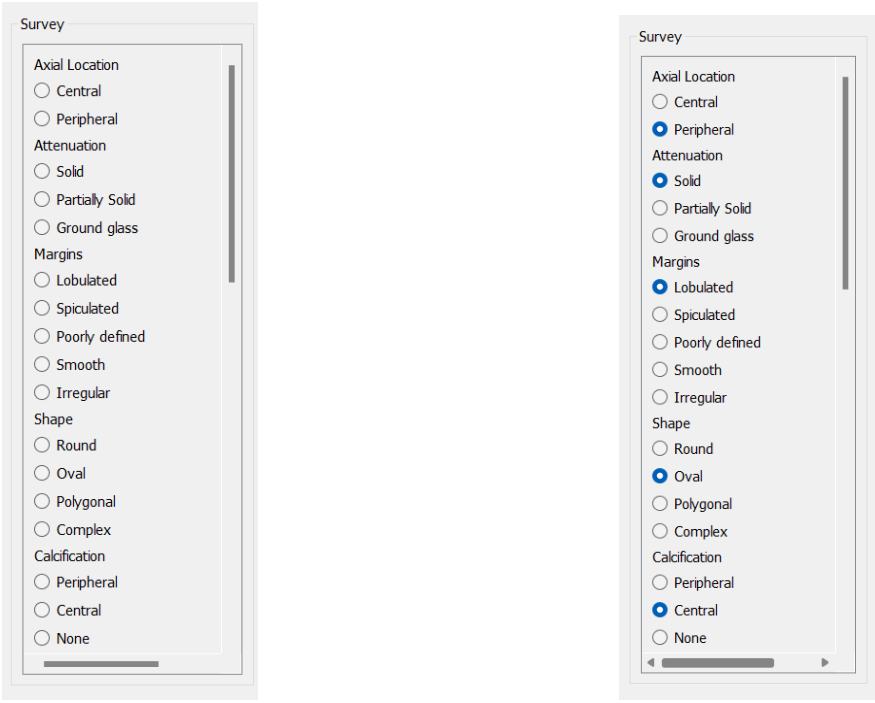
(a) Error message when the user selects a folder with more than two files.

(b) Error message when the user selects a folder that does not contain a script in python.

Figure 6.18: Error messages when loading invalid files as external scripts for automatic annotation.

Mask saving: The mask set is properly saved in a folder, in which the name is selected by the user. There are some obvious defects in this feature. The chosen name for the folder has to always be different from the already existing ones, or else it will replace the older one. Additionally, it is not possible to decide the location of the saved images, it will always go to the folder where the application is executing, which is undesired.

Survey management: All the steps of managing a survey, opening, editing and saving are performing exceptionally. The surveys are correctly loaded and displayed in the designated area for the user to observe and alter and saved with an elected name.



(a) Empty Survey.

(b) Filled Survey.

Figure 6.19: Survey for lung semantic annotation displayed in the *Survey Area*.

6.3 Evaluation of Application Usability

The evaluation of usability is of utmost importance in the development of an interface. It serves as a systematic process to assess the effectiveness, efficiency and satisfaction of the interface. By conducting a planned usability evaluation, developers can gather valuable insights into the strengths and weaknesses of the GUI. This allows them to make convenient decisions and generate improvements that enhance user satisfaction and engagement.

6.3.1 Methods Used to Evaluate Usability

The selected methodology involved the utilization of surveys, which were designed, conducted, and distributed among potential users to gather their responses. The collected survey data was subsequently analyzed. To achieve a successful usability assessment, a systematic sequence of steps has been established:

1. **Questionnaire Design:**
    - Create separate questionnaires for engineering experts and practicing doctors, customizing the questions based on their specific knowledge, needs, and expectations (Appendix A).
    - Ensure that questions related to the defined criteria are included, assigning weights to each.
    - Use rating scales on the available answers (such as 1 to 5) to facilitate result analysis.
2. **Evaluation Execution:**
    - Recruit participants for each phase.
    - Provide access to the application along with general usage instructions and guides outlining specific test sequences (Appendix B).
    - Ask participants to fill out the questionnaires after usage, allowing them to provide additional suggestions.
3. **Data Collection:**
    - Collect the questionnaires and ensure that all questions have been answered.
    - Convert the responses into data to facilitate analysis.
4. **Results Analysis:**
    - Calculate the average of responses for each question.
    - Apply defined weights to each criterion and obtain weighted scores for each group.
    - Compare scores to understand potential differences in usability perception and identify strengths and weaknesses, combining them to get an overall view of the application's performance in percentage terms.

5. **Results Presentation:**

   - Present the results in a clear and concise manner, using graphs or tables to highlight key findings.
   - Identify strengths and areas for improvement based on the defined criteria.
   - Consider including participant quotes or specific feedback to support the conclusions.

6.3.2 Findings of Usability Evaluation

After the evaluation with the experts and users, the results were processed, converting the qualitative answers into a quantitative scale of 1 to 5. This conversion was done to eliminate ambiguity and facilitate comparison, with a rating of 5 representing the ideal interface.

The participants in the expert evaluation phase were three colleagues from INESC TEC who are pursuing a doctorate in engineering (one in physics and two in biomedical engineering) and who work on projects related to the development of automatic segmentation algorithms. The test was preceded by a careful reading of the user manual. The proposed test involved testing all the functionalities of the application, followed by a request to respond to a short questionnaire. The criteria, weights, actual measures and overall rating taking into account the weights, are specified for expert evaluation phase in Table 6.1 and Figure 6.20.

Table 6.1: Criteria to evaluate the usability of the application for engineers with respective weights and measures on a scale of 1 to 5.

Usability Criteria	Normalized Weight	Actual Measure
Suitability for the Task (ST)	0.19	4
User Control (UC)	0.08	3.33
Flexibility (F)	0.10	3.33
Error Management (EM)	0.09	4.67
Compatibility (CP)	0.12	4
Self-descriptiveness (SD)	0.15	3.33
Consistency (CS)	0.13	4
User Workload (UW)	0.14	4.5
Rating	1.00	3.91

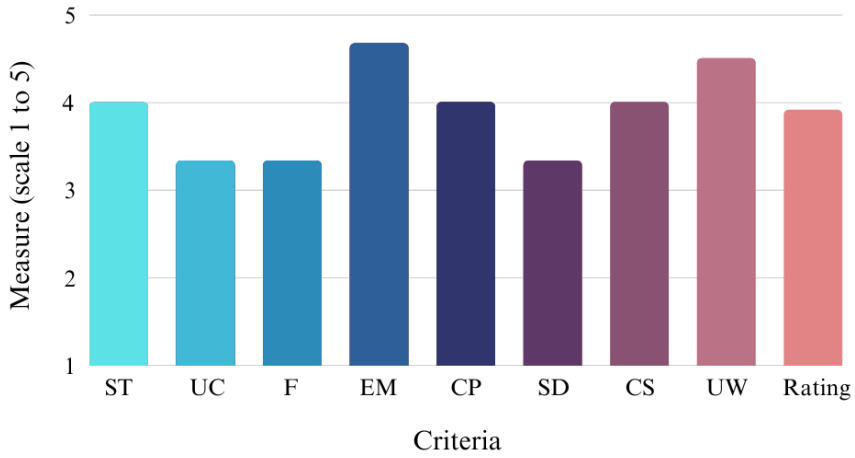


Figure 6.20: Graphic of the criteria and measure of the expert evaluation phase in a scale of 1 to 5.

The participants in the usability evaluation phase were two radiologist doctors with 11 and 12 years of experience as specialists in this field. The test was preceded by a brief reading of the user manual, which may have been a bit too short, but it was the best possible given the available time. During the test, the doctors were challenged to perform three tasks: manually annotating a dataset, automatically annotating a dataset, and semantically annotating a dataset. Afterward, they were asked to respond to a short questionnaire. The criteria and sub-criteria, corresponding weights, actual measures and overall rating, are specified for this evaluation phase in Table 6.2 and Figure 6.21.

Table 6.2: Criteria to evaluate the usability of the application for physicians with respective weights and measures on a scale of 1 to 5.

Criteria and Sub-criteria	Normalized Weight	Actual Measure
Effectiveness	0.35	3.5
Task Completion (TC)	0.60 [0.21]	3.5
Ratio of Errors (RE)	0.20 [0.07]	2.5
Severity of Errors (SE)	0.20 [0.07]	4.5
Efficiency	0.20	3.6
Task Completion Time (TCT)	0.20 [0.04]	4
Effort (E)	0.40 [0.08]	3.5
Learning Time (LT)	0.40 [0.08]	3.5
Satisfaction	0.45	3.17
Rating	1.00	3.37

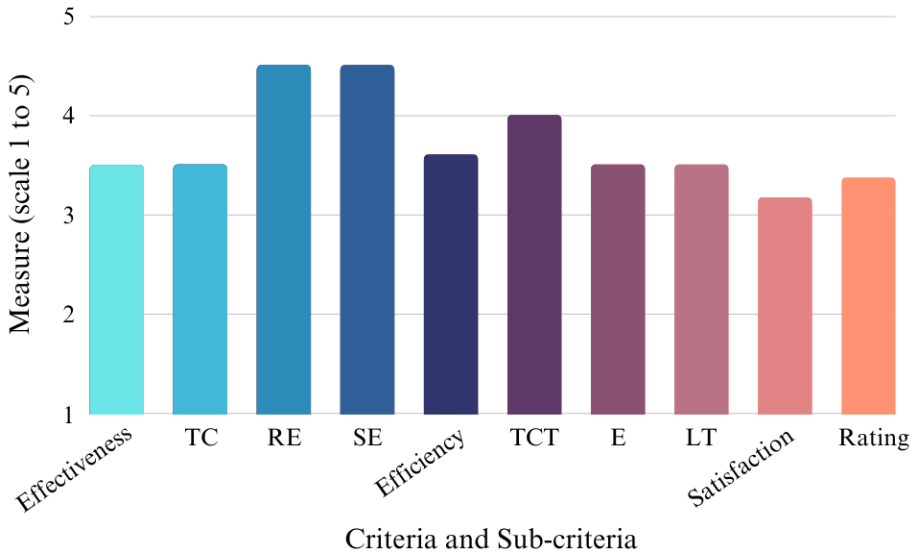


Figure 6.21: Graphic of the criteria and sub-criteria and measure of the user evaluation phase in a scale of 1 to 5.

Comparing the results, it is noticeable that the engineers were more satisfied with the performance of the application compared to the physicians. This may be attributed to several factors: the physicians spent significantly less time reading the user manual and found the interface more confusing; the physicians, who are more accustomed to using applications with a wide range of features, found the developed application too rudimentary; the feedback received by the physicians was provided very late and only once, whereas the engineers received feedback frequently throughout the project, which better aligns with their expectations than those of the radiologists; and, finally, it could have been due to the difference in the focus of the questions posed to each group, as the questions for the physicians heavily weighed their satisfaction, while those for the engineers were more focused on the application's functionality.

6.3.3 Strengths and Weaknesses

The analysis of the results obtained from the usability evaluations allowed us to draw conclusions about the strengths of the application and the improvements to be implemented, especially with the responses to the open-ended question in the questionnaire that asked for additional feedback and relevant opinions. The majority of the responses to this question suggested improvements: the zoom and undo functionality would be useful for the physicians to save time in manual annotation and to make it more accurate; more text messages should appear to indicate what the application is doing; the *Close/Open* button is not very practical, instead, it should have a function that connects points with a line to close annotations made by users; some interface components should be more visible and intuitive, such as the *File* button and the page selection in the *Tool Box*.

However, it was also mentioned that the simplicity of the application was a clear advantage since, compared to other applications previously used by the participants, it was much easier to

use. Another strong point of the application is its ability to prevent errors, identified by both engineers and physicians.

6.4 Comparison with Initial Expectations

The comparison of the requirements defined at the beginning of the project with the actual features accoutered in the developed tool is presented in Table 6.3. Additionally, in this table, the related tools are considered to help comprehend the advantages and disadvantages comparatively to the other available applications.

Table 6.3: Comparison of the initial defined requirements and the specifications of the related tools with the actual specifications of the application.

Requirements	Itk-Snap	Seg3D	MeVisLab	Fiji	Final Application
3D Annotation	●	●	●	○	◉
Re-annotation	◉	●	●	◉	●
Algorithm	○	●	●	●	●
Configurable GUI	●	●	○	●	●
Manual Annotation	●	○	●	●	●

● Implemented	◉ Half Implemented	○ Inutile	○ Missing

6.5 Limitations and Challenges

The first significant challenge encountered throughout the development of the application was the drawing and erasing functionalities. The original idea was to incorporate MatplotLib and Open CV libraries, as they offer powerful and complete frameworks for both display and manipulation of images. However, PyQt5 library, which was the chosen library to design the user interface, facilitates the creation of layers to avoid erasing the image content and to allow controlling multiple masks, representing the actual choice for the development of the drawing and erasing functions.

Another notably challenging feature to develop was the saving masks. The knowledge about DICOM files and the PyDicom library had to be profoundly extended to be able to create a DICOM file from scratch to then save it. At the beginning, the mask was being created by performing a copy of the corresponding image and changing the necessary attributes afterwards, which was not working correctly. The outstanding solution was to create the DICOM file and populate it freely with nothing but the essential attributes.

Initially, the approach to the project was to develop an application designed to take a single DICOM image. Once the functions were properly operating with only one image, they were

updated to take multiple images. This process was quite complex and time consuming as the variables' structures and the interface's design had to be rethought and readjusted.

6.6 Summary of Results and Discussion

In the testing process, the application delivered acceptable results in most implemented functions, yet some significant issues and limitations were discovered. The functionalities that encountered these limitations are the mask visibility management, saving and copy and paste, which affects the robustness of the application and may reduce the user's satisfaction. However, the problems are not urgent to solve as they do not represent a threat to the application's life and can be easily bypassed. Further testing is recommended to cover additional use cases and ensure comprehensive functionality and reliability.

The application was evaluated with experts and users, and their feedback was processed and converted into a quantitative scale of 1 to 5 for easier comparison. The evaluation included two phases: an expert evaluation with three colleagues from INESC TEC and a usability evaluation with two radiologist doctors. Comparing the results, it was observed that the engineers were more satisfied with the application's performance than the physicians. The analysis of the results highlighted a lot of areas for improvement. However, the participants appreciated the simplicity of the application compared to previously used ones and recognized the application's ability to prevent errors as a strong aspect.

Comparing the previously analyzed related tools with the developed tool, significant advantages can be observed in the vast majority of the initially imposed requirements. The majority of the objectives were fully achieved, with only the analysis of 3D images remaining partially implemented, using a slider in 2D images to simulate the depth effect.

Chapter 7

Conclusion

Communication among medical teams is fundamental to achieve positive influence in the lives of the sufferers. They must deal with a considerable amount of information in the minimum amount of time. It is then fundamental for the software design engineering to develop simple modules for visualization, annotation, detection, segmentation and classification of medical images to guide them to the best decision. As a result, a variety of applications that attempt to answer to all the clinical user's needs have been implemented.

Some of the most popular used free software tools for visualization and annotation of medical images are Itk-Snap, Seg3D, MeVisLab (with paid version also available) and Fiji. While these applications grant most of the requirements defined in the plan of the dissertation, none fully covers all of them, which reinforces the value of this project.

The fundamental models and mechanics that support the design and development of a software project such as programming languages, in particular high level programming languages, and graphical user interface design and evaluation are then explored. The PL chosen to implement the tool was Python, as it has an immeasurable amount of treasured libraries that help the developer create GUIs, such as the employed PyQt5, work with DICOM files, such as PyDicom, and integrate scripting plugins and external storage of information, for example in JSON format.

The system architecture follows the MVC design pattern, promoting modularity, maintainability and scalability. Each component contributes to the seamless completion of user tasks. A clear and user-friendly GUI design and well-defined and intuitive user workflows are very significant for the user. Various User Interaction Patterns are utilized to confirm this is achieved.

During the testing phase, the application demonstrated satisfactory outcomes in most implemented functions, yet encountered notable issues and limitations. The functionalities affected by these limitations include mask visibility management, saving, and copy and paste, which may impact the application's robustness and user satisfaction. However, these issues are not pressing and can be easily circumvented without posing a threat to the application's viability.

The application was evaluated by possible users, with their feedback being processed and quantitatively scaled from 1 to 5 for enhanced comparability. The usability evaluation encompassed two phases: expert evaluation involving three colleagues from INESC TEC and user eval-

uation involving two radiologist doctors. A comparison of the results revealed that engineers expressed greater satisfaction with the application's performance compared to physicians. Analysis of the results underscored numerous areas for improvement. Nonetheless, participants acknowledged the application's simplicity, favorably contrasting it with previous tools, and recognized its capacity to mitigate errors.

When juxtaposing the previously examined related tools with the developed tool, substantial advantages become apparent across the majority of initially specified requirements. Most objectives were fully accomplished, with the exception of partial implementation concerning the analysis of 3D images, where a 2D image slider was employed to simulate the effect of depth.

7.1 Future Work

The application developed successfully addressed various functions aimed at achieving the overall objectives of improving annotation in medical images for radiologists. However, there are still a few outstanding limitations that need to be resolved to fully accomplish the user's satisfaction. While progress has been made during the dissertation, additional efforts are required to ensure the completion of all essential components.

- Implement zoom function – Introduce a zoom feature, enabling users to magnify or reduce an image area. This functionality will enhance the precision and accuracy of manual annotations and help visualise certain areas.
- Implement undo function – Incorporate an undo feature that allows users to reverse their previous actions or annotations. This capability will provide users with greater flexibility and control, saving a significant amount of time.
- Display text messages for guidance – The application has proved to be very limited in communicating with the user by not providing feedback through text messages, so it is imperative to add more feedback to the user while performing tasks.
- Add *Help* button – Add a *Help* button to the *Menu Bar* with a digital user manual and access to the contacts of the developers.
- Enhance the visual appearance – The current aesthetic appeal is relatively modest. In order to elevate it, a deeper understanding of graphic design or the utilization of a template is required.
- Manipulate medical image and mask display – Some very useful additional functions are the manipulation of contrast and brightness of medical images and opacity of masks, to reinforce a proper visualization for diverse cases.

It is crucial to acknowledge that the application's development is an ongoing process, and its completeness can never be definitively achieved. There is always potential to introduce new and

valuable functions that may not have been previously considered. The utilization of the MVC design pattern facilitates efficient and effective implementation of such modifications.

Appendix A

Surveys for Usability Evaluation

INESCTEC

Expert Evaluation Survey

Name _____ Specialisation _____

Years as Specialist _____ Contact Phone _____ Date _____

QUESTIONS:	RATING SCALE:				
	Poorly/ Insufficient	Fairly/ Adequate	Well / Satisfactory	Very well/ Good	Extremely well/ Excellent
How well does the system meet your needs and requirements when performing tasks?	○	○	○	○	○
How would you rate the system's support in helping you accomplish specific tasks?	○	○	○	○	○
To what extent do you feel in control of the interface?	○	○	○	○	○
How would you rate the system's flexibility and customization options?	○	○	○	○	○
How effectively does the system prevent errors?	○	○	○	○	○
How would you rate the system's alignment with your existing conventions and expectations?	○	○	○	○	○
How well does the system provide clear feedback, guidance, and support?	○	○	○	○	○
How would you rate the system's ability to help you understand and effectively use its features?	○	○	○	○	○
How would you rate the system's consistency throughout your interaction?	○	○	○	○	○
How efficient is the software in preventing the cognitive overload?	○	○	○	○	○
How would you rate the system's ability to minimize cognitive load and promote efficient interaction?	○	○	○	○	○

Do you have any additional opinion or feedback regarding the product ? (Open-ended response)

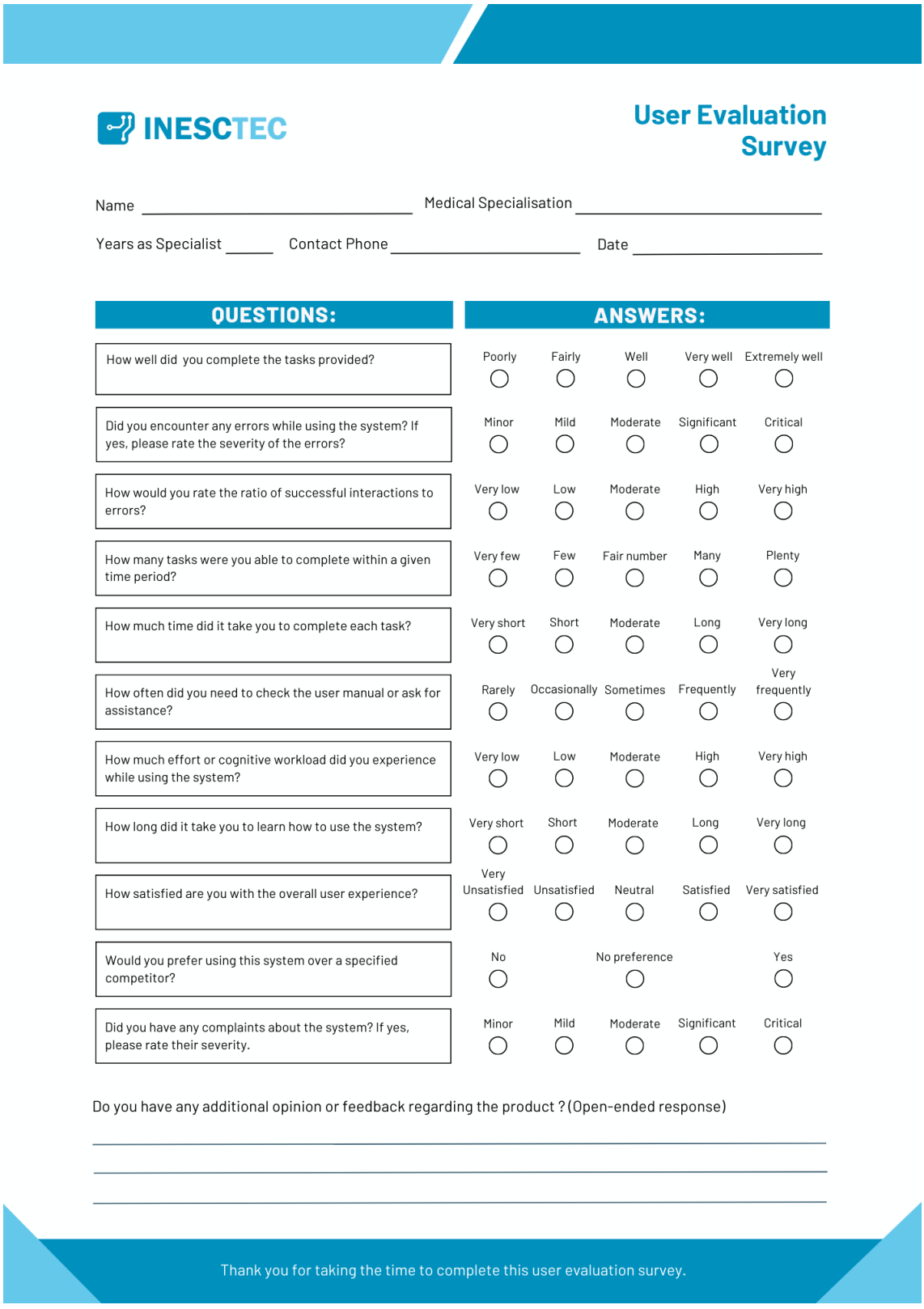
_____

_____

_____

Thank you for taking the time to complete this expert evaluation survey.

Figure A.1: Expert Evaluation Survey administered to the engineers.



INESCTEC

User Evaluation Survey

Name _____ Medical Specialisation _____

Years as Specialist _____ Contact Phone _____ Date _____

QUESTIONS:	ANSWERS:				
How well did you complete the tasks provided?	Poorly ◯	Fairly ◯	Well ◯	Very well ◯	Extremely well ◯
Did you encounter any errors while using the system? If yes, please rate the severity of the errors?	Minor ◯	Mild ◯	Moderate ◯	Significant ◯	Critical ◯
How would you rate the ratio of successful interactions to errors?	Very low ◯	Low ◯	Moderate ◯	High ◯	Very high ◯
How many tasks were you able to complete within a given time period?	Very few ◯	Few ◯	Fair number ◯	Many ◯	Plenty ◯
How much time did it take you to complete each task?	Very short ◯	Short ◯	Moderate ◯	Long ◯	Very long ◯
How often did you need to check the user manual or ask for assistance?	Rarely ◯	Occasionally ◯	Sometimes ◯	Frequently ◯	Very frequently ◯
How much effort or cognitive workload did you experience while using the system?	Very low ◯	Low ◯	Moderate ◯	High ◯	Very high ◯
How long did it take you to learn how to use the system?	Very short ◯	Short ◯	Moderate ◯	Long ◯	Very long ◯
How satisfied are you with the overall user experience?	Very Unsatisfied ◯	Unsatisfied ◯	Neutral ◯	Satisfied ◯	Very satisfied ◯
Would you prefer using this system over a specified competitor?	No ◯		No preference ◯		Yes ◯
Did you have any complaints about the system? If yes, please rate their severity.	Minor ◯	Mild ◯	Moderate ◯	Significant ◯	Critical ◯

Do you have any additional opinion or feedback regarding the product ? (Open-ended response)

_____

_____

_____

Thank you for taking the time to complete this user evaluation survey.

Figure A.2: User Evaluation Survey administered to the physicians.

Appendix B

User Manual

Figure B.1: Cover of User Manual.

INESCTEC User Manual

CONTENTS

Figure B.2: Contents of User Manual.

INESCTEC User Manual

1. GETTING STARTED

The developed aplication is a simple interface that allows users to visualize, annotate, and navigate through DICOM images, in a clinical environment. It supports manual, automatic and semantic annotations.

1.1 Conditions

The application is developed to run as a desktop application. It is possible to install on computers with both Window OS and macOS. The application does not require a connection to the internet and does not rely on external databases. It is free to use and open source.

1.2 Run

At the moment, it is only possible to run the application though code editors or Integrated Development Environment (IDE) , such as PyCharm or VS Code. As future work the python project will be converted to an executable application.

1.3 Interface

The interface is devided in six main components, as depicted in Figure 1. The *Menu Bar* (in yellow), responsible for the file management, the *Tool Box* (in red), where all the functions to manual and automatic annotate the masks are, the *Dataset Information* (in purple), which exhibits some relevant information about the dataset, as well as the number of the current slice on display, the *Display and Annotation Area* (in green), that allows both display and input submition of mouse events, the *Navigation Slider* (in pink), which permits the navigations though the slices and the *Survey Area* (in light blue), where the semantic annotation is performed.

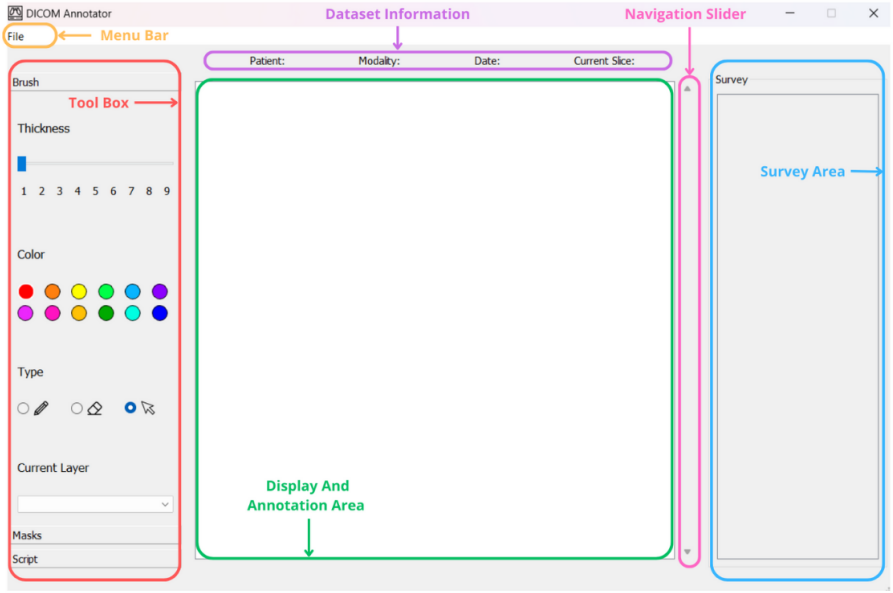


Figure 1 - Graphical User Interface (GUI) components of the application highlighted in boxes.

Figure B.3: First page of User Manual.

2. IMAGES

2.1 Open Images

The first step for any course of actions in the application is the medical images loading (in DICOM format). This is easily accomplish by clicking the *File -> Open -> Image* button in the *Menu Bar* (Figure 2) and selecting the folder with the desired dataset to study . The first slice is then displayed in the *Display and Annotation Area* and the related information is added to the *Dataset Information* (Figure 3).

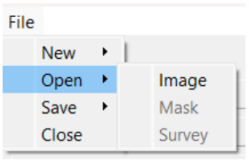


Figure 2 - *File -> Open -> Image* button in the *Menu Bar* to load the DICOM images to the application.

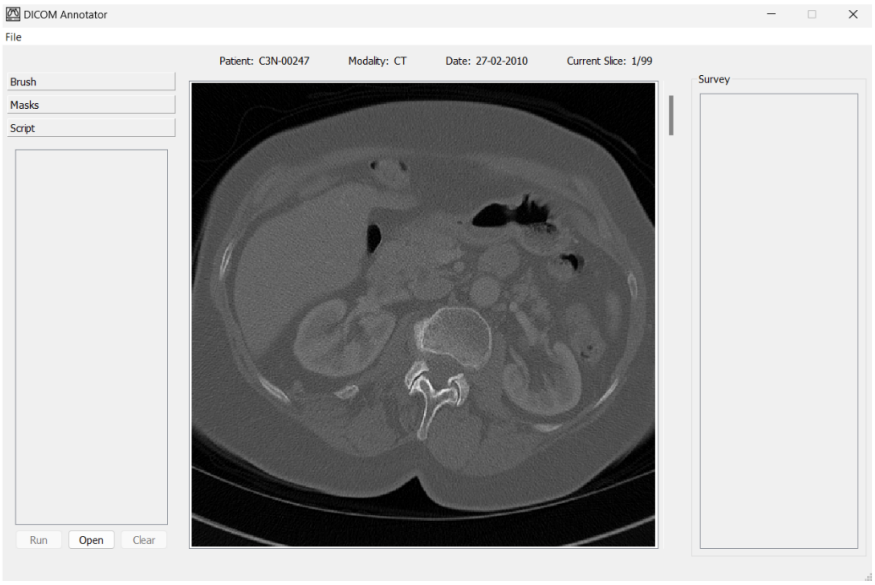


Figure 3 - First slice displayed in the *Display and Annotation Area* and information added to *Dataset Information*.

2.2 Image Navigation

Once loaded, it is possible to navigate through the images by scrolling on the Navigation Slider. The slice on display will be changed and the Current Slice attribute on the Dataset Information will be updated.

Figure B.4: Second page of User Manual.

3. MASKS

3.1 New Mask

To create a new mask for the loaded image, click on the *File -> New -> Mask* button in the *Menu Bar* (Figure 4). The mask tag has to be chosen (Figure 5) and then it will be added to all slices as an empty layer that hovers them, where the annotations can be performed. Additionally, the name will be added as an item to the *Current Layer* combo boxes, present on *Brush* page and *Masks* page of the *Tool Box*.

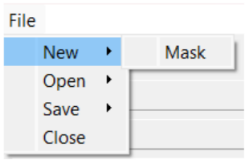


Figure 4 - *File -> New -> Mask* button in the *Menu Bar* to create the empty layers.

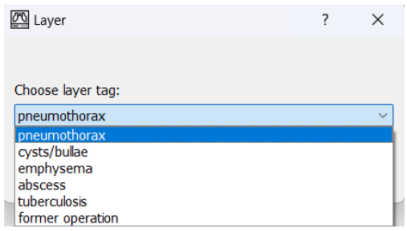


Figure 5 - Tag choice on a dialog window displayed when clicking the *File -> New -> Mask* button.

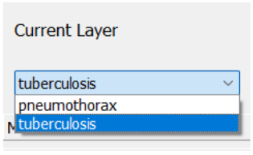
3.2 Open Maks

Other option is to load already existing masks. This is accomplish by clicking the *File -> Open -> Mask* button in the *Menu Bar* and selecting the folder with the desired dataset to study that is then displayed in the *Display and Annotation Area*, as a layer with the content of the loaded files, after choosing the tag name.

3.3 Layer Navigation

It is possible to select a different mask by clicking on the color button (Figure 6a) associated with the desired layer or choosing the desired layer's name on the *Current Layer* combo boxes (Figure 6b) with the list of masks.

(a) Color buttons present in *Brush* page of the *Tool Box*



(b) *Current Layer* combo box present in *Brush* and *Masks* pages of the *Tool Box*

Figure 6 - Possible ways of switching the layer to annotate.

Figure B.5: Third page of User Manual.

3.4 Save Mask

To save the masks, click on the *File -> Save -> Mask* button in the *Menu Bar*. Select a name for the folder (Figure 7) and then it will be created and the masks will be saved as DICOM files, ready to be loaded again.

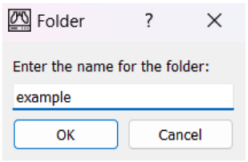


Figure 7 - Dialog window to write the desired name for the new folder.

4. MANUAL ANNOTATION

4.1 Brush Tool

The *Brush Tool* present in the *Brush* page in the *Tool Box* (Figure 8) allows both drawing and erasing actions. To draw, select a brush size in the *Thickness* slider, click on the pencil button of the *Type* buttons to switch to the draw state, and draw directly on the *Display and Annotation Area* with the mouse . Erase the drawing by selecting the eraser size in the Thickness slider, clicking the eraser button of the Type buttons and erasing the drawing on the *Display and Annotation Area*. Make sure that the mask on the *Current Layer* combo box is the one you want to edit.

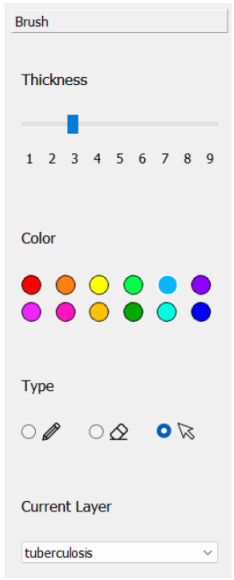


Figure 8 - *Brush* page in the *Tool Box*, with the *Thickness* slider, *Color* buttons, *Type* buttons and *Current Layer* combo box.

Figure B.6: Fourth page of User Manual.

4.2 Edit Masks Tools

There are several additional tools to edit the content of the masks and manage their annotations and display, present in the *Masks* page of the *Tool Box* (Figure 9).

- The *Visible/Hidden* button toogles the visibility of the current layer.
- The *Copy* button copies the content of the current layer, allowing to paste it on a specific range delimited by the *First Slice* and *Last Slice* combo boxes with the *Paste* button.
- The *Closed/Open* button allows the user to choose either normal annotation or automatically closed annotation modes, when selected *Closed*, the drawn lines are going to close automatically.
- The *Clear* button clears all the content of the current layer on display and the *Delete* button eliminates the current layer's masks, removing them from the system.

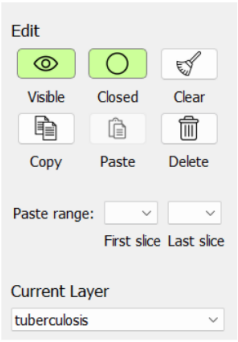


Figure 9 - Additional tools to edit and manage masks.

5. AUTOMATIC ANNOTATION

5.1 Open Script

To open the external script that generates automatic annotations click on the *Open* button in the *Script* page of the *Tool Box* (Figure 10) to trigger a folder selection dialog and select the desired folder that contains the script file and the training data file. The script content is displayed on the box after loaded to the system (Figure 11).

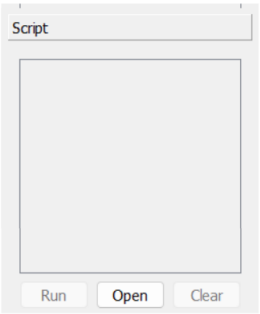


Figure 10 - *Open* button on the *Script* page of the *Tool Box*.

Figure B.7: Fifth page of User Manual.

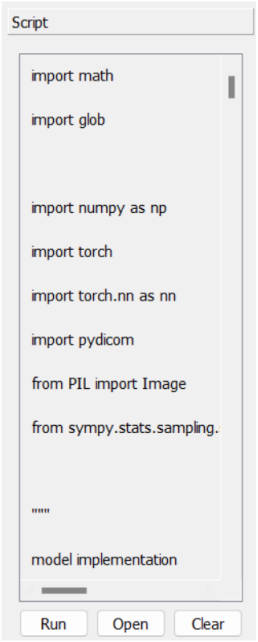


Figure 11 - Display of the loaded script content.

5.2 Run Script

After loading the script it is possible to run it, by clicking the *Run* button in the *Script* page of the *Tool Box*. Then the script will be executed and it will ask to choose the name tag, as it happens when loading or creating new masks.

5.3 Clear Script

To clear the displayed script and delete it from the system click on the *Clear* button in the *Script* page of the *Tool Box*.

6. SEMANTIC ANNOTATION

6.1 Open Survey

Click the *File -> Open -> Survey* in the *Menu Bar* and select a JSON file from the file selection dialog with the desired survey. It will then be displayed on the *Survey Area* (Figure 12).

Figure B.8: Sixth page of User Manual.

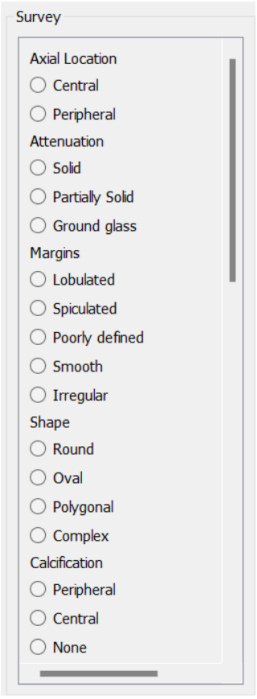


Figure 12 - Display of the loaded survey content on the *Survey Area*.

6.2 Edit Survey

After loading the survey it is possible to fill it, by clicking the buttons in the *Survey Area*.

6.3 Save Survey

To save the editted survey click on the *File -> Save -> Survey* button in the *Menu Bar* (Figure 13).

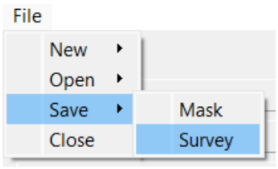


Figure 13 - Save the survey edited in the *Survey Area*.

7. EXIT

To exit the application click the X button on the top right corner of the interface window or the *Close* button in the *Menu Bar*. You can also minimize the window in the - button on the left side of the X button.

Figure B.9: Seventh page of User Manual.

References

[1] Paul A. Yushkevich, Jilei Hao, Sadhana Ravikumar, Alison M. Pouch, Ankush Aggarwal, and Lukasz Kaczmarczyk. Itk-snap downloads. Retrieved from http://www.itksnap.org/pmwiki/pmwiki.php?n=Downloads.SNAP4, 2023.

[2] NIH Center for Integrative Biomedical Computing at the University of Utah Scientific Computing and Imaging (SCI) Institute. Download seg3d. Retrieved from https://www.sci.utah.edu/download/seg3d, 2015.

[3] MeVis Medical Solutions AG and research institute Fraunhofer MEVIS. Download mevislab. Retrieved from https://www.mevislab.de/download, 2023.

[4] Fiji Software Sustainability Grant. Fiji downloads. Retrieved from https://imagej.net/software/fiji/downloads, 2022.

[5] Kyung S Park and Chee Hwan Lim. A structured methodology for comparative evaluation of user interface designs using usability criteria and measures. *International Journal of Industrial Ergonomics*, 23(5):379–389, 1999.

[6] Francisco Silva, Tania Pereira, Joana Morgado, António Cunha, and Hélder P. Oliveira. The impact of interstitial diseases patterns on lung ct segmentation. In *2021 43rd Annual International Conference of the IEEE Engineering in Medicine Biology Society (EMBC)*, pages 2856–2859, 2021.

[7] Qifei Dong, Gang Luo, David Haynor, Michael O'Reilly, Ken Linnau, Ziv Yaniv, Jeffrey G. Jarvik, and Nathan Cross. Dicomannotator: a configurable open-source software program for efficient dicom image annotation. *Journal of Digital Imaging*, 33(6):1514 – 1526, 2020. Cited by: 3; All Open Access, Green Open Access.

[8] Xiang Yu, Jian Wang, Qing-Qi Hong, Raja Teku, Shui-Hua Wang, and Yu-Dong Zhang. Transfer learning for medical images analyses: A survey. *Neurocomputing*, 489:230–254, 2022.

[9] Mingrui Zhuang, Zhonghua Chen, Hongkai Wang, Hong Tang, Jiang He, Bobo Qin, Yuxin Yang, Xiaoxian Jin, Mengzhu Yu, Baitao Jin, Taijing Li, and Lauri Kettunen. Anatomysketch: An extensible open-source software platform for medical image analysis algorithm development. *Journal of Digital Imaging*, 2022. Cited by: 0; All Open Access, Green Open Access, Hybrid Gold Open Access.

[10] Andreas Holzinger, Benjamin Haibe-Kains, and Igor Jurisica. Why imaging data alone is not enough: Ai-based integration of imaging, omics, and clinical data. *European Journal of Nuclear Medicine and Molecular Imaging*, 46(13):2722–2730, 2019.

[11] João Pedro Fonseca Teixeira. An anatomical breast atlas: automatic segmentation of key points in multiple radiological modalities. 2022.

[12] Felix Ritter, Tobias Boskamp, André Homeyer, Hendrik Laue, Michael Schwier, Florian Link, and H-O Peitgen. Medical image analysis. *IEEE pulse*, 2(6):60–70, 2011.

[13] Atam P Dhawan. *Medical image analysis*. John Wiley & Sons, 2011.

[14] Amy J Ko. What is a programming language, really? In *Proceedings of the 7th international workshop on evaluation and usability of programming languages and tools*, pages 32–33, 2016.

[15] Alan F Blackwell. First steps in programming: A rationale for attention investment models. In *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*, pages 2–10. IEEE, 2002.

[16] ISO/IEC FDIS 9126-1. Software engineering–product quality–part 1: quality model. 2000.

[17] Michael Lee Scott. *Programming language pragmatics*. Morgan Kaufmann, 2000.

[18] Edmund Callis Berkeley and Daniel Gureasko Bobrow. *The programming language LISP: Its operation and applications*. Citeseer, 1964.

[19] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The definition of standard ML: revised*. MIT press, 1997.

[20] B O'Sullivan, D Stewart, and J Goerzen. Real world haskell.— o'reilly media. *Sebastopol, CA, USA*, 2008.

[21] Christiaan Peter Jozef Koymans. Models of the lambda calculus. *Inf. Control.*, 52(3):306–332, 1982.

[22] William B Ackerman and Jack B Dennis. Val–a value-oriented algorithmic language (preliminary reference manual). Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE, 1979.

[23] William F Clocksin and Christopher S Mellish. *Programming in PROLOG*. Springer Science & Business Media, 2003.

[24] Ian D Chivers and Jane Sleightholme. *Introduction to programming with Fortran*, volume 2. Springer, 2018.

[25] Brian W Kernighan and Dennis M Ritchie. *The C programming language*. Pearson Educación, 1988.

[26] Adele Goldberg and David Robson. *Smalltalk-80: the language and its implementation*. Addison-Wesley Longman Publishing Co., Inc., 1983.

[27] Bjarne Stroustrup. *The C++ programming language*. Pearson Education, 2013.

[28] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java language specification*. Addison-Wesley Professional, 2000.

[29] R Mokua, W Mwangi, and JM Kanyaru. Design and implementation of an object oriented programing language. In *Scientific Conference Proceedings*, 2012.

[30] Piotr J Slomka, Edward Elliott, and Albert A Driedger. Java-based pacs and reporting system for nuclear medicine. In *Medical Imaging 2000: PACS Design and Evaluation: Engineering and Clinical Issues*, volume 3980, pages 235–241. SPIE, 2000.

[31] Balaji Raman, M Wilson, I Benche, and Peter Soliz. A java-based system for segmentation and analysis of retinal images. In *16th IEEE Symposium Computer-Based Medical Systems, 2003. Proceedings.*, pages 336–339. IEEE, 2003.

[32] Ken Arnold, James Gosling, and David Holmes. *The Java programming language*. Addison Wesley Professional, 2005.

[33] Nancy E Briggs and Ching-Fan Sheu. Using java in introductory statistics. *Behavior Research Methods, Instruments, & Computers*, 30(2):246–249, 1998.

[34] Saeed Seyyedi, Kubra Cengiz, Mustafa Kamasak, and Isa Yildirim. An object-oriented simulator for 3d digital breast tomosynthesis imaging system. *Computational and mathematical methods in medicine*, 2013, 2013.

[35] Marcel Bosc, Torbjorn Vik, Jean-Paul Armspach, and Fabrice Heitz. Imlib3d: An efficient, open source, medical image processing framework in c++. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 981–982. Springer, 2003.

[36] Hoe-Chun Woon and Yoon-Teck Bau. Difficulties in learning c++ and gui programming with qt platform: View of students. In *Proceedings of the 2017 International Conference on E-commerce, E-Business and E-Government*, pages 15–19, 2017.

[37] P Duval, M Lomperski, J Szczesny, H Wu, T Kosuge, and J Bobnar. Acop .net: Not just another gui builder. PCaPAC, 2018.

[38] Abdullrahman Yousif Bahar, Samer Mahmoud Shorman, Moaiad Ahmad Khder, Abdullah Muhammad Quadir, and Sayed Ahmed Almosawi. Survey on features and comparisons of programming languages (python, java, and c#). In *2022 ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems (ICETSIS)*, pages 154–163. IEEE, 2022.

[39] Dana Tsoy, Yevgeniya Daineko, and Madina Ipalakova. Cardiovascular system pathologies studying using virtual reality tools. In *2022 International Conference on Engineering & MIS (ICEMIS)*, pages 1–4. IEEE, 2022.

[40] Dave Kuhlman. *A python book: Beginning python, advanced python, and python exercises*. Dave Kuhlman Lutz, 2009.

[41] KR Srinath. Python–the fastest growing programming language. *International Research Journal of Engineering and Technology*, 4(12):354–357, 2017.

[42] AL Sayeth Saabith, T Vinothraj, and MMM Fareez. Popular python libraries and their application domains. *Development*, 7(11), 2020.

[43] Christopher P Bridge, Chris Gorman, Steven Pieper, Sean W Doyle, Jochen K Lennerz, Jayashree Kalpathy-Cramer, David A Clunie, Andriy Y Fedorov, and Markus D Herrmann. Highdicom: A python library for standardized encoding of image annotations and machine learning model outputs in pathology and radiology. *Journal of Digital Imaging*, 35(6):1719–1737, 2022.

[44] Matthew D Blackledge, David J Collins, Dow-Mu Koh, and Martin O Leach. Rapid development of image analysis research tools: bridging the gap between researcher and clinician with pyosirix. *Computers in biology and medicine*, 69:203–212, 2016.

[45] Alan Dix, Janet Finlay, Gregory D Abowd, and Russell Beale. *Human-computer interaction*. Pearson Education, 2003.

[46] Debbie Stone, Caroline Jarrett, Mark Woodroffe, and Shailey Minocha. *User interface design and evaluation*. Elsevier, 2005.

[47] ISOBSENBritish Standard. Ergonomic requirements for office work with visual display terminals (vdts)–part 11: Guidance on usability. iso standard 9241-11: 1998. *International Organization for Standardization*, 55, 1998.

[48] Kristen K Greene, John Michael Kelsey, Joshua M Franklin, et al. *Measuring the usability and security of permuted passwords on mobile platforms*. US Department of Commerce, National Institute of Standards and Technology, 2016.