Staffan Holmnäs

DESIGN AND DEVELOPMENT OF AN INTERACTIVE DENTAL CHART

Thesis CENTRIA UNIVERSITY OF APPLIED SCIENCES Information Technology May 2023



ABSTRACT

Centria University	Date	Author
of Applied Sciences	4.5.2023	Staffan Holmnäs
Degree programme		
Information Technology		
Name of thesis		
DESIGN AND DEVELOPMEN	F OF AN INTERACTIVE	E DENTAL CHART
Centria supervisor		Pages
Jari Isohanni		45 + 10
Instructor representing comr	nissioning institution	or company
Gunilla Sjöholm	-	

The purpose of this thesis was to develop an interactive dental chart in the Delphi programming language. A company called Abilita commissioned the software development task described in this thesis. The task consisted of graphic design and programming of functionality with the goal to create a complete dental chart. Dental charts are used by dentists and dental assistants to monitor the status of the patient's oral health. Therefore, it is important that a computerized dental chart includes the ability to show a variety of treatments and diagnoses including tooth decay and periodontal disease.

The topic of this thesis is to demonstrate how to design and develop the interactive elements in a dental charting application. Charts must visualize the components and be comprehensible to the user. Furthermore, the balance between graphical fidelity and comprehensibility must be considered. Consequently, the objective is to demonstrate how to implement intuitive and distinctive graphics visually representative of human teeth. In the finished software, users enter dental notations as inputs which then automatically generate a graphical visualization of the dental issues. The tools used for development consisted mainly of image manipulation software and the Delphi integrated development environment.

Designing a graphical dental chart requires knowledge of concepts from the fields of both dental healthcare and information technology. The goal of this thesis was to connect these two fields and demonstrate how to develop a lucid dental chart.

Key words

Delphi, dental charting, graphics, image manipulation, programming

CONCEPT DEFINITIONS

ADA

The American Dental Association, an organization comprised of professionals in the field of dentistry.

Anti-aliasing

A tool that smooths out the jagged edges of an image by averaging the colours of adjacent pixels.

Delphi

Delphi is a programming language and an IDE developed by Embarcadero. Based on the Object Pascal programming language, Delphi can generate code to a variety of operating systems and platforms. The Delphi IDE is available on the Windows Operating system.

Dental Charting

A dental chart is a graphical view of the teeth representing the record of a patient's treatments. During examination, the dentist uses dental notation to record treatments and diagnoses on each tooth. Different tooth surfaces can be marked a certain colour representing for instance tooth decay (caries) or fillings.

FDI notation

FDI notation is an international system for encoding teeth. It is standardized by the International Organization for Standardization as ISO 3950. The system divides the mouth into quadrants and assigns each tooth a unique number based on the quadrant and position.

GIMP

GIMP stands for GNU Image Manipulation Program; it is an open-source graphics editor designed to be used for modifying images.

GNU

GNU is an open-source operating system inspired by Unix. GNU is a recursive acronym that stands for GNU's Not Unix.

GUI

Graphical User Interfaces are interactive visual components in computer software. Examples of such components are icons, menus, lists and buttons. They are typically controlled with mice or touch screens.

ICT

Information and Communications Technology involves all the technologies that enable modern computing and networking.

IDE

An Integrated Development Environment is a piece of software used to build and develop applications. IDEs have graphical user interfaces and provide tools such as code editors, debuggers, and compilers.

OOP

Object-Oriented Programming is a programming methodology using classes and objects to represent data and methods.

Periodontal Charting

The dentist probes the depth of pockets between a tooth and the gum and notes the distance graphically on a chart. The amount of gingival recession is also noted, meaning how much the gums have receded. Six measurements around each tooth are typically taken.

RAD

Rapid Application Development is both a type of programming methodology and the name of an IDE used for development in the Delphi/Object Pascal language.

VCL

The Visual Component Library is a Delphi framework for developing graphical user interfaces for Windows.

ABSTRACT CONCEPT DEFINITIONS CONTENTS

1 INTRODUCTION	1
2 TASK OVERVIEW	3
3 THEORETICAL BACKGROUND	6
3.1 Dental notation and charting	6
3.2 Software and tools	9
3.3 Delphi	
3.4 Graphics	12
4 DEVELOPING A DENTAL CHART	16
4.1 Design	
4.1.1 Dental status chart	
4.1.2 Periodontal chart	
4.2 Development	21
4.2.1 Parsing inputs	21
4.2.2 Filling functionality	23
4.2.3 Drawing procedures	
4.2.4 Chart functions	
4.3 Testing	34
4.4 Layout and program flow	34
4.5 Results	37
5 DISCUSSION	40
5.1 Assessment	41
5.2 Improvement suggestions	41
6 CONCLUSION	43
REFERENCES	
APPENDICES	
TABLES	
TABLE 1. Requirements for the development task	4
FIGURES	
FIGURE 1. The commissioner's logotype	3
FIGURE 2. The FDI tooth numbering system	7
FIGURE 3. The names of tooth surfaces	8
FIGURE 4. GIMP tools	9
FIGURE 5. RAD Studio 10.2	
FIGURE 6. Aliasing and anti-aliasing	
FIGURE 7. Bitmap and vector image comparison	14
FIGURE 8. The dental status	
FIGURE 9. Permanent and primary teeth	

FIGURE 10. Tooth surface areas	
FIGURE 11. Four types of roots	20
FIGURE 12. Periodontal chart design	21
FIGURE 13. A coloured tooth surface	22
FIGURE 14. Pocket depth lines	22
FIGURE 15. Filling procedure call flow	24
FIGURE 16. Flood fill functionality	25
FIGURE 17. Lines on the periodontal chart	29
FIGURE 18. Clickable symbols	
FIGURE 19. A colour box component	
FIGURE 20. An artifact found during testing	
FIGURE 21. The layout of components in the form designer	
FIGURE 22. The program flow	
FIGURE 23. The dental chart	

CODES

23
25
27
28
30
30
32
33

1 INTRODUCTION

Dentists and dental assistants use dental charts to store the records of each patient's dental health. These charts are used for examination, diagnosis, and treatment. Computerized dental charts resemble their paper counterparts; however, they provide many advantages compared with paper charts. For instance, graphics and symbols can more easily be drawn or erased to illustrate dental treatments. Graphical interactive dental charts are clearer and reduce the risk of errors and mistakes. However, there are challenges in designing and developing dental charts. If the graphics are too detailed, the information that is meant to be conveyed might be lost in the clutter. Keeping this in mind, I developed a dental chart commissioned by a company. In this thesis report, I will demonstrate how I designed and developed a visually distinctive interactive dental chart.

The commissioner, a company in Information and Communications Technology (ICT) called Abilita, provided me with the task to design the interactive dental chart for a dental patient information solution used by the healthcare sector. The commissioner required original graphics for an existing dental patient system to be designed without relying heavily on prior work. My work is limited to the design and implementation of the dental charting module for the patient application.

The objective of my task was to develop a dental chart that integrates with existing software to allow for a user-friendly experience. The result should mirror the requirements and take into consideration the inputs from dental professionals. Users must easily be able to determine the patient's dental health by briefly looking at the screen. To create a clear dental chart, it is important to find a good balance between high graphic fidelity and high contrast, intuitive and distinctive graphics. The problem can be defined as the following: "How can a visually distinctive and interactive dental charting program be implemented given a set of requirements?".

To comprehend the concepts in this thesis, it is important to have a basic understanding about common dental issues and dental notation. Therefore, the theoretical part of the thesis provides references to existing dental charts. From a technical standpoint, the tools and programming language used are important. Thus, the Delphi Integrated Development Environment (IDE) and its capabilities are presented. A presentation of the 2D libraries used for development also needs to be included. Additionally, the image manipulation software used for creating the graphics is also explained in the theoretical part.

My goal is to perform research on existing dental charts and to develop an improved dental charting program. The commissioner should be able to integrate the finished dental chart into their application and offer a visually appealing solution with distinctive graphics. The purpose of this thesis is to clearly explain my task, goals, and the development process. I will also describe dental theory and concepts related to my task. Finally, I will show the results and discuss the importance of my work and suggest improvements. I hope that the commissioner will find the thesis report interesting. Ultimately, the aim is to allow the company to offer a competitive dental charting solution.

2 TASK OVERVIEW

In this chapter, I will present the company who I worked for developing the dental charting program. I will also explain in more detail the development task, which is the topic of this thesis. The requirements for the task are summarized at the end of this chapter.

The commissioner of the development task is an ICT company called Abilita (FIGURE 1). Located on the west coast of Finland in a town called Jakobstad, Abilita was founded in 1990 and the company offers consulting, IT services, and system solutions for both the public and private sectors. The company currently employs 67 people and is thus one of the largest ICT companies in the region. I have previously done my internship at Abilita, so the company was familiar to me. For the development project presented in this thesis, I had roughly three and a half months to plan my work and complete the task described below. (Abilita 2023.)



FIGURE 1. The logotype of the company that commissioned my task (Abilita 2023)

The public health sector is among the clients who are offered solutions by Abilita (Abilita 2023). One such solution is a patient information system for dental applications, for which the dental charting part needed to be updated from the beginning. My task consisted of designing and developing the graphics and functionality of the dental chart as a separate module. The end-users of this application are typically dentists, dental hygienists, and dental assistants active in Finland. A typical use case is that the dental assistant enters the dental treatments into the program as the dentist dictate them using dental notations during an examination. The graphics on the chart are automatically updated based on the assistant's inputs.

My task was to design the dental chart and create the functions that update the graphics. I was given a set of requirements such as the programming language, screen resolution, input data type, a list of functionalities and the layout of the dental chart. Both a tooth chart and a

periodontal chart had to be included. These were either customer requirements or suggestions and instructions from co-workers at Abilita. Although I received a lot of assistance and guidance, most of the development work was done independently.

Before beginning with implementing the task, I had to learn the basics of dental jargon, dental notation, and the basic requirements for dental charting. These concepts will be part of the theoretical framework, as they are quite important for understanding the task. By studying previous work and dental charts, I was able to understand the requirements and project definitions. The possibility to display for instance cavities and fillings on a tooth chart, and at the same time visualize pocket depth on a periodontal chart are a common feature of most dental charts (NEBDN 2020, 9-11). Consequently, my priority was to implement these two features first.

The requirements specify that the program must be written in the Delphi programming language for the Microsoft Windows operating system. Users input string data types using a keyboard and the result is displayed on a window on the screen measuring 800 by 900 pixels. The requirements include a dental chart that provides the function for colouring tooth surfaces up to three different colours per surface. The colours represent for instance cavities, fillings, and erosion. Another requirement is a periodontal chart that should include graphs that show measurements of pocket depth and gingival recession in millimetres based on the user's measurements. Additionally, the chart must also display any mobility, furcation, implants, crowns, root canals, and missing teeth. Additional functionalities include a legend, periodontal history, a print function, and localization in the Swedish and Finnish languages. (TABLE 1.)

Requirement	Description
	Description
Target platform	Windows
Programming language	Delphi
Input method	Keyboard
Input datatype	String

TABLE 1. Descriptions of the requirements for the development task

Window size	820 x 900 pixels
Layout	Dental chart and periodontal chart
Localization	Swedish and Finnish

For the implementation, image manipulation software was used to design the graphics for the teeth. The inputs needed to be parsed programmatically and the drawing functions developed. For programming the functionality, I used the Delphi IDE, which I had no prior knowledge of and had to learn. A specific 2D library was recommended, so I had to learn how to implement the most common draw functions from the imported 2D library. These concepts will be described in greater detail in the next chapter. In summary, the task included concepts such as design, development, debugging, testing, localization, integration, and documentation. The goal of this task was to develop a dental charting program that could be integrated with an existing patient information application.

3 THEORETICAL BACKGROUND

When developing a dental charting application, it is important to have a basic knowledge of dental notation and the examinations, diagnoses, and treatments performed by dentists. These concepts are explained in this chapter. Furthermore, it is important to show an example of a dental chart and describe its functionality. Additionally, the Delphi development environment and programming language used for this development task are also described. Finally, the basics of concepts related to graphics and image manipulation software are also explained.

3.1 Dental notation and charting

The focus in this chapter is on common terminology and concepts from dentistry. Particularly important for the topic are the concepts of dental notation and dental charting. While unrelated to the field of ICT, these concepts are relevant to understanding the development task. When developing a computerized dental chart, however, concepts from both fields are connected and explained later in this thesis.

The dentist uses dental notation when communicating with the dental assistant during an examination. The American Dentist Association (ADA) previously suggested the use of the Palmer notation system; however, since 1968, the universal numbering system was instead recommended. Another notation system was proposed by the Fédération Dentaire Internationale (FDI) and has since been adopted by several organizations. FDI notation is a two-digit system where the first digit signifies the quadrant, and the second number is the sequential number of the tooth. (Nelson & Ash 2010, 4; NEBDN 2020, 8.) The numbers corresponding to each tooth according to the FDI dental notation system are presented in figure 2 (Honkala 2022). pysyvät hampaat



FIGURE 2. The FDI tooth numbering system (Honkala 2022)

When performing an oral examination, it is common to divide each tooth into five surfaces (FIGURE 3). The names for these surfaces originate from the Latin language. The first is the occlusal surface, or the surface most used for chewing. Surface number two is called the mesial surface and it points towards the midline which separates the incisors. The third surface is called the buccal surface and it points towards the closest cheek or towards the lip. The fourth is the distal surface and points away from the midline or towards the back of the mouth. Surface number five is called the lingual surface and it points towards the tongue. In figure 3 below, the tooth surfaces are pictured from the perspective of one of the molars in the first quadrant. (Honkala 2022; NEBDN 2020, 4; Nelson & Ash 2010, 7.)



FIGURE 3. The names and positions of the surfaces of each tooth (adapted from Honkala 2022)

Dental charts are essential tools used by dentist assistants and dental nurses to monitor the status of a patient's mouth. A dental chart resembles a map of teeth and is often displayed from a top-down perspective. Every tooth on the chart is divided into five surfaces, and each one is given names in Latin indicating their direction. These surfaces are used during an examination to show completed work and diagnoses and treatments to be carried out. Examples of treatments include cavities, fillings, crowns, and implants. (NEBDN 2020; Hollins 2013, chapter 12.)

Computerized dental charts allow the dentist to record the patient's condition and determine how they respond to treatment. The purpose of the dental chart is not to overload the user with information. On the contrary, the goal is to clarify the patient's dental issues by drawing graphics that allow for the easy retrieval and comparison of the dental status. A periodontal chart includes the measurement of periodontal pockets (Hollins 2013, 327). A pocket between the gingiva and the tooth is probed by the dentist and the depth of the pocket is recorded and visualized on the chart. Furcation is another periodontal issue at the fork of a multirooted tooth. Furcation is also detected by probing and visualized by the index of severity on the dental chart. If the tooth has become unstable, the degree of mobility can also be recorded on the chart. (Newman, Takei, Klokkevold & Carranza 2019, 82.)

The registered dental hygienist and former director at the Utah College of Dental Hygiene, Brent Molen (2010), has made available a scanned version of a paper periodontal evaluation chart (APPENDIX 1). The chart shows images of teeth viewed from the side and from the top. Dentist's assistants note on the paper chart the measurements of pocket depth and gum recession taken during the examination (Hollins 2013, 327). Traditional paper charts are useful as templates when designing a computerized dental chart. Other useful tools include image editing software used for drawing and modifying the images of teeth, and a programming environment for coding the functionality. These pieces of software are described in greater detail in the next subchapter.

3.2 Software and tools

One of the most popular free and open-source image manipulation software is GIMP, which stands for GNU Image Manipulation Program (GIMP). GIMP is available for Microsoft Windows, macOS, GNU/Linux, and Unix operating systems. The program features functionality for drawing, transforming, filtering and much more (FIGURE 4). It is also possible to customize GIMP by using scripts and plug-ins. (Lecarme & Delvare 2013.)



FIGURE 4. The tools available in GIMP (David 2023)

Gimp can be used for e.g., photo manipulation, artwork creation, and graphic design. The programming languages supported for programming scripting algorithms include Python, C, C++, and Perl. It is also possible to create macros to automate commands and the workflow. Images in Gimp consist of layers with adjustable opacity, which provides flexibility and possibilities for different effects. Gimp started development in 1995 and was first released publicly

in 1996. Gimp is released under the GNU General Public Licence (GPL) version 3. (Gimp 2023.)

Another useful open-source tool for designing graphics is Inkscape. Unlike programs that manipulate rasterized images, Inkscape is a vector graphics editor. Inkscape is available for GNU/Linux, Windows, and macOS operating systems and it is distributed under the GNU GPL. The program is free, and it is typically used to design vector-based graphics in for instance webpages, illustrations, interfaces, and diagrams. Inkscape enables users to create and modify shapes by connecting paths and transforming objects. The program can be used for example to create buttons, icons, and logotypes. Layers add flexibility and filters allow users to add effects to the final image. Inkscape uses a file format called Scalable Vector Graphics (SVG), which is based on the Extensible Markup Language (XML). (Hiitola 2012, 1-26.)

Visual Studio Code (VS Code) is a free source code editor developed by Microsoft using open-source technology. Many of the most common programming languages are supported. VS Code includes a feature called IntelliSense that handles syntax highlighting and automatic code completion. The editor has built-in version control and debugging features. Additional functionality can be downloaded as extensions and allow users to customize the editor. Common extensions include support for varying programming languages, formatting tools, A.I. assistants, linting tools, themes, and debuggers. One example is an extension that highlights keywords in the code in different colours. VS code also supports cloud computing in the Microsoft Azure platform. VS Code is designed to be a streamlined editor by default, with a minimalistic approach to the Graphical User Interface (GUI). (Microsoft 2023.)

3.3 Delphi

Contrary to more basic code editors, the Delphi IDE features a full GUI and a visual designer for creating native applications (Embarcadero 2023). Several programming languages and IDEs are used in development at Abilita. The Delphi IDE was used at Abilita for the dental patient information system that the dental chart integrates with. For this reason, the dental chart was programmed using Delphi 10 Seattle, which is a version of Delphi released in 2015. Del-

phi is developed and maintained by a company called Embarcadero. The latest Delphi version is 11.3 Alexandria, and it now also supports development for macOS with the ARM 64bit architecture and iOS through simulation. (Embarcadero 2023.)

The Delphi programming language is based on Object Pascal, and thus supports object-oriented programming. Delphi is a high-level, static, and strongly typed programming language that compiles to different platforms and operating systems. The Delphi IDE is part of RAD Studio which supports Rapid Application Development (RAD) whose strengths include the easy development of GUIs (FIGURE 5). The source file produced by Delphi has the extension .pas, and the code is divided into modules called units. Additional functionality can be installed from a set of packages called components. The main framework for developing graphical user interfaces for Microsoft Windows using Delphi is called the Visual Component Library (VCL). Another popular framework is called Firemonkey and it enables cross-platform app building. (Głowacki 2017, 1-18.)



FIGURE 5. Visual design in the Delphi IDE called RAD Studio 10.2 (McKeeth 2017)

Visual Component Library (VCL) is a set of components provided by the Delphi RAD IDE that allows for quick development of Windows applications. VCL can be used for developing e.g., GUIs, web applications, and database applications. The benefit of using visual components is that the components and their properties can be manipulated at design time, which reduces the amount of coding. Another benefit of VCL is the rapid prototyping of software that it enables. (Embarcadero 2015.)

VCL is an object-oriented framework, and all components descend from the component class. The components offered by VCL can be divided in groups of visual and non-visual classes. Visual components include forms (windows), buttons, labels, and images. These are also called controls, and they inherit from the control class. Control components are typically used for GUI development. Non-visual VCL components are invisible during run time, how-ever, their properties can still be modified during design time. An example of a non-visual component is the data source class used for database application development. Other VCL classes that do not fall into the category of components are also available. Collections of components are called packages and there are many both commercial and free packages available that extend the visual component library. (Embarcadero 2015.)

Like many other IDEs, Delphi allows developers to import code libraries. Both libraries provided by Embarcadero and third-party libraries can be imported. One such library is called Image32, which is a comprehensive 2D graphics library written by Angus Johnson in the Delphi programming language. Image32 offers many functions for image manipulation, for example drawing, filling, rotating, and scaling functions. The filling procedure is called Flood fill and it takes in as arguments an image object, x and y coordinates, a colour, a tolerance value, and a compare function. The Flood fill procedure then starts at the pixel at the given coordinates, checks all adjacent pixels if they match the starting pixel, and colours all matching pixels. If the compared pixels are not matching according to the specified tolerance, the procedure has reached an edge that creates a boundary that prevents further filling. The Draw Line procedure works instead by colouring pixels in a predetermined line of a path object type, which is an array of points. (Johnson 2023a, 2023b.)

3.4 Graphics

This chapter explains concepts that are common in computer graphics. Aliasing is a blocky artifact visible at the edges of objects in low resolutions due to the pixelated structure of raster images. Anti-aliasing is a technique to smooth out these edges by colouring pixels at the

edge with a colour that is an average of the object and the background. (Heying & Long 2023; Niederst Robbins 2006, 527.) Tools that use anti-aliasing to smoothen objects in digital images can often be found in image manipulation software. An example of an image with aliasing and anti-aliasing applied can be found below in figure 6.



FIGURE 6. Aliased edges on the left and anti-aliasing applied on the right (adapted from Adobe 2023; Niederst Robbins 2006, 527)

Vector graphics are based on mathematical equations comprised of points, lines, and curves. The paths created from these elements do not depend on a specific resolution, so they can be scaled freely with persistent image quality. By contrast, raster-based graphics consist of pixels and magnifying a bitmapped image shows pixelated details in a grid-like pattern. Vector graphics on the other hand result in clean lines and curves when magnified. (FIGURE 7.) Vector graphics thus provide flexibility when switching between varying resolutions. (Hiitola 2012, 8-11.)



FIGURE 7. A comparison of a bitmap image on the left and a vector image on the right (adapted from Niederst Robbins 2006, 517)

Colours in computer graphics can be specified by their combination of red, green, and blue values (RGB). Mixing various amounts of RGB components results in a wide range of colours. The RGB colour model is commonly used in for instance in computer monitors, image manipulation software, and web design. For computer graphics, the most common RGB model is the 24-bit true colour. The total number of bits is called the colour depth (or bit depth) and it is defined as bits per pixel (bpp). True colour has an 8-bit hexadecimal value per channel for each pixel, for a total of 24-bits, which equals a total of 2²⁴, or 16,777,216 different colours. Each channel has a decimal value between 0 and 255, or 00 to FF in hexadecimal. For example, the RGB value D4E877 is a hexadecimal representation of a colour with the syntax RRGGBB. In decimal, the values would be 212, 232, and 119 for red, green, and blue. Combining these RGB values creates a light green colour. When used in webpage design, the hash symbol is prepended to the numbers: #D4E877. (Niederst Robbins 2006, 733-741.)

The data recorded at dental examinations are typically presented graphically in a dental chart. When presenting any data graphically, for instance as graphs or plots, the principles of data visualization apply. The main idea is to visualize data effectively by choosing the correct type of visual representation for the dataset. It is also important to keep the chart simple, clean, and informative. Data visualization techniques also help identify trends and patterns through visual discovery. Examples of different types of charts include plots, maps, tables,

pie charts, line charts, and histograms. Line charts, for instance, are well suited for showing change in data. This helps when comparing values, identifying outliers, and spotting relationships. (IBM 2023.)

Abilita uses the Delphi IDE for the development of the systems defined for this development task. The Microsoft Windows operating system was the target platform defined in the requirements, so the VCL framework for visual development was chosen. The Image32 library was chosen for its 2D drawing capabilities and the library was imported into the project. While many image manipulation programs exist, GIMP provides extensive features and is open source. GIMP was thus an easy choice. By combining the knowledge of dental notation, examinations, and charting, with image manipulation, Delphi programming, and graphics, the theoretical foundation for the development task has been laid. With this knowledge presented the description of the development process can begin.

4 DEVELOPING A DENTAL CHART

In this part of the thesis, I will describe how I implemented the concepts described in the previous chapters with the goal to design and develop a dental chart. I started with very basic knowledge of dental notation and the names of tooth surfaces. As the project progressed, I learned more about dental charting and the working methods of dental professionals. Although I was given detailed instructions, I still had plenty of opportunity to make my own design decisions. I had adequate guidance and never felt lack of freedom during development. During the later stages of development, we contacted a registered dentist to get feedback for the dental chart. Based on this feedback, in combination with the concepts of dental charting and previous work done by others, I developed an interactive dental charting program that I believe is both simple and informative.

4.1 Design

The programming language and the IDE had already been decided to be Delphi, whereas I was free to choose the image manipulating software. I decided to design images in GIMP due to its popularity and because it is open source. The first two weeks of development I focused on learning how to program in Delphi, since I had no prior experience of the programming language. I then quickly moved over to designing the images of the teeth for the dental chart.

4.1.1 Dental status chart

I first concentrated on designing the teeth for the dental chart, or the dental status as it is also sometimes called. The basic layout had already been decided, the teeth should be laid out in an oval shaped pattern viewed from the top, as is common in dental charts. The design of the individual teeth, however, was up to me. Since I wanted the teeth to look as correct as possible, I searched online for anatomically correct images of teeth that I could use as a template. Then, I used one of these models to draw an outline of the individual teeth in GIMP. The teeth were drawn with only one to two pixels in width and without anti-aliasing, so that each surface can be evenly filled right up to the edges. Because I deliberately did not apply any anti-aliasing, the teeth have a slightly pixelated appearance. I also divided each tooth into the five surfaces commonly referenced by dentists. (FIGURE 8.)



FIGURE 8. The dental status with five surfaces on each tooth.

Once half of the teeth were drawn, I then simply mirrored the other half because, naturally, the left and right halves are symmetric. Children's temporary teeth are called primary teeth or sometimes also deciduous or milk teeth. The requirements specify that both the permanent and primary teeth should be visible on the same screen, and since the primary teeth are much smaller, they fit inside of the primary teeth as can be seen in figure 9. Typically, only one set of teeth is displayed at a time.



FIGURE 9. Permanent and primary teeth

According to the requirements, the program must have the possibility to fill in each of the five tooth surfaces in up to three different colours. Since each surface must be evenly divisible by both two and three, I divided each surface into six areas and coloured every other area slightly darker. This way, the function that fills in surfaces can either colour the whole surface or several of its parts according to an algorithm I developed. The fill function will be explained in greater detail in a later chapter. The colours of two neighbouring areas are just slightly different shades, their HTML colour codes are #FFFFFF and #FDFDFD. On most monitors the difference is hardly noticeable. To demonstrate the slight difference in colour of the areas, I applied a display filter in GIMP. By adjusting the gamma to the lowest value, the difference is exaggerated. To avoid the possibility that the teeth have slightly noticeable colour differences in each surface, every surface not meant to be coloured can simply be filled white by an algorithm. (FIGURE 10.)

Available Filters → ▲ Activ ☑ Clip Warning → × ☑ Color Deficient Vision ■	ve Filters amma	
Contrast Gamma	~ ~	HX
Gamma Gamma	0.010 🗘 Reset	No to

FIGURE 10. Each tooth surface is divided into six areas

The design of the dental status was graphically limited by the requirements of the fill function. More detailed graphics would have meant that the edges of the teeth would have looked worse after areas had been filled a certain colour. This compromise in graphical quality for objects that need to dynamically change colour seem to be common when looking at solutions by other developers. Either the graphics for the teeth look sharp when not filled with colour, or the tooth is filled evenly. Achieving both, however, seems to be complicated. This observation appears to be true also when looking at previous work on digital dental charting. Judging by the feedback that I received, the graphical quality for the teeth in the dental status chart was adequate, and I could move on to designing the teeth for the periodontal chart.

4.1.2 Periodontal chart

After I had designed the dental status, I started with the periodontal chart. The images for the periodontal chart will also need to be separate for each tooth, so that missing teeth simply can be hidden. Unlike the status view, the images of teeth for the periodontal chart are viewed from the side. This makes sense considering the pocket depth and gingival regression will be shown as a graph over the teeth images. I again used a purchased template and modified the images in GIMP to design the individual teeth both for the upper jaw and the

lower jaw. The two jaws have subtle differences for instance in size and number of roots. Both jaws are then displayed both from the inside, lingual, and from the outside, buccal. Very little differ between images in these two views. Primary teeth are not taken into consideration in this chart.

I divided each tooth into two separate parts, the root, and the crown. The crown has to be a surface with one continuous edge, similar to the teeth in the status, since it also needs the option to be filled any colour. One colour for each crown was enough here, however, so no division of the surfaces was necessary. I designed the teeth to have four different root options: normal root, implant, root canal, and pin (FIGURE 11). This means that tooth images have four different versions, and each version consists of 64 different teeth images (APPEN-DIX 2). Many of these are very similar and in practice duplicates. I have applied anti-aliasing to some edges of the tooth images, which gives them a smoother appearance. The insides of the edges of crowns cannot have anti-aliasing, though. The reason is the same as with the teeth that I designed for the dental status chart, which I explained in the previous chapter. When the crown is filled with a colour, the colour will always fill evenly out to the edges of the whole surface, but only if there are no gradients from smoothed edges.



FIGURE 11. Magnified images of a tooth with four different types of roots

The measurements taken by dentists are given in millimetres, so the graphs in the periodontal chart also need to scale accordingly. Therefore, I created a horizontal grid with a spacing of seven pixels that functions as a background for the images. The gaps between each line correspond to two millimetres. I designed all images so that their sizes match this scale. (FIGURE 12.) In the final stages of design for the periodontal chart, I drew some symbols in GIMP that visualize furcation and mobility. I will explain how I programmed the functionality for the dental chart and periodontal chart in the next chapter.



FIGURE 12. A complete set of teeth in the periodontal chart

4.2 Development

With the design of the graphics completed, I could concentrate fully on programming the functionality of the dental chart. A static dental chart is not very useful, the goal is interactive graphics that visualize treatments according to user input. Since users enter inputs via keyboard, it makes sense to have the string data type as input. The contents of the input string determine the behaviour of the program. In this chapter, I will explain what happens in the program depending on the values parsed from the input string. I will also describe some of the functions used and give examples of algorithms that I created.

4.2.1 Parsing inputs

I began development by writing code that parses the input data. The data input string sent to my module will need to be parsed so that the program knows what to do. Typically, one string corresponds to the treatments or measurements of one tooth meaning that many strings are received by the program during an examination. The input string contains values separated by semicolons, and I wrote parsing functions that then store these values in variables and arrays. I was partially involved in developing the syntax for these values. The FDI notation is used to determine the tooth number, and an S or T determines if the tooth is found on the dental status chart or the periodontal chart. Suppose an input of "S;15;3;ff33aa" was given, then the first letter signifies the type of tooth chart, the second value is the tooth number, the

third value is the surface, and the fourth string of letters and numbers correspond to the RGB colour code. In the example below, a surface has been coloured according to the parsed input "S;15;3;ff33aa" (FIGURE 13).



FIGURE 13. A coloured tooth surface

Suppose instead that an input of "T;1;17;1;d2;b4;m6;" was given. Then the first letter signifies chart type, the second number is the panel number, third is the FDI tooth number, the fourth value is the type of root, and the remaining strings determine the measured depths at each surface. Again, to demonstrate how this works, I have included a figure below where the pocket depth has been drawn according to the parsed input "T;1;17;1;d2;b4;m6;" (FIGURE 14).



FIGURE 14. Pocket depth lines on the periodontal chart

The code for the initial parsing procedure that I wrote is straightforward. Note that functions without return values in Delphi are called procedures. I have hidden some code lines irrelevant to the explanation. The hidden lines are represented by three dots. In code 1 listed below, if the string is empty then the code exits the procedure. If a string is found, then the built-

in split function is called on the string with the semicolon delimiter as argument. The resulting parsed strings are stored in an array and any exceptions are handled. (CODE 1.)

```
procedure TTandvisningForm.ParseAtom(AtomStr: String);
var
  Atom, RetrievedString: String;
begin
  // Check whether a string exists or not
  . . .
  Try
    Atom := RetrievedString;
    // Split the atom. Any existing atom is replaced.
    AtomArray := Atom.Split([';']);
  Except
    on E : Exception do
      ShowMessage (E.ClassName + ' error raised while splitting
     atom with message : '+E.Message);
  End;
end;
```

CODE 1. A parsing procedure

4.2.2 Filling functionality

After the program knows properties such as tooth number and colour, I implemented the colour fill function from the imported 2D library called Image32. The instructions for how to use this function is found in the documentation online. The fill function is also explained in the theoretical background chapter. FloodFill is the name of the procedure, and it is used to fill in the specified colour on a chosen surface on the dental chart. The procedure is also used to fill the crown on a tooth in the periodontal chart. FloodFill works by starting at a pixel, then checking adjacent pixels if they match the starting pixel by comparing colours. Implementing the FloodFill function was a process of trial and error, even though the function itself is simple. The procedure flow goes like this: first the FillTooth procedure is called, which in turn calls the Fill procedure, which then calls the FloodFill procedure (FIGURE 15).



FIGURE 15. The call flow of procedures for colour filling

Before calling the Fill function, however, an image object must be created, and an image loaded onto it. After the fill function has been called, the image object must be destroyed to avoid memory leaks in a procedure called Free. The fill function can be called multiple times depending on how many surfaces are specified in the input string and the number of colours per surface. The fill procedure takes three arguments: the image object, the tooth number, and the surface. (CODE 2.)

```
procedure TTandvisningForm.FillTooth;
var
  ImgFill: TImage32;
 BitmapHandle: HDC;
  Image: TImage;
  I, ToothNumber: integer;
begin
     // Get the specified canvas Timage
    Image := ImageArray[ToothNumber];
    // Retrieve the bitmap from the ImageList
    GetBitmapFromList(Image, ToothNumber, ImageListStatus);
    BitmapHandle := Image.Picture.Bitmap.Canvas.Handle;
    try
      ImgFill := TImage32.Create;
      // The TImage32 object is copied from the HDC
      ImgFill.CopyFromDC(BitmapHandle, Img32.Vector.Rect(0, 0,
     Image.Width, Image.Height));
      // Fills all entered surfaces for a specific tooth
      for I := 1 to Length (AtomSurfaceArray) do
      begin
        if AtomSurfaceArray[I] <> 0 then
        begin
          Fill(ImgFill, ToothNumber, AtomSurfaceArray[I]);
        end;
      end;
```

```
// The filled image is copied back to the canvas
ImgFill.CopyToDc(Image.Picture.Bitmap.Canvas.Handle);
finally
ImgFill.Free;
end;
...
end;
end;
```

CODE 2. The fill tooth procedure

The FloodFill procedure takes five arguments: the image to draw on, the integer coordinates where to begin filling, the RGB colour to fill, and the tolerance. The tolerance determines the sensitivity of the algorithm, or when to detect a boundary that prevents further filling. One example of the procedure call is FloodFill (image, 120, 45, #ffee33, 10). The logic for which surfaces should be filled is shown in the provided procedure (APPENDIX 3). Basically, the procedure is called with different values for the tolerance depending on the number of colours per surface. The fact that I divided each surface into six areas with alternating colour shades allows the algorithm to fill up to three colours per surface (FIGURE 16).



FIGURE 16. The Flood Fill functionality

The solution to filling surfaces of a tooth is simple and sufficiently effective, in my opinion. The result also satisfies the condition to have informative graphics in the dental chart. I did not have access to any source code from competing solutions, but I suspect a very similar approach has been taken by other developers. I received positive feedback from my development supervisor for the functionality of filling teeth with colour on the dental status chart. Consequently, I could move on to develop the functionality for drawing the graphs on the periodontal chart.

4.2.3 Drawing procedures

After I had developed the colour filling functionality for the dental status chart, I began developing the drawing functions for the periodontal chart. This process was also a matter of trial and error. The desired functionality was a result of discussions with a co-worker who had knowledge of periodontal charting. The principle of the final version is simple enough. A matrix of positions is updated according to the inputs and then a line is drawn between each of the updated positions. This is analogous to a graph with vertices connected by edges. Again, I used a procedure from the imported 2D library Image32. The procedure is called DrawLine and it takes five arguments. The first is the image object to draw on, the second is a path object, the third is the pixel thickness of the line, the fourth is the colour, and the final argument is an end style object. An example of this procedure call from the code: DrawLine(Img, Path-BUpperFD, 2, clBlue32, esRound). Paths are simply Point arrays. Points are struct data types with x and y variables representing coordinates. However, in Delphi structs are called records.

I hardcoded the x-coordinates for the path positions since they never change. I named the left, the middle, and the right positions of a tooth after their surface names. Only the y-coordinates are modified by the program based on the user inputs. As with the dental status chart, the input strings are on a per tooth basis, meaning that one input determines the pocket depth only for one tooth at a time. In code 3 listed below, I have again hidden the irrelevant code and chosen to only show the most important lines. First, the code checks for valid inputs and then checks if any inputs exceed the allowed limit, since I did not want any lines to drawn outside of the allowed area. Then the values parsed from input are stored in variables that correspond to each surface. Only three out of six surfaces are shown in this example. If both pocket depth and gum recession are measured, then both will be added to their respective paths. Before storing the y-coordinates in the path, the values in millimetres are mapped to values in pixels instead. (CODE 3.)

```
procedure TTandvisningForm.UpdatePaths;
var
...
procedure CheckLimits;
...
begin
...
```

```
// Check limits
 CheckLimits;
 // Points from the outside (buckal)
 dFD := AtomFDmmArray[0];
 bFD := AtomFDmmArray[1];
 mFD := AtomFDmmArray[2];
 dR := AtomRmmArray[0];
 bR := AtomRmmArray[1];
 mR := AtomRmmArray[2];
     . . .
 if (ValidNumber = True) and (IsLower = False) and (Glob-
     allsP2Active = False) then
 begin
    // P1 Update upper
    PathBUpperFD[Left].Y := MapMmToPx(dFD + dR, False, False);
    PathBUpperFD[Middle].Y := MapMmToPx(bFD + bR, False, False);
    PathBUpperFD[Right].Y := MapMmToPx(mFD + mR, False, False);
 end;
end;
```

CODE 3. The procedure for updating paths

I developed the drawing procedure next. The DrawFD procedure in code 4 below shows what happens after the paths have been updated. First, the tooth number is mapped so it corresponds to the sequential order of the line path. Then an image object is created with the correct size parameters. A line drawing function is called, and it draws lines on the image according to the provided arguments. The parameters are image, path, line thickness, colour, and end style. The line drawing procedure is called several times in the code, each time with different paths. Since the graph image needs to be transparent to show the tooth images in the background, a bitmap canvas is modified, and the image is copied onto the handle of that canvas. Finally, the image object that was created is destroyed to free up its allocated memory. (CODE 4.)

```
procedure TTandvisningForm.DrawFD;
var
Img: TImage32;
MappedGlobalTooth, FillToothNumberFD : Integer;
procedure DrawUpperLines;
begin
DrawLine(Img, PathBUpperFD, 2, clBlue32, esRound);
DrawLine(Img, PathBUpperR, 2, clRed32, esRound);
```

```
DrawLine(Img, PathLUpperFD, 2, clBlue32, esRound);
    DrawLine(Img, PathLUpperR, 2, clRed32, esRound);
  end;
. . .
begin
  MappedGlobalTooth := MapToothNumberFD(GlobalToothNumber);
  if MappedGlobalTooth in [1 .. 16] then
  begin
    if GlobalIsP2Active = False then
      begin
      try
        // Draw lines on upper jaw
        Img := TImage32.Create(ImageFDU.Width, ImageFDU.Height);
        DrawUpperLines;
        ImageFDU.Picture.Bitmap.SetSize(ImageFDU.Width, Im-
     ageFDU.Height);
        ImageFDU.Picture.Bitmap.Canvas.FillRect(ImageFDU.Clien-
     tRect);
        Img.CopyToDc(ImageFDU.Picture.Bitmap.Canvas.Handle);
      finally
        Img.free;
      end;
     . . .
```

CODE 4. The procedure for drawing lines on the periodontal chart

The ability to show pocket depth and gum recession as line graphs on a periodontal chart seems to be a typical solution. Line charts are well suited for comparing values and spotting trends. The points are updated and drawn by the algorithm after each input specifying the measurement on a tooth. After many measurements and inputs, the connected lines drawn by the code now resemble a line chart. In the example below, the blue lines correspond to pocket depth and the red lines are gingival recession after a completed examination (FIG-URE 17). In the figure, one can easily distinguish both the receding gums to the left and the rather deep pockets on several teeth. Tooth number 43 also has an implant with a gold crown in this example. By examining the chart, I think that the development decision to also allow for crowns to be filled with colours is justified. The discussions I had with my supervisor during development also confirm this.



FIGURE 17. The periodontal chart with pocket depth and gingival recession shown as blue and red lines

4.2.4 Chart functions

After having explained the main functionality for drawing the dental status and periodontal chart, I would like to focus on some of the other functions that I developed for the dental charting program. Obviously, explaining every single function in detail is not within the scope of this thesis. Nevertheless, I will include some functionalities that are easily explained and contain only a small amount of code. Some examples include information symbols, a legend of used colours, and a reset function that clears the chart to a default state.

It is not a tall order to implement functionality to have certain things happen after clicking on a tooth with the mouse. Only a few lines of code are required to implement an OnClick event, and in this example the corresponding tooth number is shown in a message after clicking on a tooth image. The message containing the tooth number is also sent to another module. In code 5 shown below, an image object and a tooth number of integer type are declared. The tooth number becomes the value of the tag property defined in the clicked image object. I had set the tag property of every tooth image to the corresponding FDI tooth number during image initialization. Finally, the message containing the tooth number is shown and delivered. (CODE 5.)

```
procedure TTandvisningForm.ImagesClick(Sender: TObject);
var
   ImageClicked: TImage;
   ToothNumber: Integer;
begin
   ImageClicked := Sender as TImage;
   ToothNumber := ImageClicked.Tag;
   //ShowMessage(IntToStr(ToothNumber));
   ...
end;
```

CODE 5. The procedure for clicking on images

Another useful function I developed is called ShowInfo, and it displays a small symbol next to any tooth indicating that there is more information to be found by clicking on it. Any dental treatment or diagnosis not displayed by the dental chart can be explained in words on a separate module by clicking on the information symbol. The inputs can indicate whether they need additional information or not. When the input is parsed and the ShowInfo procedure is called with the tooth number as argument, the correct tooth image object is chosen from an array of images that have already been instantiated (APPENDIX 4). First, the information image objects are created, and they are assigned a parent from the panel component that they belong to. Then the images are made invisible by default, and they are assigned an OnClick event, which is the same one explained previously. Different sizes for the images are assigned depending on tooth placement. Finally, pictures of the information symbols are assigned to the image objects.

```
// Show info box images for the corresponding tooth
procedure TTandvisningForm.ShowInfo(MappedToothNumber, Tooth-
     Number : Integer);
var
  Img : TImage;
begin
  if (MappedToothNumber in [1 .. 52]) then
  begin
    Img := GlobalInfoBoxImageArray[MappedToothNumber];
    // Define positions
    Img.Top := InfoBoxPosArray[MappedToothNumber].Y;
    Img.Left := InfoBoxPosArray[MappedToothNumber].X;
    Img.Tag := ToothNumber;
    Img.Visible := True;
  end;
end;
```

CODE 6. The procedure for displaying information

In code 6 shown above, the correct image is chosen from the array of initialized images. Then the image is moved to the correct position according to the positions defined in an array of positions. The image is assigned a tooth number and finally made visible. (CODE 6.) In figure 18 below, two information symbols indicate that more information is available by clicking on the circles. The functionality is accessed by including the letter "I" in the input, for example by defining the input as "s;15;i;5;ff44ee" for tooth 15. (FIGURE 18.) Note that I have not developed any functionality for user defined inputs used for text explanations. That responsibility falls instead on the developers of other modules that are not directly related to graphical dental charting. My code simply sends the tooth number so the correct module can be referenced. Regardless, the possibility for a visual representation of additional information on the dental chart opens for custom explanations.



FIGURE 18. Clickable information symbols

Towards the end of the project, I developed another feature which is a type of legend or explanatory list of colours for the dental status chart. As usual, the functionality was part of the requirements, and I discussed the feature with my task supervisor before implementing it. I created a legend that can display all previously shown colours along with their description. However, the function does not automatically enter new colours to the legend. This functionality must be added later from another module. I used a VCL component called ColorListBox and the procedure is quite simple. First, the input string is parsed, and the colour and text string is stored in variables. Then, the code calls a procedure called UpdateLegend that simply adds the text and a new colour box object as new entries to the list. (CODE 7.)

```
// Update the Legend
procedure TTandvisningForm.UpdateLegend;
var
  Text : String;
  BoxColor : Integer;
begin
  Text := GlobalLegendMark + ' ' + GlobalLegendExplanation;
```

```
BoxColor := GlobalLegendColor;
ColorListBox.Items.AddObject(text, TObject(BoxColor));
```

end;

CODE 7. The procedure for updating legend entries

In the example shown below, a series of input strings have been received. The first string is "LEGEND;ff2233;(C);Caries". After parsing the input, the code adds the colour and description together with the abbreviation to the list. In the example below, eight strings with different colour codes and explanations have been added to the list. (FIGURE 19.)



FIGURE 19. A colour box component

The feature to translate between Swedish and Finnish was basic and easy to implement. I basically just modified the strings in the textbox properties of the panel components. Another requirement for the dental chart was a separate panel that displays the periodontal history. The code for this second panel is not particularly interesting, since I only copied the panel components and moved them around to display two periodontal charts simultaneously. I did, however, spend considerable amount of time testing this feature. When switching to the periodontal history panel, a string of text can be attached to the input. This text is meant for displaying information regarding for instance the date, place, patient, and dentist. The result is a separate periodontal history panel with the same width dimension as the main panel, which allows for smooth transitions between panels. (APPENDIX 10.)

Suppose a new patient arrives, and the dentist wants to start from an empty dental chart. Instead of simply restarting the program, it would be useful to implement a function that erases everything that have been drawn on the chart. It is also useful to have this feature during development and testing, so that the chart can simply be reset to a default state. I started by adding a VCL button component to the development panel. When the button is pressed, procedures are called that empty the dental chart. These procedures are called ClearStatus and ClearFD, and they can be accessed via normal string input as well.

```
// Clear images on the canvas
procedure TTandvisningForm.ClearStatus;
var
   I: Integer;
begin
   for I := Low(ImageArray) to High(ImageArray) do
   begin
      ImageListStatus.GetBitmap(I - 1, ImageArray[I].Picture.Bit-
      map);
   end;
   // Clear the legend as well
   EmptyLegend;
end;
```

CODE 8. The procedure that clears the status

The clear status procedure begins with a for loop, and inside the loop all tooth pictures are reloaded from the image list (CODE 8). The legend is also emptied; the dental status chart has now been reset to its default state. On the periodontal chart, the clear FD procedure begins with assigning all images to nil (APPENDIX 5). This effectively clears the drawn pixels so that the images become transparent. Then, another procedure is called that resets all paths to zero. After that, all tooth pictures are reloaded from the image lists. The code then does the same for the furcation and mobility symbols. All symbols are hidden by setting the properties that determine visibility to false. Finally, the text boxes are emptied. Now the whole dental chart has been reset to its initial state (APPENDIX 8).

4.3 Testing

I performed testing on the code during development to make sure the code worked as intended. I did not use any unit testing framework for Delphi, instead I wrote temporary code as needed. The most important test was to fill all tooth surfaces with colour (APPENDIX 6). Because the teeth were manually drawn, any disconnected or non-black edges could result in colour leakage when surfaces are filled. Since most images and symbols have hardcoded positions, these also needed to be tested so that no symbol or image had incorrect positions (APPENDIX 7). I simply wrote code containing for loops that filled all teeth surfaces or displayed all symbols. I also manually entered inputs to check for bugs. In the periodontal chart, for instance, I filled all crowns for all the different root options. In the example below, a tiny artifact about three pixels wide can be spotted below the screw of tooth number 44 (FIGURE 20). Mistakes like these are easy to fix, and with thorough testing, the chances of finding them increases dramatically.



FIGURE 20. An artifact found during testing

4.4 Layout and program flow

In this chapter, I will describe the program flow of the dental chart and explain the layout of the dental chart project. Since the Delphi IDE uses a RAD approach suitable for prototyping user interfaces by placing VCL components in a form, it is a convenient way of designing dental charts. Components can easily be moved around and resized during design time in the Delphi form designer, and the result is immediately visible. It is important again to empha-

size that the layout was decided as part of the requirement. I did not create a proper user interface for the dental chart either. The dental chart program I created will integrate with other modules containing for instance a proper UI. Instead, I designed a developer interface with a few buttons on a test panel. The test panel can easily be hidden before the dental chart is deployed, as it is only intended to be used for testing and development. The layout for the dental chart form can be seen in figure 21, with the dental status chart in the centre, the legend on the right, and the periodontal chart in the bottom. Some of these components are visual, such as the tooth images and text boxes. Other components, such as the image lists on the left, are non-visual components, meaning they will not be visible at run time. (FIGURE 21.)



FIGURE 21. The layout of the VCL components on the Delphi form designer

The flow of the program that I developed is somewhat complex, but it can be simplified and explained easily with a flowchart. I think the program flow was quite clear to me during the

early development phase, but as I added more functions the complexity increased. Still, I never encountered situations where I needed to take drastic measures to alter the flow of the program. I want to emphasize that the dental charting program executes code based on the string input, and each input correspond to only one tooth. Accordingly, only one input is entered at a time. If the desired behaviour is to draw a complete dental chart at once, then many string inputs must be executed in series for instance in a for loop.

I will explain the program flow for the main functionality with the help of a flow chart that I made (FIGURE 22). Initially, one input string is received and parsed. The tooth number is then determined. If the desired action is to hide a tooth, then the specified tooth is made invisible in both charts. If the tooth is to remain visible, then the code checks whether to continue execution on the dental status chart or the periodontal chart. The code for the dental status will check for surfaces and colours from the input. These are stored in variables and passed as arguments when the code calls the colour filling procedure. Finally, the graphics are updated. At this final stage, a tooth has been filled with colour and the program returns to the first state and waits for the next input. If the input contains information regarding the periodontal chart, then the first step is to determine the type of root. If the crown needs to be filled with colour, then the code does so on the correct tooth picture. The values for both pocket depth and gum recession are stored in variables for each surface depending on the input. Then the positions for the graph are updated and finally, the graphics are updated. The program is now ready for the next input. (FIGURE 22.)



FIGURE 22. The program flow of the dental charting program

4.5 Results

The development task for this thesis resulted in an interactive dental charting program. I designed roughly 300 individual tooth images and during development I wrote over 3500 lines of code. I implemented a majority of the desired functionality that was realistically achievable. Furthermore, I did so within the time limit. The result of this project is a dental charting program that works independently, although it is designed to be integrated with other modules creating a larger patient information system. The result of the dental charting program is displayed below (FIGURE 23).



FIGURE 23. The result of the development task: a finished dental chart

In figure 23 above, I have made notes in the chart that correspond to what I believe is a rather typical dental examination. The test panel to the right in figure 23 can be ignored, as it is only used during development, and it is not visible after the module have been integrated. The example shows the dental status chart filled with different colours and their meaning. Below that is the periodontal chart that display measurements in blue and red lines, as well as different types of roots. I have also provided an example of an empty chart that has yet to receive any inputs (APPENDIX 8). Another example of inputs given to the dental charting program shows a more detailed examination (APPENDIX 9). In this example, more colours are used, and some tooth surfaces have multiple colours filled in. The periodontal chart at the bottom has more extreme pocket depths and gingival recession that have resulted in periodontitis. This example and the example in figure 19 above show many of the possible use cases for the dental chart. The dental charting program is a tool with many possibilities since any colour can be used and their meaning can be freely modified. This gives developers the opportunity to customize the dental chart according to the customer's practice.

In addition to the functionality listed above, I developed many other features in accordance with the requirements and instructions. The chart can have text displayed in either Swedish or Finnish for example. The dental chart can also be printed together with a freely written text that describes, for instance, the date and place. I also implemented a secondary view of the periodontal chart that opens in another tab and shows two periodontal charts simultaneously (APPENDIX 10). The lower chart can display the dental treatments from a previous visit. A couple of features were de-prioritized during development due to lack of time. Examples include dental bridges and the ability to adjust tooth positions. These features will be discussed in more detail in the next chapter.

During development, I was free to divide my time between designing teeth, writing code, testing, researching, and learning. I estimate that I spent an equal amount of time with each phase. It was fascinating to learn about the practices in the dental profession and implement this knowledge in the software development phase. I also learned the Delphi programming language, which was previously unfamiliar to me. During development, I also had to consider memory management, testing, and debugging. Based on the feedback, I believe that the commissioner is satisfied with my work and the result. The objective for the development task has been met and I will discuss the implications and the assessment of my work as well as the thesis writing process in the next chapter.

5 DISCUSSION

In this chapter I will discuss my approach to the development process, what I learned, key insights, the results, and implications. I will also evaluate my work and then shortly describe the thesis writing process. Finally, I will list a couple of improvements to the dental chart. I have a rather pragmatic approach to software development, and the development process and thesis also reflect this. Many concepts were unfamiliar to me, so I learned plenty during the development process. Things I learned include practices in dentistry, dental charting, the Delphi programming language, image manipulation, to name a few.

The most time-consuming aspects of development involved how to implement the dental charting ideas in Delphi. For instance, I had to plan the data structures and take memory management into consideration. I also thought about the user experience and accessibility. In hindsight, I should have focused on building more structured code. This is important as the project develops and the code increases. I should also have avoided using global variables as much as possible to make the code easier to maintain. For example, writing fewer procedures and more functions would have accomplished this. In addition to the maintainability of the code, I also gave some thought to the program flow and performance. These are all important concepts in the field of information technology. The knowledge I gained from researching, learning, and implementing these concepts will be beneficial to me in the future.

The key insights I obtained from developing the dental chart involves data structures and fundamental properties of data science. For instance, the way I implemented colour filling is not necessarily the most elegant solution. However, since the pixels are the limiting factor, in general, the result of the tooth models tends to look similar regardless of implementation. I can think of other viable solutions, but when a surface must dynamically change colour, a filling function is a good choice. Then the interesting problem of the balance between having smooth edges and an evenly filled surface becomes evident. I believe I avoided this problem by making the design choices that I did. Regarding periodontal charting, it is clear to me that any type of data structure that can represent a line chart is preferable. I chose to implement 2-dimensional arrays for its simplicity. The rest of the features and functionality for the dental chart were by large rudimentary to develop. In summary, I often found the process of combining knowledge from dentistry and software development to be fascinating.

5.1 Assessment

I believe I have successfully completed the task given to me and solved the problem defined in the beginning to a satisfactory degree. My employer was very satisfied with the result. They successfully implemented my dental charting module into their application, and they now can offer a more modern and informative dental charting solution. I believe I have documented the development process and reported my thoughts, insights, and choices sufficiently well in this thesis report. Considering I worked mostly independently, I am also personally satisfied with the result. Especially given the number of unfamiliar concepts that I had to first learn. I was able to perform research and connect the results to develop a complete product. I understand that my solutions are not necessarily the most elegant, but they meet the requirements satisfactorily. Considering I had a limited amount time to develop a working program, I believe my approaches are justified.

The significance of my work is for the commissioner to decide, but I believe that the company can now offer a more competitive product. This thesis was divided into two separate phases: the development phase and the writing phase. The development phase took roughly three months. I successfully completed the development task in time, so my objectives for this thesis have been met. I will be working for Abilita as well in the future, which in a way is also an acknowledgement of my work. By documenting the development process in this thesis, I believe I have successfully demonstrated how to implement a visually distinctive and interactive dental charting program. The problem I defined in the introduction has thus been solved.

5.2 Improvement suggestions

Although I managed to implement most features suggested to me, a couple were still left out due to lack of time. For instance, I did not develop graphics for dental bridges on the status chart or the ability to slightly alter the position of teeth. Although time-consuming, these features are not particularly difficult to develop. I focused instead on the more important features and the quality of already implemented code. If I had more time, I would have also improved the software documentation. I still want to list a few suggestions for improvement, in addition to those already mentioned. These are improvements that I have realized may benefit the final product.

In addition to the keyboard, I think it would be an advantage to allow the mouse to be used as an input device as well. Users could select and click on teeth to fill colours and click on small input boxes to enter pocket depth on the periodontal chart. This would be especially beneficial for trainees who have not yet fully learned the program. An English option for the localization would be trivial to add but was never part of the requirement. Another improvement I propose is to show missing teeth as an outline of the edges of the tooth. This would clear up any confusion whether the tooth is missing or simply not shown. Finally, the biggest improvement to the graphics would be to design vector-based images instead of bitmap images. This would allow the teeth to be scaled without losing detail and make the edges straighter. I researched the possibility to have vector graphics, but I decided it was too unfamiliar and difficult to implement within the time limit. These are just a few suggestions for further research and development that could improve the dental chart and make it even more user-friendly and informative.

6 CONCLUSION

The advantages of digital charting during dental examinations are evident. The purpose of this thesis was to develop a dental chart for Abilita and to document the process and the result. Additionally, I aimed to demonstrate how the chart could be made as clear and informative as possible, without losing graphical fidelity. The development task was limited to graphic design and programming of the chart's functionality, so I did not work on the connection to the database or the end-user interface. I carefully followed the instructions and requirements set by the commissioner. However, my work is based on my own research and trials, including discussions with the company and customers. I worked independently and divided the task into separate phases for development and for writing this thesis.

The chart that I developed has been integrated with the rest of the dental program. The results show that the dental chart is descriptive, informative, visually pleasing, and works as intended. In this thesis I have motivated my decisions and explained what I learned during the design and development phase. Therefore, the objectives defined for this thesis have been met. I have suggested improvements to the commissioner including how the user could interact with the dental chart more easily. For further research, I recommend vector-based graphics, as it would provide a noticeable impact on graphics by improving sharpness while also adding flexibility. The combination of the concepts found in information technology and dental healthcare has resulted in a development task that was both interesting and rewarding.

REFERENCES

Abilita. 2023. *Om oss*. Available at: <u>https://www.abilita.fi/se/om_oss/</u>. Accessed 14 February 2023.

Adobe. 2023. *Aliasing & Anti-aliasing*. Available at: <u>https://helpx.adobe.com/in/photoshop-ele-ments/key-concepts/aliasing-anti-aliasing.html</u>. Accessed 23 March 2023.

David, P. 2023. *Layer Masks.* Available at: <u>https://www.gimp.org/tutorials/Layer_Masks/</u>. Accessed 30 March 2023.

Embarcadero. 2015. *VCL Overview*. Available at: <u>https://docwiki.embarcadero.com/RADStu-dio/Alexandria/en/VCL_Overview</u>. Accessed 3.3.2023.

Embarcadero. 2023. *Delphi IDE*. Available at: <u>https://www.embarcadero.com/products/delphi</u>. Accessed 14 February 2023.

Gimp. 2023. *GNU Image Manipulation Program*. Available at: <u>https://www.gimp.org/</u>. Accessed 16 March 2023.

Głowacki, P. 2017. *Expert Delphi: Robust and fast cross-platform application development*. 1st edition. Birmingham, Mumbai: Packt Publishing.

Heying, P., Long, A. 2023. *Understanding aliasing and anti-aliasing techniques in photography*. Available at: <u>https://www.adobe.com/creativecloud/photography/discover/anti-alias-</u> <u>ing.html</u>. Accessed 30 March 2023.

Hiitola, B. 2012. *Inkscape Beginner's Guide*. 1st edition. Birmingham, Mumbai: Packt Publishing.

Hollins, C. 2013. *Levison's Textbook for Dental Nurses*. 11th edition. Chichester: John Wiley & Sons.

Honkala, S. 2022. *Hampaiden numerointi*. Lääkärikirja Duodecim. Kustannus Oy Duodecim. Available at: <u>https://www.terveyskirjasto.fi/trv00006/hampaiden-numerointi</u>. Accessed 14 February 2023.

IBM, 2023. *What is data visualization*? Available at: <u>https://www.ibm.com/topics/data-visuali-zation</u>. Accessed 30 March 2023.

Johnson, A. 2023a. *Image32 GitHub repository*. Available at: <u>https://github.com/AngusJohn-son/Image32</u> Accessed 7 April 2023.

Johnson, A. 2023b. *FloodFill*. Available at: <u>http://www.angusj.com/delphi/im-age32/Docs/Units/Img32.Extra/Routines/FloodFill.htm</u>. Accessed 7 April 2023.

Lecarme, O., Delvare, K. 2013. *The book of GIMP: A complete guide to nearly everything*. San Francisco: No Starch Press.

McKeeth, J. 2017. *RAD Studio FMX IDE Screenshot*. Available at: <u>https://en.wikipe-dia.org/wiki/File:RAD_Studio_FMX_IDE_Screenshot.png</u>. Accessed 30 March 2023.

Microsoft. 2023. *Visual Studio Code*. Available at: <u>https://code.visualstudio.com/</u>. Accessed 23 March 2023.

Molen, B. 2010. *Periodontal Evaluation Chart*. Available at: <u>https://en.wikipe-dia.org/wiki/File:Periodontal Chart Illustrated.jpg</u>. Accessed 14 February 2023.

NEBDN. 2020. *National Examining Board for Dental Nurses. Dental Charting*. Preston: NEBDN. Available at: <u>https://www.nebdn.org/app/uploads/2020/07/Dental-Charting-V0.5-July-2020.pdf</u>. Accessed 14 February 2023.

Nelson, S., Ash, M. 2010. *Wheeler's Dental Anatomy, Physiology and Occlusion*. 9th edition. St. Louis: Saunders.

Newman, M.G., Takei, H., Klokkevold, P.R. and Carranza, F.A., 2019. *Newman and Carranza's Clinical periodontology*. 13th edition. Philadelphia: Elsevier.

Niederst Robbins, J. 2006. *Web design in a nutshell: A desktop quick reference*. 3rd edition. Sebastopol (California): O'Reilly.

Patient Nar	ne					F	eriode	ontal I 1 st Clir	Evalua nician I	ation (Name	Chart							
st Visit Dat	e	2	2 nd Clir	nician	Name									2 nd Vis	it Date	e		
Upp CAL (Rec+Probe rd Probe Date: CAL (Rec+Probe Ing Probe Date:	er Right →) = Total) = Total	1	2	3	4/A	5/B	6/C	7/D	8/E	9/F	10/G	11/H	12/1	13/J	14	15	16	
Maxillary Facial Declusal Maxillary Palatal/Lingual	10/201 2nd 0 0 0 141 0 0 0 2nd 0 0 0 141 0 0 0 141 0 0 0 2nd 0 0 0 2nd 0 0 0 2nd 0 0 0 2nd 0 0 0																	Occlusion Class I Class II Class II Class II Other Occlusal Facto
CAL (Rec+Probe Probe Date: CAL (Rec+Probe 3 rd Probe Date: CAL (Rec+Probe CAL (Rec+Probe 3 rd Probe Date: CAL (Rec+Probe 2 rd Probe Date: CAL (Rec+Probe	$) = Total$ $) = Total$ $er Right \rightarrow$ $) = Total$ $) = Total$ $) = Total$ $) = Total$	32	31	30	29/T	28/S	27/R	26/Q	25/P	24/0	23/N	22/M	21/L	20/K	19	18	17	Charle On
Mandibular Lingual Occlusal Mandibular Facial	2nd 0 0 0 1st 0 0 0 0 Sub Core 0 0 0 0 1st 0 0 0 0 0 2nd 0 0 0 0 0 2nd 0 0 0 0 0 2nd 0 0 0 0 0																	CRECK ORE NORMAL CASE TYP GINGIVITIS CASE TYP PERICOONTIT CASE TYP PERICOONTIT CASE TYP PERICOONTIT CASE TYP PERICOONT
1 ⁴¹ Probe Date: CAL (Rec+Probe 2 nd Probe Date: CAL (Rec+Probe 3 ¹⁴ Probe Date: CAL (Rec+Probe Comments:) = Total) = Total) = Total																	

APPENDIX 2/1



APPENDIX 2/2



```
procedure TTandvisningForm.Fill(Image: TImage32; ToothNumber: byte;
       Surface: byte);
1641
       I, J: integer;
        // The colors derived from an atom
1643
1644
        Color1, Color2, Color3: TColor32;
1645
        White: TColor32;
      begin
1648
         White := $FFFFFFF;
1649
1650
        Color1 := AtomColorArray[Surface, 1];
        Color2 := AtomColorArray[Surface, 2];
         Color3 := AtomColorArray[Surface, 3];
         J := 1;
        // Fill logic
         if Color3 <> White then
         begin
           // Fill this pattern only for surface 1 on molars and premolars
           if (Surface = 1) and (
             (ToothNumber in [4 .. 8]) or
             (ToothNumber in [12 .. 16]) or
             (ToothNumber in [20 .. 24]) or
             (ToothNumber in [28 .. 32]) or
             (ToothNumber in [36 .. 37]) or
             (ToothNumber in [41 .. 42]) or
             (ToothNumber in [46 .. 47]) or
1670
             (ToothNumber in [51 .. 52])) then
1671
           begin
             for I := 1 to 6 do
             begin
1674
              if (I = 4) then
1675
               begin
1676
               J := 1;
               end;
```

APPENDIX 3/2

```
1678
               FloodFill(Image, ToothArray[ToothNumber, Surface, I].X,
                 ToothArray[ToothNumber, Surface, I].Y, AtomColorArray[Surface, J], 3);
               inc(J);
           begin
             J := 1;
             for I := 1 to 6 do
             begin
               FloodFill(Image, ToothArray[ToothNumber, Surface, I].X,
                 ToothArray[ToothNumber, Surface, I].Y, AtomColorArray[Surface, J], 3);
               if ((I mod 2) = 0) then
                 inc(J);
         else if Color2 <> White then
         begin
           J := 1;
           for I := 1 to 6 do
           begin
1700
             FloodFill(Image, ToothArray[ToothNumber, Surface, I].X,
               ToothArray[ToothNumber, Surface, I].Y, AtomColorArray[Surface, J], 3);
1702
             if (I = 3) then
1703
               J := 2;
1704
         end
1706
         else if Color1 <> White then
1707
         begin
1708
           FloodFill(Image, ToothArray[ToothNumber, Surface, 1].X,
             ToothArray[ToothNumber, Surface, 1].Y, AtomColorArray[Surface, 1], 6);
       end;
```

```
2704
       // Initialize images
2705
       procedure TTandvisningForm.InitializeImages;
2706
2707
         procedure InitializeInfoBoxes;
2708
2709
            I: Integer;
2710
         begin
2711
            // Iniatialize images for InfoBoxes
            for I := 1 to 32 do
2712
2713
            begin
2714
2715
              GlobalInfoBoxImageArray[I] := TImage.Create(Self);
2716
              GlobalInfoBoxImageArray[I].Parent := StatusPnl;
2717
              GlobalInfoBoxImageArray[I].Visible := False;
              GlobalInfoBoxImageArray[I].OnClick := ImagesClick;
2719
2720
              if ((I = 1) \text{ or } (I = 9) \text{ or } (I = 17) \text{ or } (I = 18) \text{ or } (I = 25) \text{ or } (I = 26)) then
2721
              begin
2722
                GlobalInfoBoxImageArray[I].Width := 20;
2723
                GlobalInfoBoxImageArray[I].Height := 20;
2724
                GlobalInfoBoxImageArray[I].Picture.Assign(ImageInfoBox2.Picture);
2725
              end
2726
              else
2727
              begin
2728
                GlobalInfoBoxImageArray[I].Width := 23;
2729
                GlobalInfoBoxImageArray[I].Height := 23;
2730
                GlobalInfoBoxImageArray[I].Picture.Assign(ImageInfoBox.Picture);
2731
              end;
2732
            end;
2733
            for I := 33 to 52 do
2734
2735
            begin
2736
              GlobalInfoBoxImageArray[I] := TImage.Create(Self);
              GlobalInfoBoxImageArray[I].Width := 20;
2737
2738
              GlobalInfoBoxImageArray[I].Height := 20;
              GlobalInfoBoxImageArray[I].Picture.Assign(ImageInfoBox2.Picture);
2739
2740
              GlobalInfoBoxImageArray[I].Parent := StatusPnl;
2741
              GlobalInfoBoxImageArray[I].Visible := False;
2742
              GlobalInfoBoxImageArray[I].OnClick := ImagesClick;
2743
            end;
2744
          end;
2745
         procedure InitializeFurcationImgs;
```

```
procedure TTandvisningForm.ClearFD;
I : Integer;
  // Clear FD lines
  ImageFDU.Picture := nil;
  ImageFDL.Picture := nil;
  ImageFDUP2.Picture := nil;
  ImageFDLP2.Picture := nil;
  ResetPath;
  for I := Low(ImageArrayFD) to High(ImageArrayFD) do
    ImageArrayFD[I].Picture := nil;
    ImageArrayFDL[I].Picture := nil;
    ImageArrayFDP2[I].Picture := nil;
    ImageArrayFDLP2[I].Picture := nil;
    ImageListFDBucc.GetBitmap(I - 1, ImageArrayFD[I].Picture.Bitmap);
    ImageListFDLing.GetBitmap(I - 1, ImageArrayFDL[I].Picture.Bitmap);
    ImageListFDBucc.GetBitmap(I - 1, ImageArrayFDP2[I].Picture.Bitmap);
    ImageListFDLing.GetBitmap(I - 1, ImageArrayFDLP2[I].Picture.Bitmap);
  for I := Low(GlobalFurcImgArray) to High(GlobalFurcImgArray) do
    GlobalFurcImgArray[I].Visible := False;
    GlobalFurcImgArray[I].Picture := Nil;
    GlobalFurcImgArrayP2[I].Visible := False;
    GlobalFurcImgArrayP2[I].Picture := Nil;
  end;
  for I := Low(GlobalMobImgArray) to High(GlobalMobImgArray) do
    GlobalMobImgArray[I].Visible := False;
    GlobalMobImgArray[I].Picture := Nil;
    GlobalMobImgArrayP2[I].Visible := False;
    GlobalMobImgArrayP2[I].Picture := Nil;
  TextLblUpperP1.Caption := '';
  TextLblUpperP2.Caption := '';
  TextLblLowerP1.Caption := '';
  TextLblLowerP2.Caption := '';
```









