2004

# Developing software for wound measurement

Savo Kordic
*Edith Cowan University*

# Edith Cowan University

# Copyright Warning

# DEVELOPING SOFTWARE FOR WOUND MEASUREMENT

A Thesis to be submitted in Partial Fulfilment of the Requirements for the Degree of

**Bachelor of Science (Computer Science) with Honours**

January 2004

By: Savo Kordic
Student ID: 2011361

**Supervisor: Dr Dong Li**

Faculty of Communications, Health and Science
School of Computer and Information Science
Edith Cowan University
Perth, Western Australia

# USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

# ABSTRACT

Chronic wounds such as leg ulcers, pressure ulcers and diabetic ulcers affect many thousands of people in Australia. In addition to the costs of these wounds in terms of human suffering, loss of income and resources, there are costs related to the treatment of ulcers. Thus, there is a genuine need to develop an accurate and a fully objective application for wound measurement. The aim of this project was to create software for the measurement of wounds. In achieving this goal, several issues were addressed: an accurate measurement method capable of detecting small changes in an open wound surface area, a user friendly interface, written in the .NET languages and a relational database in order to keep an accurate visual record of changes in an open wound's surface area.

With an accurate wound measurement system, practitioners will be able to speed wound healing by adjusting treatment according to the total area of ulceration on the affected limb. This project involved the development of a program to measure the surface area of a wound, with the intention of improving the efficiency of measurement.

## *COPYRIGHT AND ACCESS DECLARATION*

*I certify that this thesis does not, to the best of my knowledge and belief:*

i. *incorporate without acknowledgement any material previously submitted for a degree or diploma in any institution of higher education;*
ii. *contain any material previously published or written by another person except where due reference is made in the text; or*
iii. *contain any defamatory material.*

Signature:_____

Date: 15.03.04

# ACKNOWLEDGMENTS

First, I would like to thank my Project Supervisor Dr Dong Li, for his patience and guidance during my work on this project.

I also wish to thank Frances for her constant support and encouragement.

# Table of Contents

# 1 INTRODUCTION

In this section the general background of wound measurement is discussed.

## 1.1 The Background of the Study

Treatment of chronic wounds has become a crucial issue in medical practice. There are enormous costs involved in wound care; therefore, the evaluation of efficacy of these treatments has become a priority. According to the National Institutes of Health (2000) "Not only are non-healing wounds painful, but they are expensive to treat. The cost of wound care is over $3 billion annually."

Chronic wounds are irregular in shape and they are very difficult to measure. Ovingthon (1999) describes wound measurement as "inherently difficult" because wounds are three-dimensional, so they possess area and volume. Despite the wide range of available techniques, "many of the methods commonly used are not entirely accurate and may not capture the full extent of the healing response." Furthermore, Goldman and Salcido (2002, p.236) note that "Assessment of chronic wounds is a complex and broad field ..."

There are many techniques available but they are slow or imprecise. In discussing different tools and techniques available to health care practitioners Goldman and Salcido (2002, p.237) stated that "A reasonable approach to determining wound size during a brief patient encounter is to document the wound's linear measurement - that is, perpendicular linear dimensions". Goldman and Salcido further explain that currently most practitioners measure a wound as a rectangle or an

ellipse. They calculate the area of an ellipse by "measuring 2 perpendicular diameters, such as maximum diameter (major diameter) and maximum diameter perpendicular to the first diameter (minor diameter)." A measuring guide with a centimetre rule, used to ensure that exact dimensions are obtained, is shown in Figure 1.1.

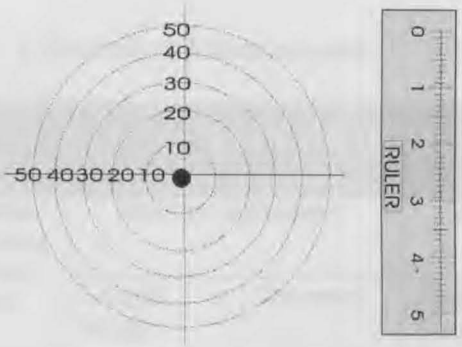



*Figure 1.1* Centimetre rule or a circular diameter guide (Krasner, 1990, p. 121)

Even though this method is simple and relatively cheap (for example, linear distance can be assessed with markings on a scalpel-handled ruler), the method is not precise because it assumes that the wound area can be calculated as a simple shape by measuring in 2 dimensions.

Because of round or irregular wound margins, as well as wound bases that vary in thickness, we need more accurate methods of measurement. In discussing this, Ovingthon (1999) stated that "A more accurate method of determining wound area is tracing of the wound margins onto a clear plastic film." According to Goldman and Salcido (2002, p.237) "Wound outlines are drawn onto a clear plastic sheet with a marking pen." However, this method is regarded as invasive.

It is imperative to document the size and location of existing pressure ulcers. Krasner (1990, p. 120) stated that the location of the existing wound has to be acknowledged. According to Krasner, this should be documented on a diagram of the body, and a number should be assigned for identification purposes (see figure 1.2). The following table (Table 1.1) illustrates what is important to be documented.

Table 1.1 *Records of Measurements*

| RECORD OF MEASUREMENT | | | | | |
|---|---|---|---|---|---|
| **Location** (as marked on body chart) | **Size** (in centimetres) | | **Depth** (in centimetres) | | **Classification** (including colour and pain) |
| | Length x Width | Diameter | Pressure Sore | Undermining (if present) | |
| (A) | | | | | |
| (B) | | | | | |
| .... | | | | | |

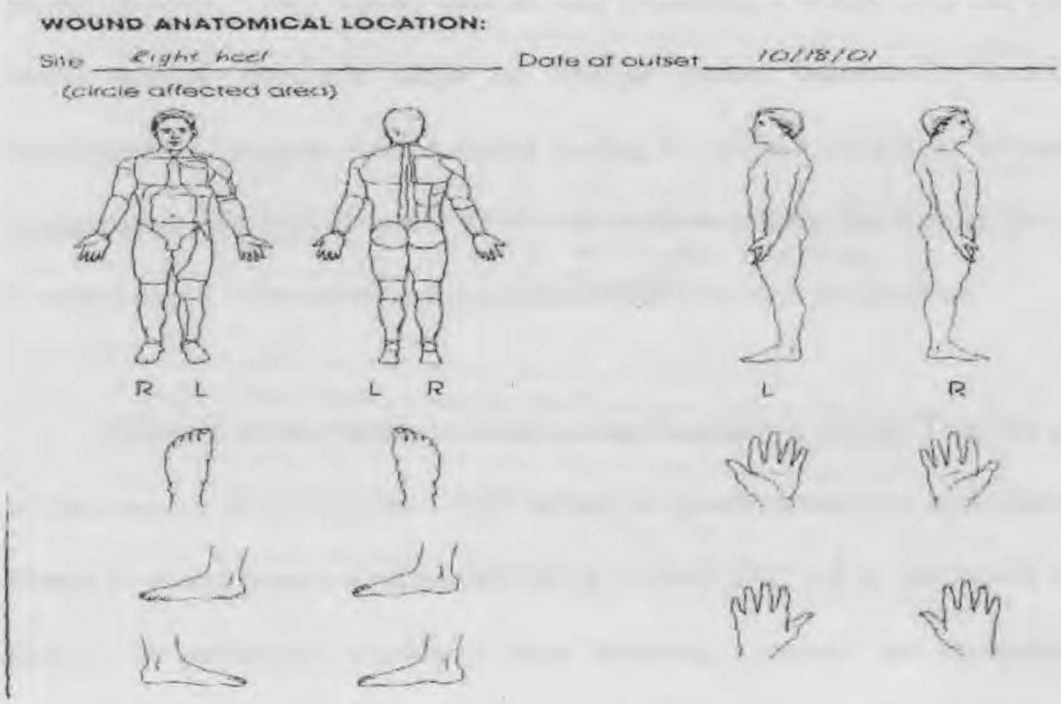



*Figure 1.2* A diagram of the body for the purpose of identifying the locations of wounds. (Krasner, 1990, p. 120).

# 2 REVIEW OF THE LITERATURE

## 2.1 Measurement of Wounds

In order to estimate the progress of wound treatment it is important to measure the skin around the wound and to repeat this at regular intervals. According to Hess (2002, p.91) "The health care professional's thorough mastery of the assessment and documentation components is imperative. " Hess further explains that these skills become "the link to ensure effective management and healing of wounds". Similarly, Goldman and Salcido (2002, p. 237) state that "Simultaneous analysis of the physiological status of a chronic wound provides the practitioner with even more information on which to base a treatment decision for optimal patient prognosis and patient outcome". They further describe that measuring a wound area can be "a useful way to document single or multiple patient outcomes". Therefore, measurement of progress toward wound healing is a critical component of wound management. This study proposes a relational database system that enables the user to extract useful information from the patient records stored in the database.

There are several factors to consider when assessing a wound. Thus, the goal of this research is to determine which variable or group of variables best illustrate change in wound status over time. According to Hess (2002, p.91), factors that may need to be particularly considered when assessing a wound and documenting findings, include classification by degree of tissue layer destruction or colour, and appearance of the wound bed and surrounding skin. Another factor is size,

specifically, length, width, and depth. The length and width of any wounds are measured as linear distances from wound edge to wound edge, and for this, the consistent use of units of measure is essential.

On the other hand, according to Dealey (1994, p.65), factors that may need to be considered include wound classification, the depth of the wound, the shape and size of the wound, the amount of wound exudate, the position of the wound, wound appearance and the environment of care. According to Krasner (1990, p. 121) we should "Never estimate the size or depth of a pressure ulcer". To measure the depth of a pressure ulcer we can use "a sterile, flexible, 6-inch, cotton-tipped applicator" and the following procedure: "With a gloved hand, gently insert the applicator into the deepest portion of the wound." (See Figure 2.1)

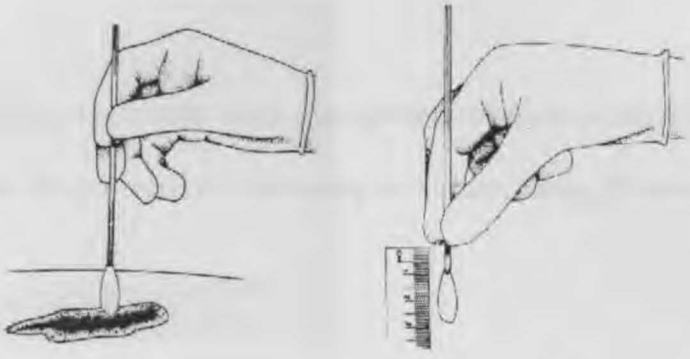


*Figure 2.1* Insert the applicator into the deepest portion of the wound. (Krasner, 1990, p.121)

The next step is to measure from the tip of the applicator to the deepest point, and record the depth in centimetres.

Other methods are available to provide an estimate of the surface area and volume of the wound. According to Krouskop, Baker & Wilson (2002, p.338) "A

technique that has been used clinically to assess wound volume involves filling the wound cavity with a substance such as alginate." There is also a variation of this technique for measuring wound volumes using saline. Basically, the volume of fluid needed to fill the wound is recorded as the volume of the wound. Krouskop, Baker & Wilson (2002, p.338) describe how the contact methods of measuring a wound have several significant problems:

- Potential for disrupting the tissue (when contact is made)
- Risk of contamination of the wound site
- Fluids may be spilled on the bed or clothing (infection can be spread to other patients or clinical staff)
- Failure to account for other information (such as colour)

Therefore, successful ulcer management requires a comprehensive approach that eliminates the potential for damaging or contaminating the wound.

## 2.2 Specific Studies similar to the Current Study

A technique that is similar to the current study is photography. According to Dealay (1994, p.78) "a photograph provides clear evidence of the appearance of a wound and some suggestion of its size". Dealay (1994, p.78) further explains that "when managing chronic wounds, regular photographs can provide real encouragement to both patients and carers". This method also has the advantage that it does not require contact with the wound. Furthermore, according to Krouskop, Baker & Wilson (2002, p.338) "Tissue colour and texture appear to provide clinicians with useful information about the health of the wound."

Even though photographs give good illustration of wound progress, there are some problems associated with photographs. Dealay (1994, p.78) stated that "The depth of a wound is not demonstrated in a photograph as it does not accurately record wounds on curved surfaces, and obviously not all nurses have access to a camera..." Krouskop, Baker & Wilson (2002, p.338) state that "Another limitation of 2-D measurements is that wound healing frequently occurs through changes in depth rather than surface area..." Therefore, healing which results in a reduction of wound depth will not necessarily be accurately represented by measurements taken using 2-D techniques such as flat photographs.

Because of the limitation of this technique, another technique called stereophotography was developed in order to obtain measurements of the volume of a wound. According to Dealay (1994, p.79) this is done by "providing a three-dimensional picture from two photographs taken simultaneously from different angles." Dealay describes that this method is "far superior to other methods such as tracing or photography." However, there are several drawbacks. First, accessibility

(it is highly unlikely that nurses would have access to such equipment). Second, the availability of trained personnel would be necessary. Third, it is time consuming (it takes about 20 minutes to carry out the procedure).

Fortunately, nowadays the availability of digital cameras and powerful personal computers offer the potential to present object colour and texture, so we can distinguish between tissue types. For example, Hess (2002, p.58), describes the first stage of pressure ulcers (based on the recommendations of the National Pressure Ulcer Advisory Panel) "as a defined area of persistent redness in lightly pigmented skin, whereas in darkness tones, the ulcer may appear with persistent red, blue, or purple hues." Therefore, in wound analysis colour image processing could provide additional information that can help determine the healing state of the wound. Figure 2.2 illustrates an ulcer which presented clinically as a deep crater.

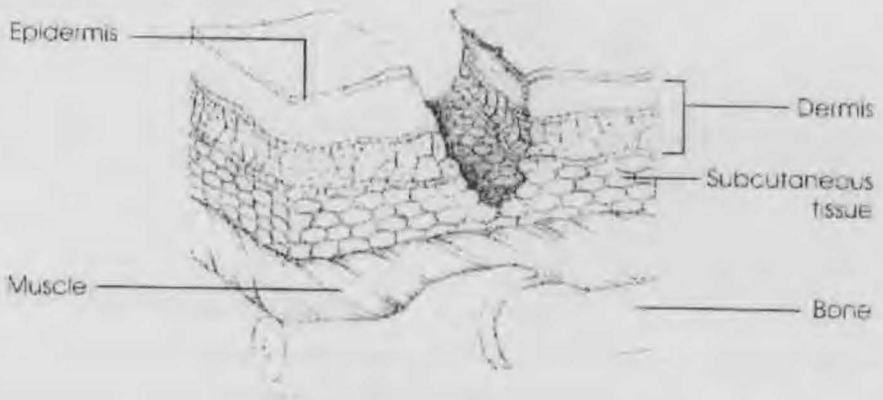



*Figure 2.2* The ulcer as a deep crater – Stage 3 (Hess, 2002, p.59).

Another method involves the use of structured light. According to Dealay (1994, p.79) "Plassman et al. (1993) describe how a camera is connected to an image processing computer which scans the wound which is bathed in light." Dealay further describes that the computer is able to calculate the wound dimensions (the computer

calculates a three dimensional map of the observed view) but, it is not able to "measure deep wounds or wounds which change shape."

Wound measurement is an essential aspect of the evaluation process. According to Sussman (1997, p. 75) "Prediction of pressure ulcer treatment outcomes is essential to clinical decision making and triage, and it provides improved utilisation management to payers and providers". However, the measure of a chronic wound's progress is difficult. Whatever method is used, regular measurement will facilitate some sort of monitoring of the rate of healing. Thomas (1997, p. 86) explains that "An instrument designed to measure healing in pressure ulcers must conform to statistical concepts: validity and reliability". According to Thomas (1997, p. 86) "Validity is the ability of the tool to reflect the outcome it is intended to assess..." and "Reliability is the ability of the same observer or another observer to get the same scores in the absence of real change".

## 2.3 Weaknesses and Strengths in the Methodologies

There are several ways in which measurement of wounds can be made. In discussing different advantages and disadvantages of common measurement methods, Xakelliss and Frantz (1997, p.23) explain that "A review of these studies reveals that each of these methods has inherent strengths and limitations". Table 2.1 shows advantages and disadvantages of common measurement methods.

Table 2.1 *Advantages and Disadvantages of Common Measurement Methods*

Note: 'INVASIVE METHODS' are those which may cause discomfort, pain or distress to the patient, and may disrupt healing tissue, or contaminate a wound site, leading to infection.

| | MEASUREMENT METHOD | | Advantages and Disadvantages |
|---|---|---|---|
| 1 | **SIMPLE MEASUREMENT**<br>(e.g. ruler, probe)<br><br>*INVASIVE* | | **+'s**<br>Useful if wound is a regular shape. A probe may be used to measure depth, especially useful if the wound has a sinus formation.<br>**-'s**<br>This is a very invasive method, and not accurate, especially if necrotic tissue or sloughs are present, as the surface area will appear to vary as debridement progresses.<br>'Sampling error' may occur as different people make the measurements over time.<br>Overall wound appearance is not recorded.<br>(Dealey, 1994, p.76) |

| | | | |
|---|---|---|---|
| 2 | **TRACING**<br>(eg. acetate paper, Kundin measuring tool)<br><br>*INVASIVE* | | **+'s**<br>The apparent surface area of the wound can be calculated with varying degrees of accuracy. Best used on simple, shallow wounds.<br><br>**-'s**<br>This is a very invasive method, and not accurate, especially if necrotic tissue or slough is present, as the surface area will appear to vary as debridement progresses.<br><br>'Sampling error' may occur as different people make the measurements over time.<br><br>Depth is not measured.<br><br>Overall wound appearance is not recorded.<br><br>(Dealey, 1994, p.77)<br><br>**+'s**<br>The Kundin measuring tool provides an estimate of wound surface area and volume, based on assumptions about the geometry of a typical wound.<br><br>**-'s**<br>This is an invasive method.<br><br>(Krouskop, Baker, & Wilson, 2002, p.338). |
| 3 | **PHOTOGRAPHY**<br><br>*NON-INVASIVE* | | **+'s**<br>If some kind of scale is included in the photograph, this is a good, non-invasive method of recording wound appearance, and apparent surface area. |

| | | | |
|---|---|---|---|
| | **PHOTOGRAPHY** cont... | | -'s<br>Depth is not measured.<br><br>Photographic equipment is not always available in hospitals.<br><br>Care must be taken to ensure that the same angle and distance are used each time a wound is photographed, so that consistency in image size is maintained over time.<br><br>This method takes longer than (1) and (2).<br><br>(Dealey, 1994, p.78) |
| 4 | **STEREOPHOTOGRAMMETRY**<br><br>*NON-INVASIVE* | | +'s<br>Volume is measured in a three-dimensional picture composed of two photographs taken simultaneously from different angles. Can be used in conjunction with computerised image analysis. This is a non-invasive method.<br><br>-'s<br>Such equipment is not always available in hospitals.<br><br>Care must be taken to ensure that the same angle and distance are used each time a wound is photographed, so that consistency in image size is maintained over time.<br><br>This method takes longer than (1), (2) and (3).<br><br>(Dealey, 1994, p.79)<br><br>-'s<br>Distortions can occur when a three dimensional image is projected onto a two dimensional surface such as a photographic print.<br><br>(Krouskop, Baker, & Wilson, 2002, p.338). |

| | | | |
|---|---|---|---|
| 5 | **MOLDING**<br><br>*INVASIVE* | | **+'s**<br>A mold is cast by filling the wound cavity with a substance such as alginate, and from this the volume of the wound can be calculated.<br><br>**-'s**<br>This is an invasive method, and does not record the appearance of the wound.<br><br>(Krouskop, Baker, & Wilson, 2002, p.338). |
| 6 | **SALINE INJECTION**<br><br>*INVASIVE* | | **+'s**<br>The volume of the wound is calculated by measuring the quantity of saline required to fill the wound cavity. This method is less likely to damage the wound site than (5).<br><br>**-'s**<br>This is still an invasive method, and does not record the appearance of the wound.<br><br>(Krouskop, Baker, & Wilson, 2002, p.338). |
| 7 | **COMPUTERISED SYSTEMS OF WOUND MEASUREMENT**<br><br>*NON-INVASIVE*<br><br>Dealey, 1994, p.79, describes some computerised systems using:<br><br>• *digital image analysis using a video attached to a computer, and*<br>• *the use of structured light to calculate wound dimensions* | | **+'s**<br>Automatically measures parameters. Provides texture analysis with 3-D-structured light-volume. Can be used clinically.<br><br>**-'s**<br>A practitioner is required to manually delineate the boundaries of the wound and the boundaries of different tissue types within the wound.<br><br>(Krouskop, Baker, & Wilson, 2002, p.339). |

## 2.4 Summary

The use of *invasive methods* such as some of those described above, is undesirable due to obvious reasons of hygiene, patient comfort, and due to the weakness of healing tissue. Invasive methods such as simple measurement (1) and tracing (2) do not record the appearance of a wound, and are limited in their accuracy because they rely on manual measurement by the practitioner. Invasive methods such as molding (5) and saline injection (6) also do not record the appearance of a wound, and carry a high risk of contamination and of damaging the wound site.

Of the *non invasive* methods described above, photography (3) and stereophotogrammetry (4) are considered the most time consuming, and both have some problems with inconsistency due to positioning of cameras. Computerised systems (7) provide texture analysis with use of structured light, but a practitioner is required to manually delineate the boundaries of the wound.

**It is apparent that current measurement methods fail to demonstrate an ability to provide both accurate and minimally invasive methods for the measurement of wound.**

The ideal method would be non invasive, accurate, consistent, efficient, and easy to use.

## 2.5 Other Literature of Significance to this study

In developing a new system to addresses the complex requirements for wound measurement, the Visual Basic .NET programming language has been chosen. This section is a review of literature about this language, including issues such Common Language Runtime (CLR), Visual Basic .NET class libraries, data types, Visual Basic operators, control structures, the Visual Basic .NET exception handling, the objects required to connect a Visual Basic .NET application to a database, and the Visual Basic .NET graphics contexts and graphics objects.

### 2.5.1 Introduction

According to Sebesta, (2002, p. 66) "BASIC (Beginner's All-purpose Symbolic Instruction Code) was designed at Dartmouth College (now Dartmouth University) in New Hampshire by two mathematicians, John Kemedy and Thomas Kurtz, ..." He further explains how the original version of BASIC was very small and had only several different statement types. According to Sebesta (2002, p.68) "Visual Basic is based on QuickBASIC but is designed for developing software systems that have windowed user interfaces." The following illustrates genealogy of VISUALBASIC:

BASIC (1964) ----------→ QuickBASIC (1988) ---→ Visual BASIC (1990)

(Sebesta, 2002, p. 69)

**The Different Versions of Visual Basic**

According to Schneider (2003, p. 22) "Visual Basic 1.0 first appeared in 1991. It was followed by version 2.0 in 1992, version 3.0 in 1993, version 4.0 in 1995, version 5.0 in 1997, and version 6.0 in 1998. VB.NET, released in February 2002, ..."

Visual Basic .NET is an event-driven, visual programming language that is a significant different approach from Visual Basic 6.0 (previous version). According to Schneider (2003, p. 22) Visual Basic .NET "is not backward compatible with the earlier versions of Visual Basic."

### 2.5.2 About Microsoft .NET

This section is about Microsoft .NET. The .NET platform includes the common language runtime (**CLR**) and the .NET Framework class library (**FCL**).

**Common Language Runtime (CLR)**

According to Prosise (2002, p.3) "The .NET Framework is a platform for building and running .NET applications." Prosise explains that the framework is very useful because it provides a common API for all .NET languages (such as Visual Basic .NET, Visual C++ .NET, C# and others). In addition to this, the .NET architecture can exist on multiple platforms. According to Prosise (2002, p.6) "Every byte of code that you write for the framework either runs in the **CLR** or is given permission by the CLR to run outside CLR." Prosise further describes that the CLR provides "a virtual environment for hosting managed applications.", thus it executes Visual Basic

.NET programs. Programs are compiled into native machine code in two steps. First, the program is compiled into language called Common Intermediate Language (CIL). Then, according to Prosise (2002, p.6) "CIL instructions are just-in-time (JIT) compiled into native machine code (typically x86 code) at run time." Therefore, another compiler (the JIT compiler) in the CLR translates the CIL into machine code.

In discussing the benefits of running code in the managed environment of the CRL, Prosise states that "For starters, as the JIT compiler coverts CIL instructions into native code, it enacts a code verification process that ensures the code is type safe." Prosise (2002, p.7) further describes that "It's practically impossible to execute an instruction to access memory that the instruction isn't authorized to access." Hence, it is extremely difficult to write a code that can cause harm on the host system. Secondly, Prosise ((2002, p.7) also states that "Another benefit of running in a managed environment comes from the fact that resources allocated by managed code are garbage collected." Prosise further explains that "you allocate memory, but don't free it; the system frees it for you." Thus, garbage collection (by performing memory management internally) improves performance. However, depending on the automatic garbage collection is not always the best way to manage resources. According to Deitel, Deitel, & Nieto (2002, p. 326) "Certain resources, such as network connections, database connections and file streams, are better handled explicitly by the programmer."

**.NET Applications**

.NET applications are composed of three primary entities. According to Conard, Dengler, Francis, Harvey, Hollis, Ramachandran, Schenken, Short and Ullman

(2000, p. 37), the first is an assembly, which is a primary unit of deployment of a .NET application. It is composed of a *manifest* and one or more modules. The manifest contains information about the identity of the assembly. The second entity is a set *of modules*, which are the individual files that make up an assembly. The third is a *type*, which is a basic unit of encapsulating data with a set of behaviours. According to Prosise (2002, p.14) "Multiple assemblies are commonly used to glue together modules written in different languages and to combine managed modules with ordinary files containing JPEGs and other resources." Multifile assembly that consists of three managed modules is show in Figure 2.3 Note that *metadata* is essential because it fully describes a module and its external dependencies (it informs what types are present in each managed module).

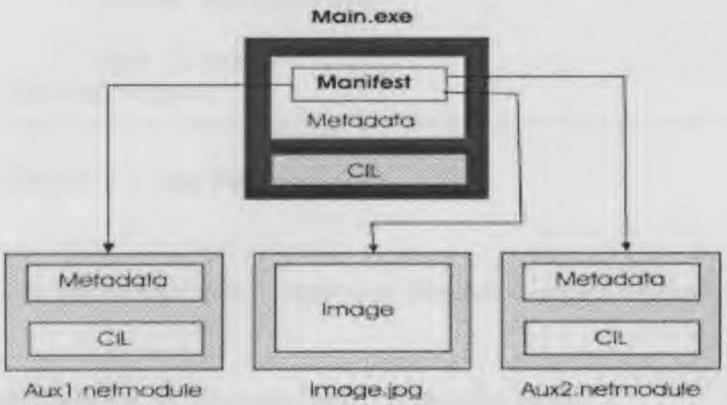



*Figure 2.3* Multifile assembly, (Prosise, 2002, p.16)

**The .NET Framework Class Library**

.NET is a new approach to software development. It includes a class library with more functionality than any other software package. All languages based on the .NET framework have .NET Framework Base classes available. Most of them reside

in a *namespace* called System. For example, the System.Math.Sqrt is a method in the framework classes.

According to Petzold (2003, p.11) "The namespace concept helps to ensure that all names used in a particular program or project are unique." Petzold (2003, p.7) describes that this concept is vital in a case of class libraries (see Figure 2.4) that have name conflicts (for example, both libraries contain a class named *WoundVolume* that is implemented entirely differently in each DLL).

```
Namespace WoundMeasurement.WoundLibrary
      Class WoundVolume
      ...
      End Class
End Namespace
```

```
Namespace WoundAssessment.MeasurementLibrary
      Class WoundVolume
      ...
      End Class
End Namespace
```

*Figure 2.4* .Net Namespaces

So you can then refer to the particular *WoundVolume* class using the name:

WoundMeasurement.WoundLibrary.WoundVolume

**Or**

WoundAssesment.MeasurementLibrary.WoundVolume

The primary goal of the .NET platform is to enable programmers to create applications easily, using such prepackaged components as buttons and textboxes. There are a number of these framework classes that will affect the outcome of the research questions. These are as follows:

- System.Drawing.Graphics – used to draw graphic elements on the form surface
- System.Drawing.Drawing2d – adds capability for two-dimensional vector graphics
- System.Drawing.Imaging – namespace that includes functions to work with various image formats. According to Conard, Dengler, Francis, Harvey, Hollis, Ramachandran, Schenken, Short and Ullman (2000, p. 181) some of the supported formats include:
    - BMP Windows bitmap image format
    - EMF Enhanced Windows metafile image format
    - GIF Graphics Interchange Format
    - JPEG JPEG image format
    - PNG Portable Network Graphics format
    - TIFF Tag Image File Format
    - WMF Windows metafile image format

### 2.5.3 Visual Basic .NET Data Types

Because the differences between types can alter the program behaviour, it is important to understand Visual Basic .NET data types. According to Deitel, Deitel, & Nieto (2002, p. 198) all Visual Basic .NET data types can be categorized as either:

- Value types
- Reference types (contains a location in memory where data is stored)

The Common Language Specification (CLS) as implemented in the .NET Framework is shown in Fig 2.5.1.

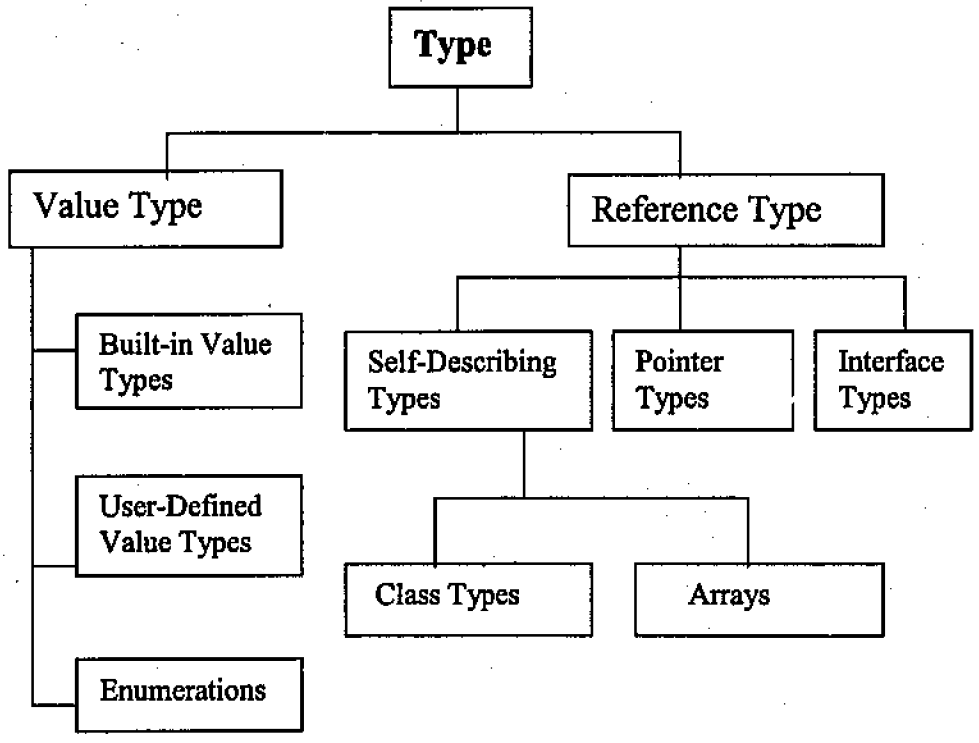



*Figure 2.5.1* Common Type System (based on Evjen & Beres, 2002, p.10).

Refer to Appendices A, B, C and D, for further information about Visual Basic .NET data type changes, built-in data types, and operators.

| | |
|---|---|
| **Appendix A** | describes data type changes in Visual Basic .NET. |
| **Appendix B** | contains more information about Visual Basic .NET value types (built-in types). |
| **Appendix C** | contains more information about Visual Basic .NET reference types (built-in types). |
| **Appendix D** | describes the Visual Basic .NET operators. |

The *value types* that can be constructed by the programmer include *Structures* and *Enumerations.*

**Structures**

You can create a user-defined type by using the Structure statement. For example, the following declaration creates a user-defined data type named Student that can store the name, date of birth, and grade associated with a student:

```
Structure Student
      Dim Name As String
      Dim Surname As String
      Dim Grade As Integer
End Structure
```

*Figure 2.5.2* Structure Student

**Enumeration**

This example uses the *Enum* statement to define a set of named constants. In this case, the constants are students' grades.

```
Enum StudentGrade
      HD = 80
      D = 70
      CR = 60
End Enum
```

*Figure 2.5.3* Enumeration Student Grade

The *reference types* that can be created by the programmer include the following:

- Classes
- Interfaces
- Delegates

**Classes**

Classes are reference types that contain data (constants and fields) and sets of methods that manipulate data. Deitel, Deitel, & Nieto (2002, p. 299) stated that "Classes in Visual Basic fasciate the creation of special data types, called *abstract data types* (ADT), which hide their implementation from clients." In Visual Basic .NET classes can contain the following members:

- Fields (analogous to member variables in C++)
- Methods (analogous to member function in C++)
- Properties (implemented using accessor /get and set/ methods )
- Events (the Event statement to declare the event)

(Prosise, 2002, p.28)

Even though fields and properties both store and retrieve information in an object, according to Petzold (2003, p.27) "The property has the advantage over a field of being able to perform validity checks."

The following example demonstrates class CWound (see Fig. 2.5.4). Class CWound contains the information needed to represent a wound area. In Fig. 2.5.4 lines 1-2 begin the CWound class definition, indicating that the class CWound inherits from class Object (in the Visual Basic .NET, Object is the universal data type). The Inherits keyword (line 2) indicates that class CWound inherits existing fields, properties methods and events of class Object (of namespace System).

```
 1 Class CWound
 2    Inherits Object
 3
 4    ' Fields - declare Integer values for height and width
 5    Private mHeight As Integer = 0
 6    Private mWidth As Integer = 0
 7    Private count As Integer
 8
 9    ' Method New is CWound constructor
10    ' initialize count instance variable to zero
11    Public Sub New()
12      count = 0
13    End Sub
14
16   ' Properties
17   Public Property Width() As Integer
18      ' return mWidth value
19      Get
20         Return mWidth
21      End Get
22
23      ' set mWidth value
24      Set(ByVal value As Integer)
25         If (value >= 1 AndAlso value < 761) Then
26            mWidth = value
27         Else
28            mWidth = 0
29         End If
30      End Set
31   End Property
32 End Class
```

*Figure 2.5.4* CWound Class

Note that while there is no restriction on the use of public entities (declared with the *Public* member access modifier), a private entity (declared with the *Private* modifier) is accessible within its declaration context. CWound class has five class members: three fields (line 5-7), one property (line 17-31) and one special method called 'constructor'. Essentially, the programmer writes a code for the constructors that are called each time an instance of the class is created. The CWound constructor (line 11-13) initializes the count to 0. According to Deitel, Deitel, & Nieto (2002, p. 314) "Each property contains a **Get** accessor (to retrieve the field value) and **Set** accessor (to modify the field value)." The Get accessor (line 19-21) returns the appropriate instance variable's value (mWidth). The Set accessor (line 24-30) controls the setting of the variable mWidth. If the value is not between 1 and 760 then it will be set to zero.

**Interfaces**

Prosise (2002, p.28) states that "An interface is a group of zero or more abstract methods – methods that have no default implementation but they are to be implemented in a class or struct." Interfaces are defined with the keyword *Interface.* Each interface contains a list of *Public* methods and properties. In the following example (see Fig. 2.5.5), the interface ISize returns the size information for the class CWound. The definition of the interface ISize begins at line 1 and ends at line 4 with the keyword *End Interface.*

```
 1  Public Interface ISize
 2    Property Width as Integer
 3    Property Height as Integer
 4  End Interface
 5
 6
 7  Class CWound
 8    Implements ISize
 9
10    ' Fields - declare Integer values for height and width
11    Private mHeight As Integer 'm for members
12    Private mWidth As Integer
13    ....
14     ' property Width implementation of interface ISize
15    Property Width() As Integer Implements ISize.Width
16      Get
17         Return mWidth
18      End Get
19    End Property
20
21    ' property Height implementation of interface ISize
22    Property Height() As Integer Implements ISize.Height
23      Get
24         Return mHeight * 2
25      End Get
26    End Property
27  End Class
```

*Figure 2.5.5* .ISize Interface

A class that is to implement an interface must specify that it *Implements* the interface (line 8). Class CWound has member variables mHeight and mWidth (lines 11-12). Because Cwound implements interface ISize, it must provide concrete implementations of properties Width (defined on line 15-19) and Height (line 22-26).

**Delegates**

Microsoft basic .NET provides delegates which are classes (encapsulate a set of references) that contain method references that can be passed to another method. Delegates in Visual Basic .NET are provided for dealing with events, such as a mouse click. Because the sender class (in event communication) does not know which object will receive the events it raises, a delegate acts as a type-safe function

pointer. As mentioned above events are based on the delegate model. Fig. 2.5.6 illustrates the event handling model using delegates.

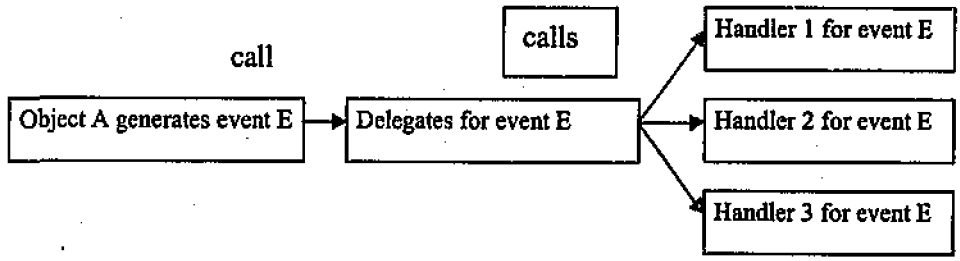



*Figure 2.5.6* .Event-handling model (Deitel, Deitel, & Nieto, 2002, p. 480)

According to Deitel, Deitel, & Nieto (2002, p. 433) "Rather than send a method reference directly, an object can send the delegate instance, which contains the reference of the method that we would like to send". They describe that delegates can be either:

- *Singlecast delegates* (delegates containing a single method and are created or derived from class *Delegate*)
- *Multicast delegates* (delegates containing multiple methods and are derived from class *MulticastDelegate)*

Class CDelegateLecturer (Fig. 2.5.7) uses delegates to handle the tutor's marking event. Lines 2-4 provide the declaration for delegate Marking. Delegate Marking defines a method that receives one Integer (Mark) and one String (Name).
In the class CDelegateLecturer the Marked event is a delegate of type Marking (line 6). Class CTutor (line 9-20) on the other hand, needs to handle the lecturer's request for marking.

In the declaration section of class CTutor (line 10), the WithEvents keyword is used to declare the object variable (for the class that is used with the handles clause in event handler, line 18 – Handles Lecturer.Marked). Lines 16-20 implement method *HandleMarked* that handle the **Marked** event.

```
 1.  Public Class CDelegateLecturer
 2.       Public Delegate Sub Marking( _
 3.           ByVal Mark As Integer, _
 4.           ByVal Name As String)
 5.
 6.       Event Marked As Marking
 7.  End Class
 8.
 9.  Public Class CTutor
10.       Public WithEvents Lecturer As CDelegateLecturer
11.
12.     Sub New()
13.           Lecturer = New CDelegateLecturer()
14.     End Sub
15.
16.     Private Sub HandleMarked( _
17.         ByVal Mark As Integer, _
18.         ByVal Name As String) Handles Lecturer.Marked
19.         'mark assignements code
20.     End Sub
20.  End Class
```

*Figure 2.5.7* .Using Delegates

### 2.5.4 Control Structure

Visual Basic .NET language has 11 control structures. These are as follows:

- ***Sequence***
- Three types of ***Selection***
- Seven types of ***Repetition***.

A program can be formed by combining as many types of control structure as necessary.

| **Appendix E** contains more information about Visual Basic .NET control structure. |
|---|

### 2.5.5 Exception Handling

According to Deitel, Deitel, & Nieto (2002, p. 442) "An exception is an indication of a problem that occurs during a program's execution." They explain how exception handling enables the programmer to handle exception (detects and protects blocks of code that have the potential to raise errors). Even though Visual Basic .NET continues to provide Visual Basic 6 error-handling keywords to identify runtime errors (such as On Error Goto, Resume Next and Err.Description property), the **Try/Catch** block is a new mechanism for exception handling. The following code shows the structure of a Try..Catch statement:

```
Dim Result, x, y As Integer
x = 5
y = 0
Try
    Result = x \ y
    ' attempted to divide by zero
Catch exc As Exception
    Console.WriteLine(exc)
End Try
```

*Figure 2.6* .Try/Catch block and DivideByZeroException

The code in Fig. 2.6 uses Try and Catch block to detect the exception (attempt to divide by zero) and then displays the block of text describing the error (that is: Attempted to divide by zero).

**Appendix F** contains more information about Exception handling.

### 2.5.6 Displaying Data Stored in a Database

**The connection**

The first step is to declare a connection object. The class you use to represent a connection to a datasource is OleDbConnection of namespace **System.Data.Oledb** (see figure 2.7.1)

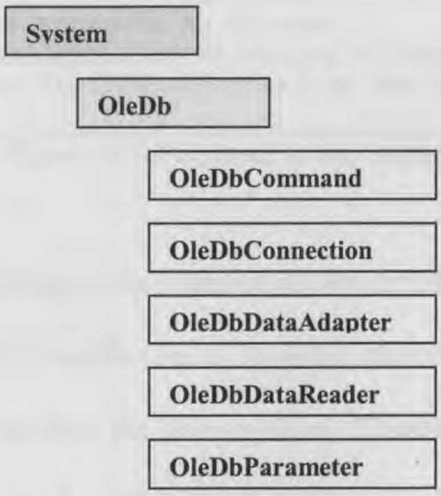



*Figure 2.7.1* OleDb Managed-Provider Namespace (Microsoft Corporation, 2003).

The declaration of a Connection object "OleDbConnenction1" is given below:

```
Imports System.Data.OleDb 'namespace
...
...
        'the declaration of a Connection object
        Dim OleDbConnection1 As OleDbConnection
```

*Figure 2.7.2* Connection object

The next step is to transfer information from the database to a dataset. For this project Microsoft Access Database will be used. Microsoft Jet 4.0 OLE DB Provider, is the **driver** (provide specific information about the protocol of communication) for the Access databases.

The following lines of code illustrate how to use the DataAdapter (using OleDbDataAdapter) object in order to connect to the database WOUNDS.MDB:

```
1.  Dim OleDbConnection1 As OleDbConnection
2.  Dim ConnStr As String = "Provider=Microsoft.Jet.OLEDB.4.0; " & _
3.              "Data source = c:\wound measurement\dbwound.mdb"
4.  Dim DataSet1 As DataSet
5.  Dim SqlString As String = "SELECT * FROM WOUNDS"
6.  Dim OleDbDataAdapter1 As New OleDbDataAdapter(SqlString,ConnStr)
```

*Figure 2.7.3* Connect to the database

Line 1 declares the Connection object "OleDbConenction1" for this program. Property ConnStr (line 2-3) specify the driver and path to the Access database file. Line 4 declares the Dataset object "DataSet1" to populate Dataset with data from a datasource. Line 5 in Figure 2.7.3 demonstrates how to use the SQL select statement to select the entire contents of the WOUNDS table. Finally, line 6 creates the instance of OleDbDataAdapeter "OleDbDataAdapter1" and passed the connection string (ConnStr), and the query string "SqlString" that is a SQL SELECT statement.

**Using the DataAdapter Object to access data**

The **DataAdapter** () object is then used to fill the **DataSet** and update the data source. According to Salvage (2003, p. 413) the syntax required to load data into a data set is as follows:

*DataAdapterName*.Fill(*DataSetName*)

**Example**

The following example (see Fig. 2.7.4) uses an **OleDbConnection** and an **OleDbDataAdapter**. The OleDbConnection is opened (Line 3) and then tries to fill the dataset (dataSet1) through the OleDbDataAdapter1 (Line 5). The next two lines (line 6 and 7) show how to catch and throw the exception (the catch block catches fillException derives from System.Exception), and finally the connection (Line 10) is closed whether or not the exception was thrown.

```
1.      Try
2.          'Open the connection.
3.          Me.OleDbConnection1.Open()
4.          'Attempt to fill the dataset
5.          Me.OleDbDataAdapter1.Fill(dataSet1)
6.      Catch fillException As System.Exception
7.          Throw fillException
8.      Finally
9.         'Close the connection
10.         Me.OleDbConnection1.Close()
11.     End Try
```

*Figure 2.7.4* Coding the Loading of a Data Set

### 2.5.7 Visual Basic .NET graphics contexts and graphics objects

This section explains the fundamentals of graphics programming in Visual Basic .NET. It also includes discussions of Visual Basic .NET coordinates, controls that can contain graphics, and other issues that allow programmers to improve their graphics context.

Advanced graphics methods described in this section reside in the System.Drawing namespaces. Figure 2.8.1 illustrates a portion of the Sytem.Drawing class hierarchy:

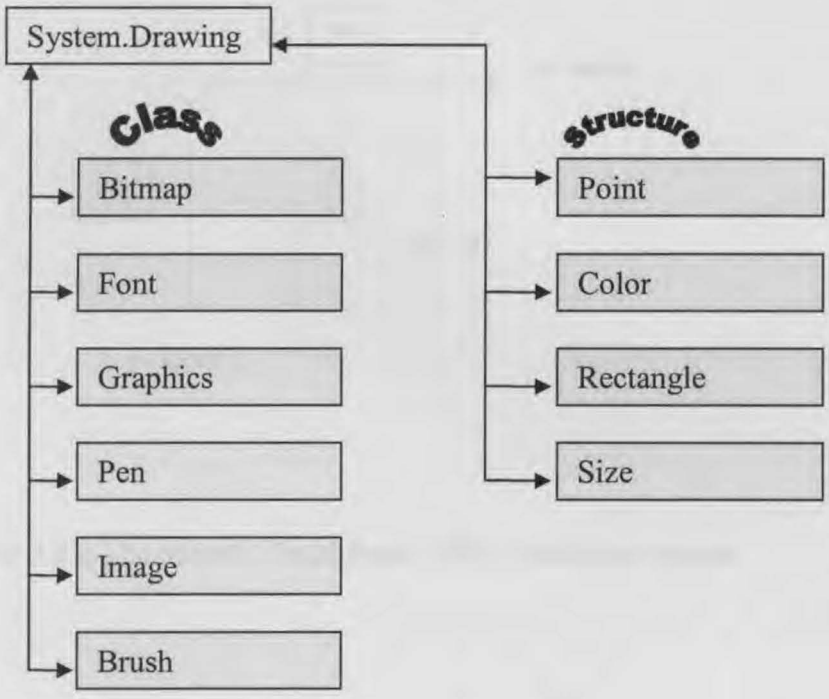



*Figure 2.8.1* System.Drawing Classes and Structures (Deitel, Deitel, & Nieto, 2002, p. 685)

**Visual Basic Coordinate Systems**

It is important to understand how Visual Basic .NET measures drawing coordinates. According to Deitel, Deitel, & Nieto (2002, p. 685) "coordinate units are measures in pixels ('picture element'), which are the smallest units of resolution on a display monitor" (in contrast, Visual Basic 6 uses the vbTwips mode).

By default, the *x-coordinate* represents the horizontal distance from the upper-left corner and the *y-coordinate* is the vertical distance from the upper-left corner. Figure 2.8.2 illustrates the default Visual Basic .NET coordinate system.

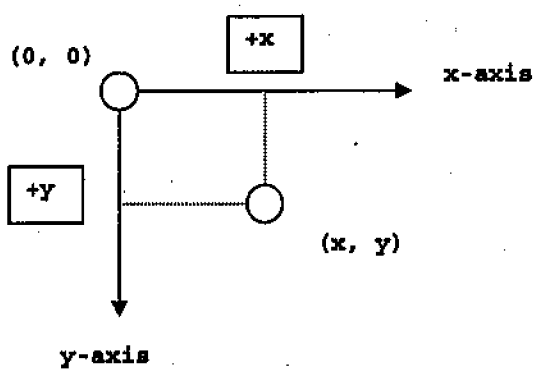



*Figure 2.8.2* The default Visual Basic .NET coordinate system.

### The Point Structure

According to Petzold (2003, p.87) "The Point structure has two read-write properties, named X and Y, which are defined as 32-bit integers." The following lines of code illustrate how to define a point variable called 'Point1':

```
Dim Point1 As Point
Point1.X = 20
Point1.Y = 50
```

The following statement initializes the values (20 and 50):

```
Dim Point1 As New Point(20, 50)
```

### The Array of Points

For example, an array of *Point* structures could represent four corners of an image.

The following statement creates an array of four Point structures:

```
Dim ImageCorners(4) As Point
```

The following lines of code initialize the array elements for the ImageCorners array:

```
Dim ImageCorners As Point() = { _
              New Point(0, 0), _
              New Point(width, 0), _
              New Point(0, height), _
              New Point(width, height) _
              }
```

### The Size Structure

The *Size* structure uses Height and Width properties instead of the X and Y properties. The following statement creates a new Size structure called Size1.

```
Dim Size1 As New Size(20, 50)
```

**The Rectangle Structure**

The *Rectangle* structure defines rectangular shapes and is a combination of a Point and a Size. According to Petzold (2003, p.93) the *Rectangle* structure has two constructors as follow:

```
Rectangle(ByVal pt As Point, ByVal sz as Size)
Rectangle (ByVal x As Integer, ByVal y As Integer, _
             ByVal cx As Integer, ByVal cy As Integer)
```

*Figure 2.8.3* Rectangle Constructors

The following statement creates and initializes a new Rectangle structure called Rectangle1.

```
Dim Rectangle1 As Rectangle = New Rectangle(15, 35, 80, 80)
```

The following lines of code illustrate how to specify the location and size of a rectangle:

```
Rectangle1.Location = New Point(150, 45)
Rectangle1.Size = New Size(80, 40)
```

**The Color Structure**

The Color structure defines methods to manipulate colours, and is based on the ARGB (alpha-red-green-blue) model. According to Deitel, Deitel, & Nieto (2002, p. 687) "The alpha determines the intensity of the colour". Next they explained that all components in the ARGB model are Bytes in the range from 0 to 255. For example, alpha values range from 0 for a transparent colour to 255 for an opaque colour.

The following statement creates a new Color structure called Color1

```
Dim Color1 As Color = New Color()
```

The following line of code uses the Color.FromArgb method to initialize the ARGB values:

```
Dim Color1 As Color = Color.FromArgb(50, 0, 0, 255)
```

According to Petzold (2003, p.105) "The Color structure has 140 shared read-only properties that are actual names of colours ranging (in alphabetical order) from AliceBlue to YellowGreen." He also stated that "The Color class has a 141st property named Transparent that represents a transparent color."

The following line of code uses the Color.Blue method (colour for blue)

```
Dim Color1 As Color = Color.Blue
```

**The Image Class**

The image class can store and manipulate images. The Image class contains several properties that provide information about the size, resolution and pixel format. The following table (Table 2.8.1) lists the properties (selection) of the image class:

Table 2.8.1 *The properties (selection) of the Image Class*

| Property | Type |
|---|---|
| Size | Size (gets the width and height of the Image) |
| Height | Integer (gets the height of the Image) |
| Width | Integer (gets the width of the Image |
| HorizontalResolution | Single |
| VerticalResolution | Single |
| PixelFormat | PixelFormat |

(According to Petzold, 2003, p.482-483)

The *HorizonalResolution* and *VerticalResolution* properties provide the resolution of the image in dots per inch. The *PixelFormat* property shows the format of the pixel information.

| **Appendix G** contains members of the PixelFormat enumerations type |
|---|

For example, the following lines of code show how to display an image's width, height, vertical and horizontal resolution in each of the relevant controls (labels):

```
lblImageWidth.Text = "Width: " & PictureBox1.Image.Width
lblImageHeight.Text = "Height: " & PictureBox1.Image.Height
lblVertRes.Text = "HRes: " & PictureBox1.Image.VerticalResolution
lblHorizRes.Text = "VRes: " & PictureBox1.Image.HorizontalResolution
```

**The PictureBox class**

A picture box (class PictureBox) is used to display graphics from a bitmap and the other image formats. The *Image* property specifies the image you want to display. The following line of code shows how to get an Image object from the file and then display it in the PictureBox called PictureBox1:

```
PictureBox1.Image = Image.FromFile("C:\images\image1.jpg")
```

The *SizeMode* property is responsible for clipping and positioning of the image in the display area. The following statement shows how to change the SizeMode property to AutoSize (resizes PictureBox1 to hold image).

```
PictureBox1.SizeMode = PictureBoxSizeMode.AutoSize
```

**The Pen Class**

A pen is used to draw lines and curves. The following statement shows how to create a basic blue pen that has width of 1 (that is, 1 pixel wide):

```
Dim bluePen As New Pen(Color.Blue, 1)
```

### The Brush Class

Brushes are used with a Graphics object to colour the interiors of solid shapes and to render text. There are several different types of brushes as shown in Table 2.8.2

Table 2.8.2 *Brushes (MSDN, 2003)*

| Brush Class | Description |
|---|---|
| SolidBrush | Paints in a solid colour. |
| HatchBrush | Variety of patterns to paint with |
| TextureBrush | Paints using a texture, such as an image. |
| LinearGradientBrush | Paints two colours blended along a gradient. |
| PathGradientBrush | Paints using a complex gradient of blended colours. |

The following statement shows how to create a solid black brush:

```
Dim mBrush As SolidBrush = New SolidBrush(Color.Black)
```

### The Graphics Class

The Graphics Class provides drawing functions and methods. The following lines of code create a new graphic object called 'NewGraphic' (Graphics object associated with the PictureBox object called 'PictureBox1') and then use the Graphics method *FillRectangle* to draw a solid black rectangle:

```
Dim NewGraphic As Graphics = Me.PictureBox1.CreateGraphics
NewGraphic.FillRectangle(myBrush, 5, 5, 200, 150)
```

**The Bitmap Class**

A Bitmap object is an object used to work with images. The Bitmap class extends the Image Class and allows the programmer to directly read and write to individual pixels. A bitmap has a particular width, height (measured in pixels) and colour depth (the number of bits per pixel). The following lines of code (see Fig. 2.8.4) create a new Bitmap object called 'BitmapFromPicture' (Bitmap object associated with the PictureBox object called 'PictureBox1'), then calculate its dimensions, and finally use the Bitmap method *SetPixel* to set all pixels to black:

```
' Get the bitmap
        Dim BitmapFromPicture As Bitmap = PictureBox1.Image

        'get its dimensions.
        BitmapX = BitmapFromPicture.Width - 1
        BitmapY = BitmapFromPicture.Height - 1

        ' Convert all the pixels to black colour
        For y = 0 To BitmapY
            For x = 0 To BitmapX
                ' Convert each pixel to black.
                BitmapFromPicture.SetPixel(x, y, _
                    Color.FromArgb(255, 0, 0, 0))
            Next x
        Next y

        ' Display the results.
        PictureBox1.Image = BitmapFromPicture
```

*Figure 2.8.4* Bitmap object

## 2.6 The Significance of the Study

Despite the wide range of available techniques, there is a need for an accurate and reliable method to determine wound size. This project will resolve the issue by creating a reliable wound measurement system. An accurate and reliable wound measurement system will save practitioners time and effort.

- In terms of time a new computer technique will reduce human involvement in wound assessment.
- In terms of effort, an accurate computer assisted technique for the measurement of wounds will reduce healthcare cost.

Improving the treatment strategy by providing a computer software program for the measurement of wounds would support clinical practices.

The patients' wound information can be archived, compared and reported. The new software proposed in this study can provide reports which include text, charts, treatment notes and other information such as length, width and depth of the wound. Assessment of the current state of the wound together with comparisons of the sequence of wound data collected over time can help health care professionals in planning appropriate wound care. Thus, management of ulcers will improve, accelerating the wound-healing process and therefore, reducing human suffering.

## 2.7 Research Questions

The main question:

"How may the proposed software increase wound assessment accuracy and reliability"?

The components of the above question are:

1) "Is the application designed to easily record the changes in the size of the wound?"
2) "How may image area, perimeter, length and width affect the overall reliability and validity of a new computerized technique for measuring chronic wounds?"

# 3 DESIGN AND METHODS

## 3.1 Target Population

Random image shapes (polygons, fractals, etc.) have been used in this study.

Only copyright free images and photos have been used in this study.

**Accuracy and Reliability**

One component of the main research question was to see how image area, perimeter, length and width may affect the overall reliability and validity of a new computerized technique for wound measurement. For this purpose, seven test shapes of specified dimensions were drawn using the computer package Corel Draw (version 10), and then the test shapes were measured. The test shapes consisted of:

- Four circles (1, 2, 4, and 6 cm diameter).
- Three rectangles (area 6 $cm^2$ , but differing dimensions – 2cm x 3cm, 1.5cm x 4cm, and 1cm x 6cm)

## 3.2 Design and Procedure of the Study

### 3.2.1 Procedure

The process consists of the following steps:

1. Identify the Needs
2. Design User Interface(Visual Basic .NET)
3. Create a Database (MS Access 2002)
4. Implementation
5. Testing
6. Summary

The project work breakdown structure (WBS) is illustrated in Figure H.1 (see Appendix H: Work Breakdown Structure)

**Identify the Needs**

This study was designed to see how the proposed software may increase wound assessment accuracy and reliability.

The parameters for software construction include the following:

- Able to run under the current Microsoft Windows operating system
- User friendly User Interface

- Precisely and objectively measure wound sizes
    - Scanned image of wound tracing will be calculated by computer digitizing method
    - Calibration of computerizing measurement (using ruler)
    - Calculate the distance on an image using mouse to indicate the two points
    - Calculate the area on an image using mouse to indicate the outline (perimeter) of an image
- Efficiency
    - Patient wound information should be easily stored and retrieved
    - Storing data efficiently in a database – using database tools, such as Microsoft Access
    - Strategies for Construction – coding standards and naming conventions
    - Error Handling
    - Image Processing
        - Contrast
        - Brightness
        - Detect edge
- Availability
    - Splash screen keeps the user informed during start up
    - Menu system
    - Toolbar
    - Shortcut keys
- Security

- Login screen – user is authenticated by entering the password
- Assigning a Database Password – Creating Users and Groups
- Monitoring – Windows Event Log

- Reporting
  - Charts can be viewed
  - Text can be printed or e-mailed

**User-Interface Design**

- How will the user view and communicate with the application?

Figure 3.1 shows the *Use Case(user view and communication)* diagram for the Wound Measurement Software.

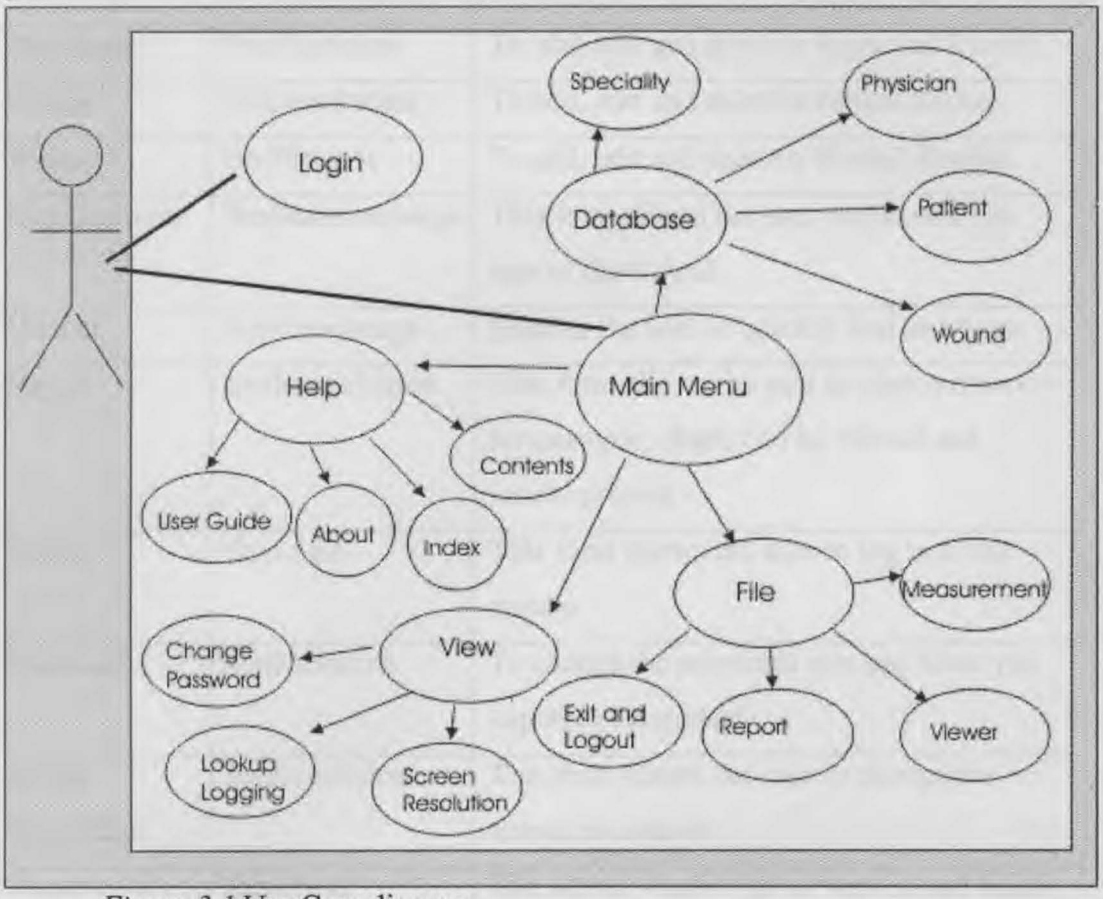



*Figure 3.1* Use Case diagram

As shown in Figure 3.1 the user is first interacting with the login screen, and then with the main manu (if the user is identified to the system).

| |
|---|
| **Appendix I** contains details of the Wound Measurement Software menu implementation. |
| **Appendix K** illustrates the process required when the Wound Measurement System is started. |

**Form Design**

Table 3.1 shows the Wound Measurement Forms.

Table 3.1 *Wound Measurement Forms*

| Form | Name | Purpose |
|---|---|---|
| Specialty | frmSpecialty | To add, edit and delete a Specialty Record. |
| Physician | frmPhysician | To add, edit and delete a Physician Record. |
| Patient | frmNewPatient | To add, edit and delete a Patient Record. |
| Wound | fmrWounds | To add, edit and delete a Wound Record. |
| Measurement | frmMeasureImage | This form allows the user to measure the size of the wound. |
| Viewer | frmViewImage | Enables the user to quickly find an image. |
| Report | frmImageReport | This form allows the user to view a report – for example, charts can be viewed and results printed. |
| Login | frmLogin | This form allows the user to log in to the system. |
| Password | frmPassword | To change the password you use when you log on to the system. |
| Screen Resolution | frmResolution | This form allows the user to change the screen resolution. |
| About | frmAbout | The About window gives version and build information. |
| Event Log | frmEventLog | This keeps record when you log in |
| Splash | frmSplash | Splash screen |
| Main | frmWounds | Main window |

**Database Design**

Figure 3.2 shows the Entity Relationship (ER) diagram for the project.

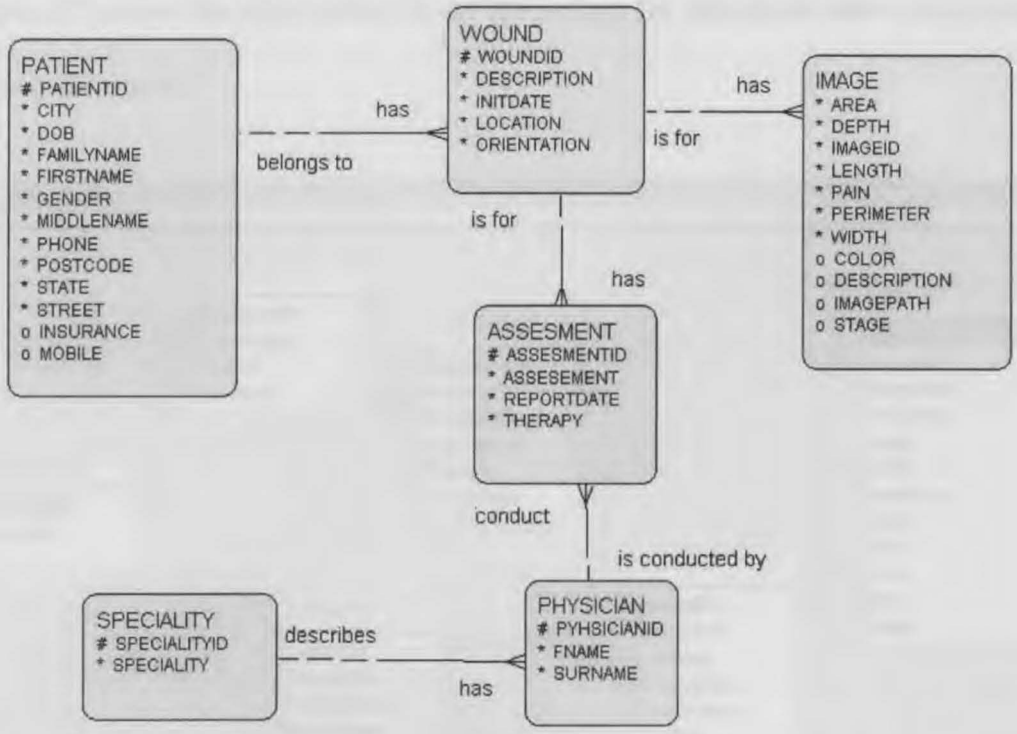



*Figure 3.2* ER diagram

- There is a *one-to-many* relationship between the Patient table and the Wound table.
- There is a *one-to-many* relationship between the Wound table and the Image table.
- There is a *many-to-one* relationship between the Assessment table and the Wound table.
- There is a *one-to-many* relationship between the Physician table and the Assessment table.

- There is a *one-to-many* relationship between the Speciality table and the Physician table.

Figure 3.3 shows the relationships in six tables from the **dbwound.mdb** (Microsoft Access database).

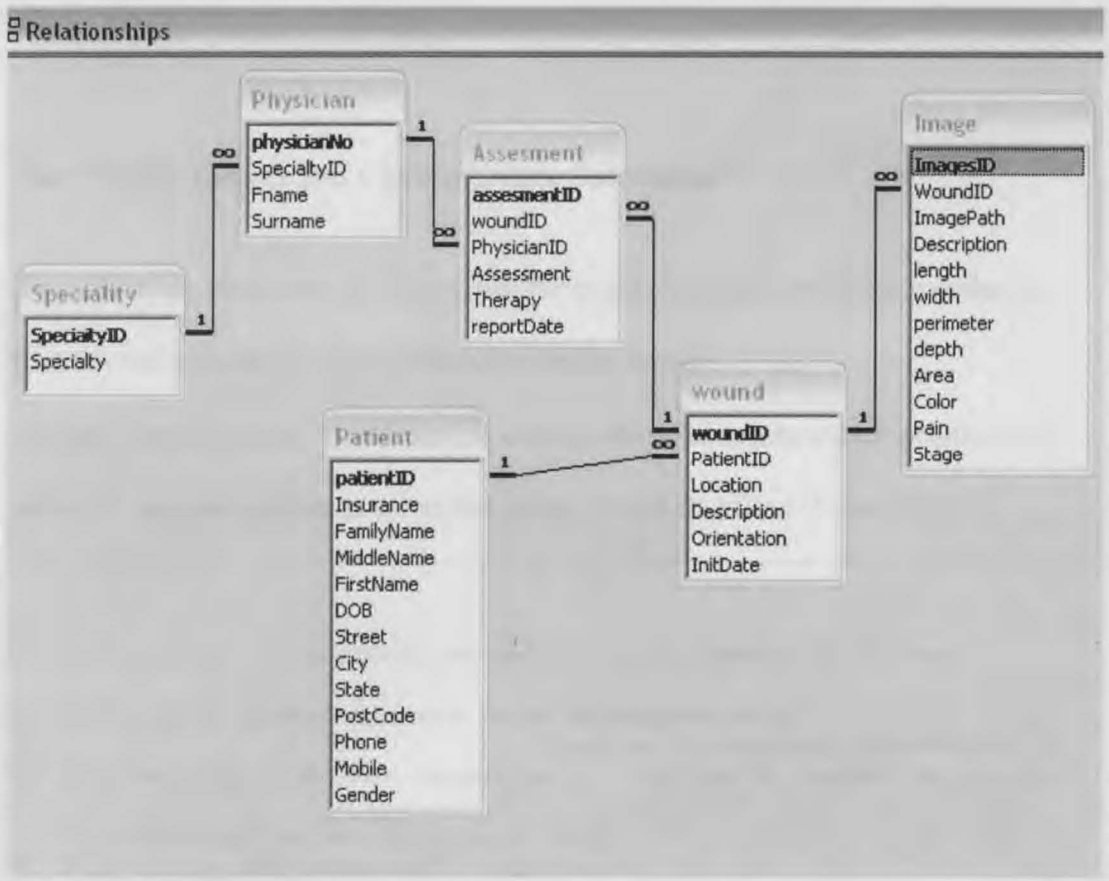



*Figure 3.3* dbwounds.mdb database

### 3.2.2 Implementation

#### Methods

As stated earlier, the wound size assessment should include length, width, area and perimeter. This section describes the methods for calculating a wound size and area. It also explains methods for manipulating images one pixel at a time.

#### The 'Width, Length and Circumference Calculation'

The following code uses the *onmousemove* event (MouseMove event handler) to identify the location of the mouse cursor on the screen.

A point variable called 'NewPoint' is used to store the mouse cursor position (e.X and e.Y) and then elements are added to the ArrayList called 'Point Array'.

```
1. Private Sub PictureBox1_MouseMove(ByVal sender As Object, _

2. ByVal e As System.Windows.Forms.MouseEventArgs) _
3.                                  Handles PictureBox1.MouseMove
4. Dim PointArray As New ArrayList() ' series of connected points

5. Dim NewPoint As New Point(e.X, e.Y)
6. PointArray.Add(NewPoint) ' Add point
```

*Figure 3.4* Capture of mouse movements

Next, in order to measure the distance between two points (see Fig. 3.5) the following formula is used (Note that the Math class method Abs – returns the absolute value):

```
Dim distX As Single = Math.Abs(PointArray(0).X - PointArray(1).X)
Dim distY As Single = Math.Abs(PointArray(0).Y - PointArray(1).Y)
```

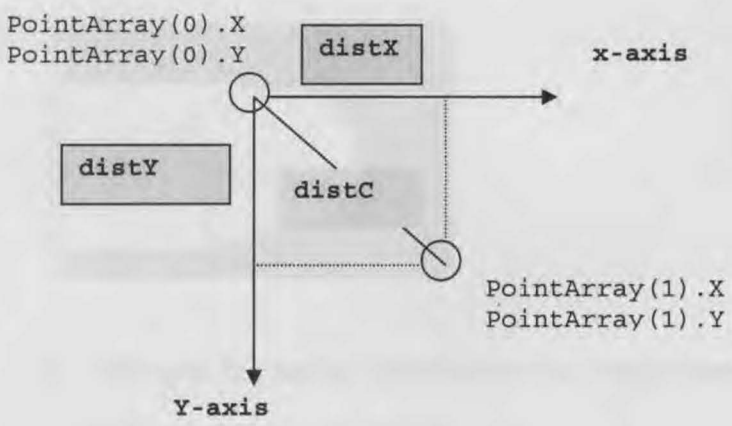



*Figure 3.5* Distance between two points

The following statements calculate the size of the image in centimetres (one inch = 2.54 centimetres):

```
Dim NewGraphic As Graphics = Me.PictureBox1.CreateGraphics
. . .
distX = (distX / NewGraphic.DpiX) * 2.54
distY = (distY / NewGraphic.DpiY) * 2.54
```

Note that "DpiX" and "DpiY" indicates the resolution of the image in dots per inch.

Finally, the following statement uses the Math class method **Sqrt** to calculate the real distance between two points (Pythagoras' theorem).

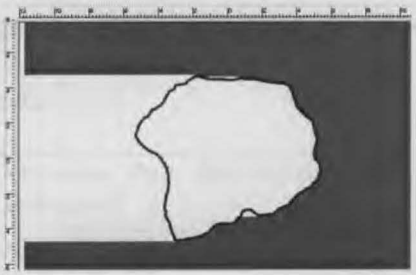
```
distC = Math.Sqrt(distX ^ 2 + distY ^ 2)
totalDistance = totalDistance + distC
```

**Calculate Area**

The image area is measured by first filling the 'negative' image area with lines of red pixels, which will later be counted. The negative image area is then subtracted from the total field area to finally obtain the actual image size.

The algorithm for filling the negative area can be divided into four steps:

1. Fill up to the outline (perimeter) of an image from right to left (←)



2. Fill up to the outline (perimeter) of an image from left to right (→)

3. Fill up to the outline (perimeter) of an image from top to bottom

4. Fill up to the outline (perimeter) of an image from bottom to top

5. Calculate the area enclosed by that outline.

The following lines of code (see Fig. 3.6) demonstrate how to implement such an algorithm. The Image class provides information about the resolution and pixel format.

```
1. Private Sub CalculateArea(ByVal mode As Integer, ByVal color1 As
2. Integer)
3.     On Error GoTo ErrArea
4.     Dim BitmapY, BitmapX, x, y, colorval As Integer
5.     Dim NewGraphic As Graphics = Me.PictureBox1.CreateGraphics
6.     ' Get dimensions of bitmap
7.     PictureBox1.Image = BitmapFromPicture
8.     BitmapY = BitmapFromPicture.Height - 1
9.     BitmapX = BitmapFromPicture.Width - 1
10.    ' Set cursor to wait cursor
11.    Cursor = System.Windows.Forms.Cursors.WaitCursor
12.    Call FillRightLeft(BitmapX, BitmapY, color1)
13.    Call FillLeftRight(BitmapX, BitmapY, color1)
14.    Call FillUpDown(BitmapX, BitmapY, color1)
15.    Call FillDownUp(BitmapX, BitmapY, color1)
16.
17.    ' Display the results.
18.    PictureBox1.Image = BitmapFromPicture
19.    ' Set cursor to arrow cursor
20.    Cursor = System.Windows.Forms.Cursors.Arrow
21.    Dim counter As Long
22.    Dim area As Integer
23.    counter = 0
24.    'count number of red pixels
25.    For y = 0 To BitmapY
26.        For x = 0 To BitmapX
27.            With BitmapFromPicture.GetPixel(x, y)
28.                If .R = 255 And .G = 0 And .B = 0 Then
29.                    counter = counter + 1
30.                End If
31.            End With
32.        Next
33.    Next
34.    ' area percentage
35.    area = 100 - counter / BitmapY / BitmapX * 100
36.    'display area in square centimeters
37.    txtArea.Text = CStr((BitmapY / NewGraphic.DpiY) * 2.54 _
38.             * (BitmapX / NewGraphic.DpiX) * 2.54) * area / 100
39. ErrArea:
```

```
40.          MsgBox("Unexpected error - GDI+. Try again",
41.                 MsgBoxStyle.Critical, "Error Message to User")
42.
43. End Sub
```

*Figure 3.6* Calculate Area

**Appendix L** contains details about the implementation of this software.

**Image Processing**

This section covers image processing methods. According to Stephens (2000, p. 163) "Using these techniques, you can adjust an image's colour balance, brightness .."

- **Example: The Complement Technique**

According to Stephens (2000, p. 173) "To complement an image, you simply subtract each pixel's red, green, and the blue components values from the largest values from the possible value of 255." The following lines of code (Fig. 3.7) complement an image:

```
With BitmapFromPicture.GetPixel(x, y)
    'I changed this part to work
    red = 255 - .R
    Green = 255 - .G
    Blue = 255 - .B
End With
BitmapFromPicture.SetPixel(x, y, _
    Color.FromArgb(255, red, Blue, Green))
```

*Figure 3.7* Complement

- **Example 2: Brightness (Decrease)**

According to Stephens (2000, p. 174) "To decrease a colour's brightness by a certain percentage, a program can subtract from each pixel's components the percentage times the component's value." The following lines of code (Fig. 3.8) make the colour darker (by 20 percent):

```
factor = 0.2
With BitmapFromPicture.GetPixel(x, y)
  If value < 1 Then
      'make it darker
      red = (1 - factor) * .R
      Green = (1 - factor) * .G
      Blue = (1 - factor) * .B
  Else

End Width
```

*Figure 3.8* Brightness

## 3.3 Limitations

This project was originally intended to be developed at PhD level. However, in this case it has been conducted at honours (undergraduate) level. Consequently this study has been limited in the scope. It is also limited by the relatively small amount of arbitrary data (shapes) used to test validity.

# 4 RESULTS

## 4.1 Main Question

*"How may the proposed software increase wound assessment accuracy and reliability"?*

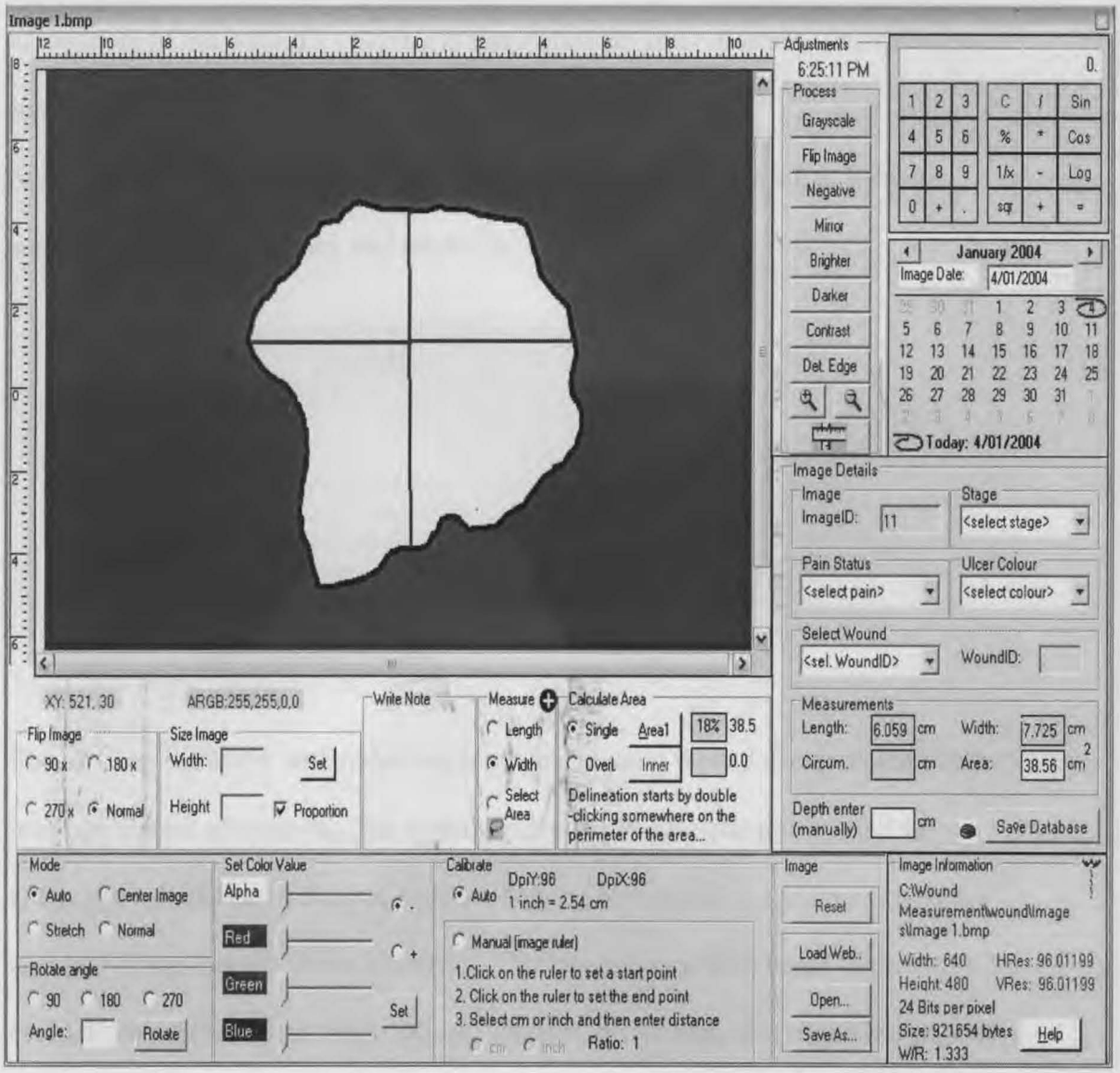



*Figure 4.1* Wound Measurement user-interface

### 4.1.1 Evaluation of Functionality

In response to the research question, this section measures aspects of the implementation in order to evaluate the functionality of the program.

The system was developed in accordance with the design and methods outlined in section three. The program enables the user to calculate the linear distance (width and length) of an image. The mouse is used to trace the cursor around the image outline (perimeter). The subsequent automatic calculation of the wound size and assessment of the wound's coloring is more accurate and repeatable (reliable) than manual methods alone can be.

Manual calibration (fig. 4.2) of the measurement can be carried out to increase wound assessment accuracy and reliability.

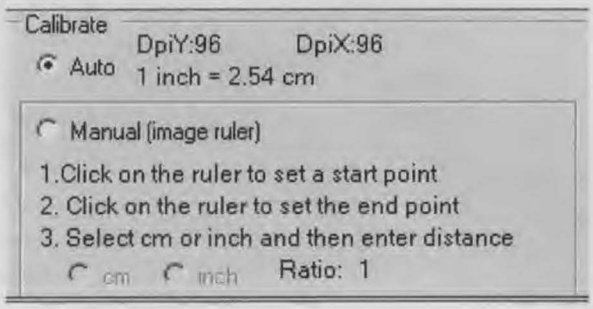



*Figure 4.2* The Calibration Panel

The obvious difficulty with obtaining reliable and valid wound measurements is the irregular shapes of wounds. The system addresses this by providing a tool for automatic calculation of size, as well as allowing for manual calculations, with the user providing data for shape and depth. The Image Processing Panel (figure 4.3) enables the user to adjust colour values, contrast, brightness, size of the image, and other attributes.

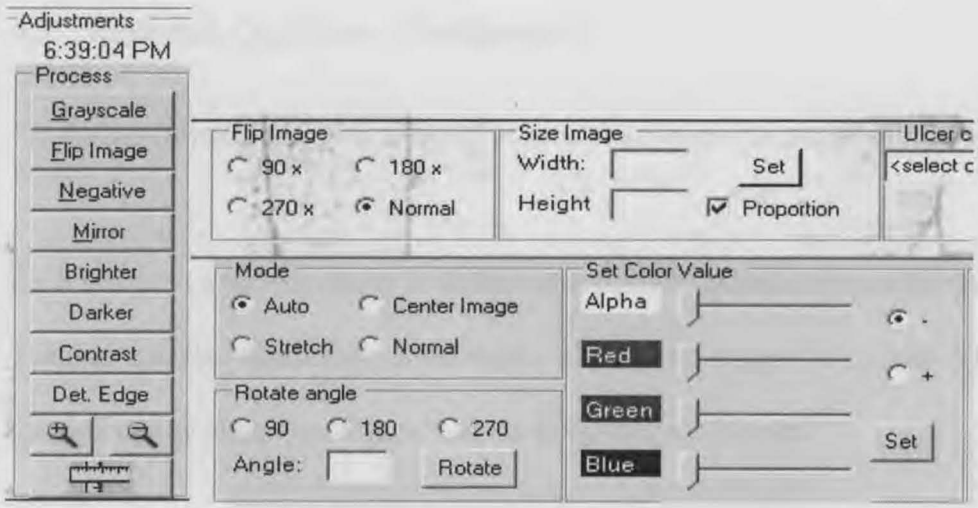



*Figure 4.3* Image Processing Panel

In addition to this, at any time, a patient's wound images and image data (see Figure 4.4) may be viewed in chronological order for the purpose of comparison. This provides a visual and informative record of changes to the wound over time (for example, charts can be viewed and information printed).

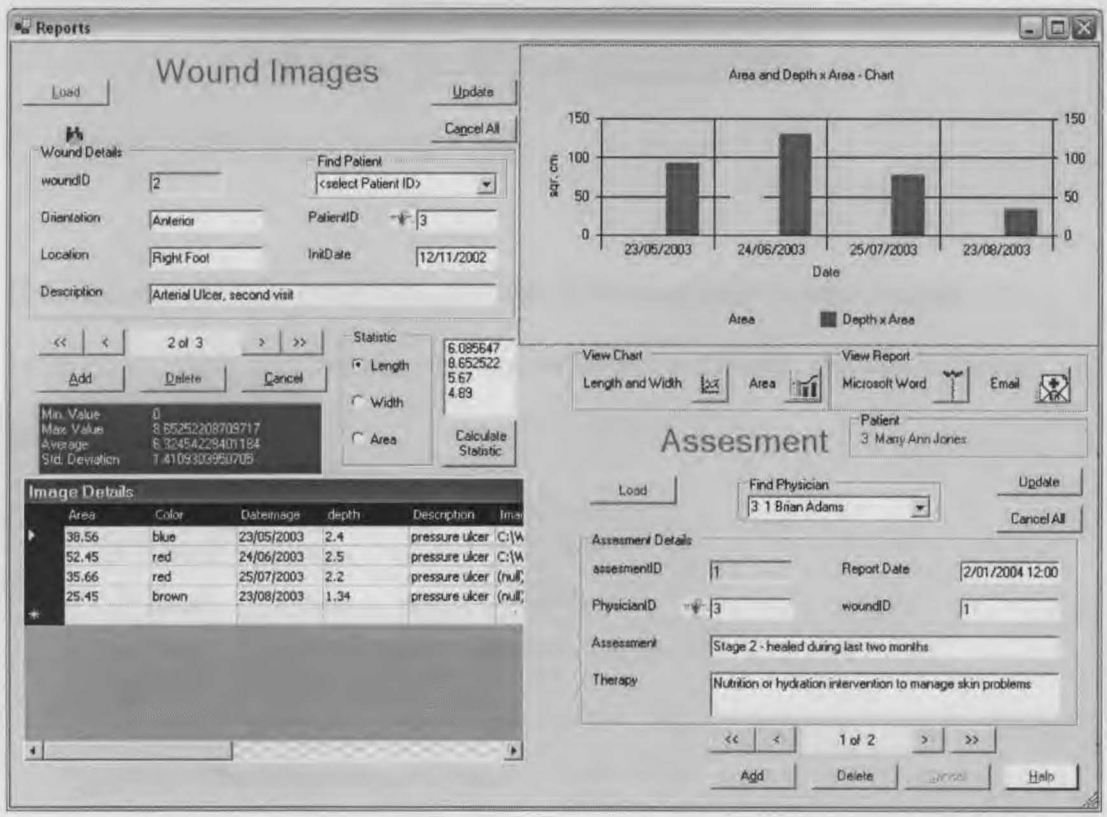



*Figure 4.4* Form Wound Images

## 4.2 Research Question - Component 1

*"Is the application designed to easily record the changes in the size of the wound?"*

Each image in a patient record includes automatically calculated data for wound size. The user can also select the colour, degree of pain and stage of the ulcer. The system can record any notes (see Figure 4.6) made by the practitioner.


*Figure 4.6* The Image Record Panel

If the manual mode is being used, the user is required only to input simple information, such as delineating the wound length, and width, and the perimeter (using 'Select Area').

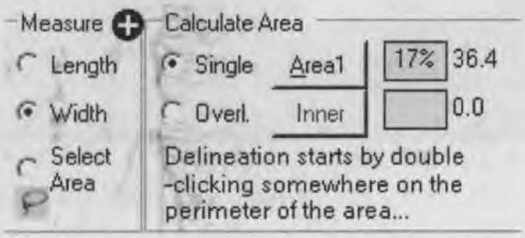



*Figure 4.5* The Measurement Panel

Features such as the ruler, calculator and measurement line (Figure 4.7), make the recording and annotation of images very user friendly.

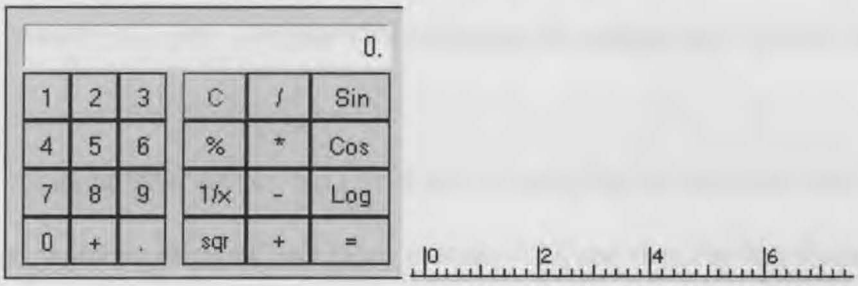



*Figure 4.7* Calculator and Ruler

Finally, Wound Measurement Software provides all the features necessary to work with databases, such as entering new data, and modifying or viewing existing data. Figure 4.8 shows the Physician form.

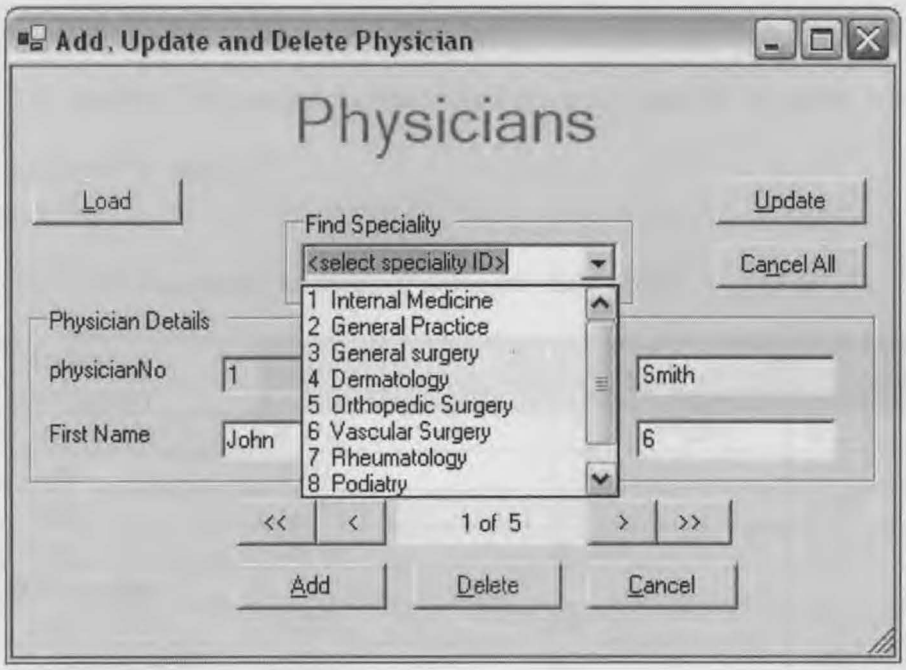



*Figure 4.8* Physician Form

## 4.3 Research Question – Component 2

*"How may image area, perimeter, length and width affect the overall reliability and validity of a new computerized technique for measuring chronic wounds?"*

As mentioned earlier, ten test shapes of specified dimensions were drawn using the computer package Corel Draw (version 10), and then the test shapes were measured.

| **Image Information** |
|---|
| Horizontal Resolution: 182.82 pixels/inch |
| Width: 640 |
| Vertical Resolution: 182.82 pixels/inch |
| Height: 480 |
| 24 bits per pixel |

The results of the statistical analysis of the study data for the **auto mode** are presented in table 4.1

Table 4.1 *Statistical Analysis of Data for Auto Mode*

| **Circles (diameter)** | **Result (area) $cm^2$** | **Expected Result $cm^2$** | **Difference** | **Accuracy** |
|---|---|---|---|---|
| 1cm | 0.762 | 0.785 **($r^2$xPi)** | 0.023 | 97.07% |
| 2cm | 3.10 | 3.141 | 0.041 | 98.69% |
| 4cm | 12.48 | 12.57 | 0.09 | 99.28% |
| 6cm | 28.16 | 28.274 | 0.114 | 99.6% |
| **Rectangles** | | | | |
| 2cm x 3cm | 6.003 | 6.0 | +0.003 | 99.95% |
| 1.5 cm x 4cm | 6.003 | 6.0 | +0.003 | 99.95% |
| 1cm x 6cm | 5.989 | 6.0 | -0.011 | 99.82% |

The results of the statistical analysis of the study data for **manual delineation** (after calibration) are presented in table 4.2

Table 4.2 *Statistical Analysis of Data for Manual Delineation*

| **Circles (diameter)** | **Length(cm)** | **Expected Result** | **Width(cm)** | **Expected Result** | **Difference** | **Accuracy** |
|---|---|---|---|---|---|---|
| 1cm | 1.000 | 1 | 1.000 | 1 | 0.000, 0.000 | 100%, 100% |
| 2cm | 2.014 | 2 | 2.028 | 2 | +0.014, +0.028 | 99.30%, 98.61% |
| 4cm | 4.046 | 4 | 4.029 | 4 | +0.046, +0.029 | 98.86%, 99.28%, |
| 6cm | 6.043 | 6 | 6.084 | 6 | +0.043, +0.084 | 99.23%, 98.61% |
| **Rectangles** | | | | | | |
| 2cm x 3cm | 2.043 | 2 | 3.071 | 3 | +0.043, +0.071 | 97.89%, 97.80% |
| 1.5 cm x 4cm | 1.550 | 1.5 | 4.114 | 4 | +0.050, +0.114 | 96.77%, 97.22% |
| 1cm x 6cm | 1.000 | 1 | 6.116 | 6 | 0.000, +0.116 | 100%, 98.10% |

# 5 CONCLUSION

The primary objective of the research was the development of a software application for wound measurement.

This software program is an easy-to-use application that allows clinicians to measure the progress of a wound.

## 5.1 Accomplishment of Objectives

The system described in this document is completed in all areas stated in the design section of this thesis, however quantitative and qualitative comparisons against existing methods are necessary to test the real value of the proposed Wound Measurement System.

The Wound Measurement Software program imports images of wounds, and then uses a unique measurement approach to accurately measure wound sizes. The result of this study shows that this measurement system is able to precisely detect the width, length and area of an image.

The system is simple to use, and contains the essential features necessary to monitor wound healing.

In summary, the answer to the main research question is in the affirmative.

## 5.2 Contribution of this Work

This thesis claims the following as its original contribution to knowledge:

- The original design and algorithms for the purposes of area measurement
- A unique way of handling complex graphic detail
- The analysis of the Visual Basic .NET language
- The analysis of wound measurement techniques and methods

## 5.3 Further Research

There are a number of areas where the research gained through this project could be expanded. Below are some areas which would be useful for further research:

- Quantitative and qualitative testing against existing methods
- Clinical trials using sufficient varied data
- User (practitioner) testing
- More research on algorithms

# 6 APPENDICES

# APPENDIX A: DATA TYPE CHANGES IN VISUAL BASIC

The following table (Table A.1) describes some data type changes from previous versions in Visual Basic .NET.

Table A.1 *Lists data type changes in Visual Basic .NET (based on Evjen & Beres, 2002, p. 18- 21).*

| Topic | Changes | Visual Basic 6 | Visual Basic .NET |
|---|---|---|---|
| *Data Type* | Variant variable not supported | Dim num1 As Variant<br><br>(Variant data type was the universal data type, default type) | Dim num1 as Object<br><br>(Any type can be stored in a variable of type Object)<br>The default value is Nothing |
| | Integer | Dim num1 As Integer<br><br>(Integer data type was a 16-bit number) | Dim num1 As Short<br><br>(The Short data type is a 16-bit number) |
| | Long | Dim num1 as Long<br><br>(The long data type was a 32-bit number) | Dim num1 As Integer<br><br>(Using Integer for 32-bit operations) |
| | Currency not supported (data type for calculations involving money) | Dim money as Currency<br><br>(Currency was a 64-bit number, with 4 digits to the right of the decimal points) | Dim money As Decimal<br><br>(The new Decimal data type is a 96-bit signed integer and can have up to 28 digits to the right of the decimal place) |
| | Date | Dim mydate as Double<br><br>(A Date is stored in a Double format was a 64-bit | Dim myDate as Date<br><br>(The Date type is now a 64-bit integer) |
| *Array* | Option Base not supported | Dim x (0 to 4) as String<br><br>In Visual Basic 6.0, you can change the default lower bound (0) to 1 with the Option Base statement | Dim x (4) as String<br><br>(now declares just the upper bound to the array) |

# APPENDIX B: VISUAL BASIC .NET VALUE TYPES

**Value Types (Built-in types)**

There are several new fundamental data types. According to Petzold (2003, p.11) Visual Basic .NET supports four integral data types, which are listed below:

Table B.1 *Visual Basic .NET Integral Data Types*

| Number of Bits | Data Type |
|---|---|
| 8 (unsigned) | Byte |
| 16 (signed) | Short |
| 32 (signed) | Integer |
| 64 (signed) | Long |

Petzold (2003, p.12) also stated that Visual Basic .NET supports the two common floating-point data types Single (32 bits) and Double (64 bits) and types Boolean (16 bits), Date (64 bits), Decimal (128 bits) and Char (16 bits).

**Constants**

Constants are added to the program when you want to represent values that will not change during the program execution. The following code illustrates the syntax of a constant:

```
Cons ConstantName As Data type = Value
```

For example, the following code defines a constant called PI:

```
Const PI As Single = 3.14159
```

**The Dim Statement**

The *Option Explicit* statement determines whether local variables may be implicitly declared (set to ON by default, forces the programmer to declare all variables explicitly.). When set to ON, *Option Strict* can prevent the accidental conversion of one variable data type to another (set OFF by default).

For this reason, Visual basic .NET variables should be declared before you can use them. According to Halvorson (2003, p. 126) "You should declare all variables using **Dim** and keyword **As** to identify the type of data that they'll hold." This is a change from earlier versions of Visual Basic, where you could omit a Dim statement (you could declares a variable implicitly). For example, the following statement creates a variable named Grade:

```
Dim Grade As Integer
```

# APPENDIX C: VISUAL BASIC .NET REFERENCE TYPES

**Reference Types**

The built-in reference types include Object and String. The following table (Table C.1) lists the fundamental reference types in Visual Basic .NET.

Table C.1 *Visual Basic .NET Object and Strings*

| Data Type | Size | Range | Sample Usage |
|---|---|---|---|
| String | Usually 16-bits per character | 0 to approximately 2 billion 16-bit Unicode characters | Dim Dog As String<br>Dog = "pointer" |
| Object | 32-bit | Any type can be stored in a variable of type Object | Dim MyApp As Object<br>MyApp = CreateObject _<br>("Word.Application") |

Halvorson (2003, p. 139)

**Visual Basic .NET Arrays**

Arrays refer to a series of variables (group of contiguous memory location) with the same name and the same type. You can refer to a particular location in an array by using the index (position) number. Arrays can be one (single) dimensional or multi-dimensional (you can specify up to 32 dimensions). According to According to Deitel, Deitel, & Nieto (2002, p. 247) "The first element in every array is zeroth element".

Figure C.1 shows an integer array named intArray.

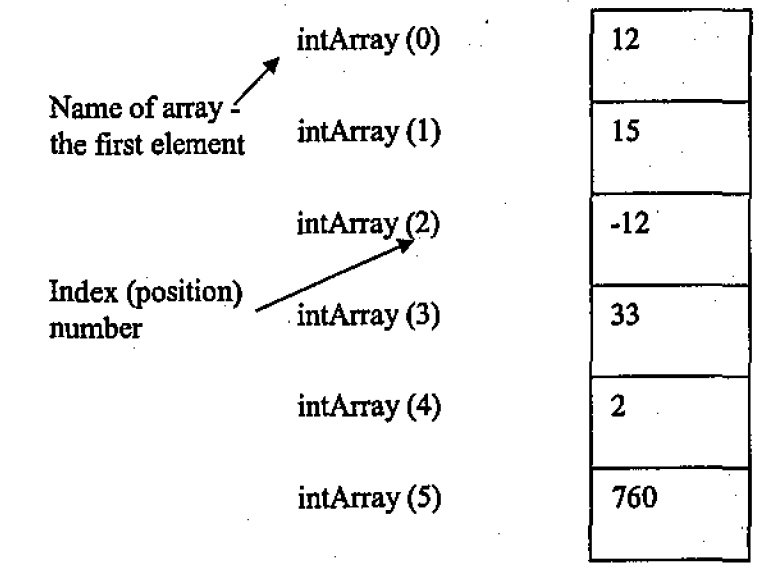



*Figure C.1* Array consisting of 6 elements.

The following statement declares (create a variable that can store a reference to array) the array in Fig C.1:

```
Dim intArray As Integer()
```

To allocate memory (arrays occupy space in memory) for the array intArray, the statement

```
intArray = New Integer(5) {}
```

is used. Note that the number 5 defines the upper bound for the array. This can be combined into one statement:

```
Dim intArray As Integer() = New Integer(5) {}
```

The following statement creates the array intArray and initializes the values in the array:

```
Dim intArray As Integer() = New Integer(5) {12, 15, -12, 33, 2, 760}
```

Notice that the size of the intArray (consisting of 6 elements) is one member larger then the upper bound (5) specified in the allocation.

*Multidimensional Array*

The following statements create a two-dimensional array, intTwoDimArray, of type Integer.

```
Dim intTwoDimArray As Integer(,)
intTwoDimArray = New Integer(,) {{1, 2}, {2, 3}}
```

# APPENDIX D: VISUAL BASIC .NET OPERATORS

**Arithmetic Operators**

In addition to the four basic mathematical operators:

- *Addition* (+ operator)

  Syntax `result = number1 + number 2`

- *Multiplication* (* operator)

  Syntax `result = number1 * number2`

- *Subtraction* (- operator)

  Syntax `result = number1 - number2`

- *Division* (/ operator)

  Syntax `result = number1 / number2`

Visual Basic also provides four advanced operators, which perform:

- *Exponentiation* (^ operator)

  Syntax `result = number^exponent`

- *String Concatenation* (& operator)

  Syntax `result = expression1 & expression2`

- *Integer Division* (\ operator)

  Syntax `result = number1 \ number2`

- *Remainder Division* (**Mod** operator)

Syntax `result = number1 Mod number2`

The arithmetic operators have an order of precedence that is described below (see Table D.1).

Table D.1 *Order of Precedence (Arithmetic Operators), (Adapted from table in Deitel, Deitel & Nieto 2002, p. 77)*

| Operation | Symbol | Precedence (Order of Evaluation) |
|---|---|---|
| *Parenthesis* | ( ) | Evaluated first. |
| *Exponentiation* | (^) | Evaluated second. |
| *Multiplication and division* | (*, /) | Evaluated third. |
| *Integer division* | (\) | Evaluated fourth. |
| *Modulus arithmetic* | (Mod) | Evaluated fifth. |
| *Addition and subtraction, String concatenation* | (+, -), (+) | Evaluated sixth. |
| *String concatenation* | (&) | Evaluated last. |

Notice that each operation is evaluated in order of appearance from left to right.

Parentheses can be also used to override the order of precedence.

**Relational and Equity Operators**

The following table (Table D.2) lists the relational and equity operators in Visual Basic .NET

Table D.2 *Visual Basic .NET Object and Strings (Adapted from table in* Deitel, Deitel & Nieto 2002, p. 79)

| Visual Basic Operator | Example of Visual basic condition | Meaning of Visual Basic condition |
|---|---|---|
| *Equity operator* | | |
| = | x = y | **x** is equal to **y** |
| <> | x <> y | **x** is not equal to **y** |
| *Relational operators* | | |
| > | x > y | **x** is greater than **y** |
| < | x < y | **x** is less than **y** |
| >= | x >= y | **x** is greater than or equal to **y** |
| <= | x <= y | **x** is less or equal to **y** |

**Logical Operators**

Visual Basic .NET provides the following logical operators:

- **And** (a logical And operation is performed, see below)
- **AndAlso** (a logical And operation is performed only if the first operand is evaluated to False)
- **Or** (A logical Or operation is performed)
- **OrElse** (if the first operand is True the expression returns True; otherwise a logical Or is performed)
- **Xor** (a logical exclusive-Or operation is performed)

- And - if both bits are 1 the result bit is 1; otherwise the result is 0.
- Or – if either bit is 1 the result bit is 1; otherwise the result is 0.
- Xor – if either bit is 1 but not both, the result is 1; otherwise the result is 0.

# APPENDIX E: VISUAL BASIC .NET CONTROL STRUCTURE

***Sequence***

Normally, a computer executes a Visual Basic statement sequentially (one after another in the order in which they are written). However, there are various Visual Basic statements that enable the programmer to state that the next statement will not be one in sequence (for example, use of GoTo statement in order to transfer control). According to Deitel, Deitel, & Nieto (2002, p. 99) there is a problem with programs containing GoTo statements: "The excessive use of GoTo statements caused the program to become quite unstructured and hard to follow."

***Selection***

Visual Basic provides three types of selection structures:

- ***If/Then***
- ***If/Then/Else***
- ***Select Case***

***If/Then* Selection Structure**

**If** *condition* **Then** [ *statements* ]

*If/Then* selection control structure performs an action if a condition is true. An example of the *If/Then* selection structure is shown in Figure E.1:

```
If Grade >= 70 Then
    Console.WriteLine("Distinction")
End If
```

*Figure E.1* If/Then Selection example

If the condition is true (if the student grade is greater or equal to 70), then "Distinction" is printed, otherwise the print statement is ignored.

***If/Then/Else Selection Structure***

**If** *condition* **Then** [ *statements* ] [ **Else** *elsestatements* ]

*If/Then/Else* selection structure performs an action if a condition is true and performs a different action of the condition is false. An example, of the *If/Then/Else* structure is shown in Figure E.2.

```
If Grade >= 50 Then
    Console.WriteLine("Passed")
Else
    Console.WriteLine("Failed")
End If
```

*Figure E.2* If/Then/Else Selection example

If the condition is true (student grade is greater or equal to 50), then "Passed" is printed, otherwise (grade is less then 50) prints "Failed".

***Select Case Structure***

**Select** [ **Case** ] *testexpression*
  [ **Case** *expressionlist*
    [ *statements* ] ]
  [ **Case Else**
    [ *elsestatements* ] ]
**End Select**

The **Select Case** structure performs one of many actions depending on the value of an expression. If the matching "Case" is found, the code in the Case executes. The **Case Else** (optional) statement is used to check for invalid input. The required **End Select** keywords terminate the Select Case block. Multiple expression clauses (when multiple values are tested in a Case statement) are separated by commas. An example of the *Select Case* structure is shown in Figure E.3.

```
Select Case Grade
      Case 80 To 100
            Console.Write("HD")
      Case 70 To 79
            Console.Write("D")
      Case 60 To 69
            Console.Write("CR")
      Case 50 To 59
            Console.Write("P")
      Case 0, 1 To 49
            Console.Write("F")
      Case Else
            Console.Write("Incorrect value")
End Select
```

*Figure E.3* Select Case structure example

The first Case statement determines if Grade is between 80 and 100 inclusive. The next Case statement determines if grade is between 70 and 79 inclusive, and so on.

***Repetition***

Visual Basic provides seven types of repetition structures – While, Do While/Loop, Do/Loop While, Do Until/Loop, Do/Loop Until, For/Next and For Each/Next. The **While/End While** repetition structure, **Do/Loop While** repetition structure and **For/Next** repetition structure are discussed in more details below.

***While/End While Repetition Structure***

**While** *condition*
[ *statements* ]
**End While**

The **While/End While** repetition structure executes a series of statements as long as a given condition is true. An example of the **While/End While** repetition structure is shown in Figure E.4.

```
Dim counter as Integer = 1

While counter <= 10
      Console.Write(counter)
      counter += 1
End While
```

*Figure E.4* While/End While loop example

When the **While** structure is entered the counter is 1. The next variable *counter* is repeatedly printed out and increased by 1 until the product becomes 11, so the condition (counter <= 10) in the while structure becomes false. Because it is False, execution resumes with the statement following the **End While** statement.

***Do/Loop While Repetition structure***

**Do**
  [ *statements* ]
[ **Exit Do** ]
  [ *statements* ]
**Loop** { **While** | **Until** } *condition*

The **Do/Loop While** structure repeats a block of statements while the condition is true. The **Do/Loop Until** structure repeats a block of statements until the condition becomes true. The **Exit Do** statement transfers control immediately to the statement following the **Loop** statement.

An example of the **Do/Loop Until** repetition structure is shown in Figure E.5.

```
Dim counter As Integer = 1

Do
      Console.Write(counter)
      counter += 1
Loop Until counter > 9
```

*Figure E.5* Do/Loop Until loop example

***For/Next Repetition Structure***

**For** *counter* = start **To** end [ **Step** *step* ]
  [ *statements* ]
[ **Exit For** ]
  [ *statements* ]
**Next** [ *counter* ]

The **For/Next** repetition structure repeats a group of statements a specified number of times. The **To** keyword is required in the **For/Next** structure. The keyword **Step** (optional) specifies the amount by which the *counter* (the control variable) is incremented each time through the loop. The **Exit For** statement transfers control immediately to the statement following the Next statement. An example of the **For/Next** repetition structure is shown in Figure E.6.

```
For x = 1 To 9 Step 1
      Console.Write(counter)
Next
```

*Figure E.6* For/Next loop example

When the For/Next structure begins its execution, the control variable **x** is initialized to 1. Then, when the **Next** keyword is reached, variable **x** is increment by the specified value of 1 (**Step** 1) and the loop continues until condition counter <= 9 is tested.

# APPENDIX F: VISUAL BASIC .NET EXCEPTION HANDLING

**Structured Exception Handling**

Visual basic .NET also provides the Finally block. According to Deitel, Deitel, & Nieto (2002, p. 453) "Visual Basic's exception handling mechanism provides the **Finally** block, which is guaranteed to execute if program control enters the corresponding **Try** block." They explain that the Finally block is executed following the code in the Catch block regardless of whether or not an exception occurs, so this makes the Finally block "an effective way to eliminate resource leaks." (Deitel, Deitel, & Nieto, 2002, p. 453). The Try...Catch...Finally syntax is shown in Fig. F.1.

```
Try
  ... (Statements that may produce runtime errors)
Catch ex As Exception
  ... (Statements to run only if error occurs)
Finally
  ... (Statements to run always - errors occurs or not)
  ... (Usually perform cleanup operations)
End Try
```

*Figure F.1* Try...Catch...Finally block

*Throwing Exception*

The Throw statement raises an exception you can use to trap errors within your code. The following example demonstrates the Throw statement:

```
Dim password As String
Dim correctPassword As Boolean = False

 If password = "wound" Then
      correctPassword = True
 Else
      Throw New System.ArgumentException("Invalid password")
 End If
```

The New keyword creates a new object of type System.ArgumentException to trap the error (ArgumentException) and then display information: Invalid password.

**Unstructured Error Handling**

Unstructured exception handling is implemented using the following statements:

- Error statement (throws a Microsoft.VisualBasic.Helpers.VB6Exception exception)
- On Error Statement (On Error GoTo *LabelName* ->exception-handler location).
- Resume Statement (the current exception is set to Nothing)

The following code block demonstrates the use of the On Error statement.

```
1. Private Sub ErrorHandler()
2.         Dim x As Integer
3.         On Error GoTo errTrap
4.         x = x / 0
5.         Exit Sub
6.  errTrap:
7.         MsgBox("Divide by zero error")
8.  End Sub
```

The **On Error GoTo** statement (line 3) tells the compiler that if an error does occur (line 4, x = x / 0), then jump to the **errTrap** label (line 6) and display the error message (line 7).

# APPENDIX G: PIXELFORMAT ENUMERATION

The following table (Table G.1) lists the members of the PixelFormat enumeration.

Table G.1 *PixelFormat Enumeration (selection)* (MSDN, 2003)

| Member | Description |
|---|---|
| **DontCare** | No pixel format is specified. |
| **Format1bppIndexed** | Specifies that the pixel format is *1 bit* per pixel and that it uses indexed colour. The colour table therefore has *two colours* in it. |
| **Format4bppIndexed** | Specifies that the format is *4 bits* per pixel, indexed. |
| **Format8bppIndexed** | Specifies that the format is *8 bits* per pixel, indexed.<br>The colour table therefore has *256 colours* in it. |
| **Format16bppGrayScale** | The pixel format is *16 bits* per pixel. The colour information specifies *65536 shades* of grey. |
| **Format16bppRgb555** | Specifies that the format is *16 bits* per pixel; 5 bits each are used for the red, green, and blue components. The remaining bit is not used. |
| **Format16bppRgb565** | Specifies that the format is *16 bits* per pixel; 5 bits are used for the *red* component, 6 bits are used for the *green* component, and 5 *bits* are used for the blue component. |
| **Format16bppArgb1555** | The pixel format is *16 bits* per pixel. The colour information specifies *32,768 shades of colour*, of which 5 bits are red, 5 bits are green, 5 bits are blue, and 1 bit is alpha. |
| **Format24bppRgb** | Specifies that the format is *24 bits* per pixel; 8 bits each are used for the red, green, and blue components. |
| **Format32bppRgb** | Specifies that the format is *32 bits* per pixel; 8 bits each are used for the red, green, and blue components. The remaining 8 bits are not used. |
| **Format32bppArgb** | Specifies that the format is *32 bits* per pixel; 8 bits each are used for the alpha, red, green, and blue components. |

| | |
|---|---|
| **Format48bppRgb** | Specifies that the format is *48 bits* per pixel; 16 bits each are used for the red, green, and blue components. |
| **Format64bppArgb** | Specifies that the format is *64 bits* per pixel; 16 bits each are used for the alpha, red, green, and blue components. |

# APPENDIX H: WORK BREAKDOWN STRUCTURE

The project work breakdown structure (WBS) is illustrated in Figure H.1

**1 Analysis**

- 1.1 Identify needs and benefits
- 1.2 Define desired output
  - 1.2.1 Observe current approach to problem
  - 1.2.2 Define the functionality/behavior
- 1.3 Create a Scope Definition
- 1.4 *Milestone*: Analysis phase Completion

**2 Planning and Design**

- 2.1 Software Product Design
- 2.2 Database Design
- 2.3 Requirements Plan
- 2.4 Integration Plan
- 2.5 Test Plan
- 2.6 *Milestone*: Planning Completion
- 2.7 Risk Analysis
- 2.8 *Milestone:* Risk Completion

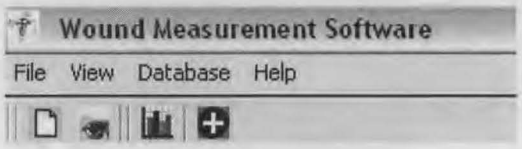
**3 System Engineering**

- 3.1 Software Requirements
- 3.2 Implementation
  - 3.2.1 Develop algorithm
  - 3.2.2 Detect wound edge
  - 3.2.3 Calculate area
- 3.3 Testing
  - 3.3.1 Unit Testing

*Figure H.1* Work Breakdown Structure

# APPENDIX I: MENU STRUCTURE AND ICONS

**Main Menu**




There are four main menus:

- File
- View
- Database
- Help

**File Menu**

The *File menu* has four items implemented: New Measurement, Open Thumbnail, Report Viewer and Exit.

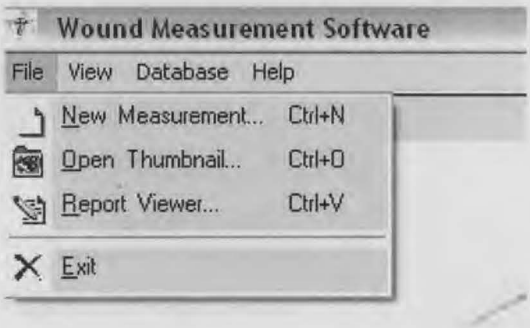



*Figure I.1* File Menu

Table I.1 *Purpose of File Menu*

| Menu | Purpose |
|---|---|
| New Measurement… (Ctrl+N) | Opens a new image (allows the user to measure the size of the wound) |
| Open Thumbnail… (Ctrl+O) | Opens the Thumbnail window (enables the user to quickly find an image) |
| Report Viewer… (Ctrl+V) | Opens the Report Viewer window (allows the user to view a number of images – for example, charts can be viewed and results printed) |
| Exit | Quits the application |

**View Menu**

The View menu has three items implemented: Password, Logging and Screen Resolution.


*Figure I.2* View Menu

Table I.2 *Purpose of View Menu*

| Menu | Purpose |
|---|---|
| Password… (Ctrl+P) | Displays the Password window (use this procedure to change the password you use when you log on system) |
| Logging… (Ctrl+L) | Displays the Logging window (keeps record when you log in) |
| Screen Resolution… | Opens the Screen Resolution window (use this function to change the screen resolution) |

**Database Menu**

The Database menu has four items implemented: Speciality, Wound, Physician, and Patient.


*Figure I.3* Database Menu

Table I.3 *Purpose of Database Menu*

| Menu | Purpose |
|---|---|
| Specialty… | To add, edit and delete a Specialty Record |
| Wound… | To add, edit and delete a Wound Record |
| Physician… | To add, edit and delete a Physician Record |
| Patient | To add, edit and delete a Patient Record |

**Help Menu**

The Help menu has four items implemented: User Guide, Contents, Index and About.


*Figure I.4* Help Menu

Table I.4 *Purpose of Help Menu*

| Menu | Purpose |
|---|---|
| User Guide… | Quick Tutorial |
| Contents… (F1) | Displays the Help file installed on your system and navigate to specific topics |
| Index… | To find information based on keywords that are associated with the topics. |
| About… | Shows the About window giving version and build information. |

**The ECU Icon**

The ECU icon is shown in Figure I.5

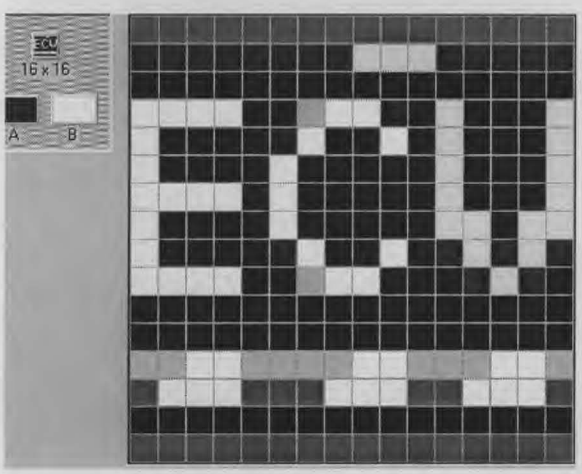



*Figure I.5* ECU Icon (16x16)

**The Database Icon**

The Database icon is shown in Figure I.6

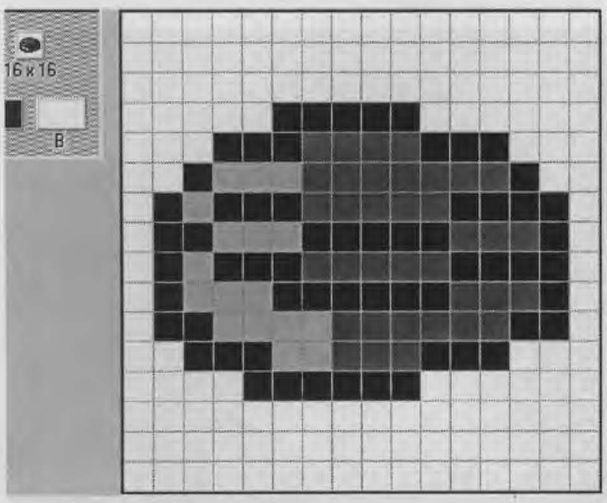



*Figure I.6* Database Icon (16x16)

**The Help Icon**

The Help icon is shown in Figure K.7

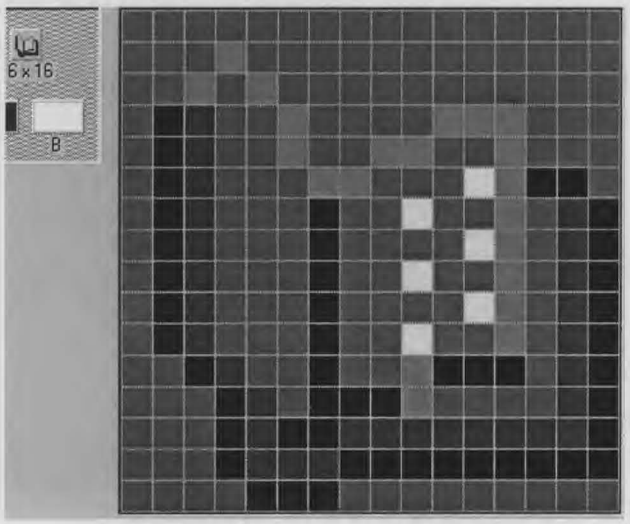



*Figure I.7* Help Icon (16x16)

**The Users Icon**

The Users icon is shown in Figure I.8

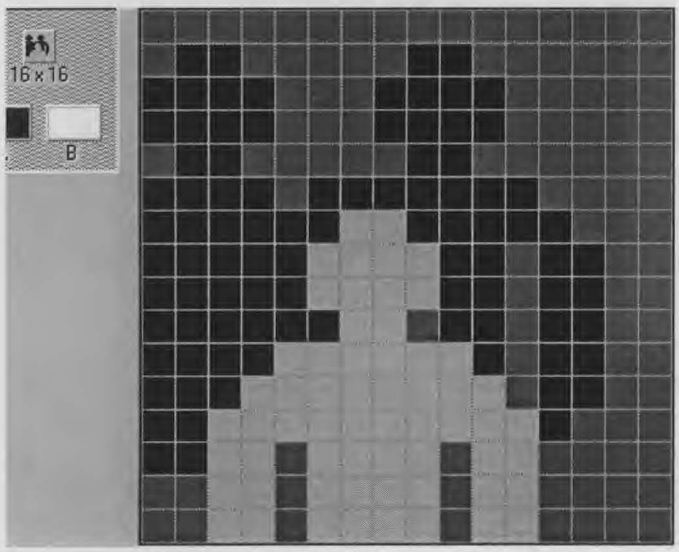



*Figure I.8* Users Icon (16x16)

## APPENDIX K: START-UP FLOWCHART

The following flowchart (see Fig. K.1) illustrates the process when the Wound Measurement System is started.

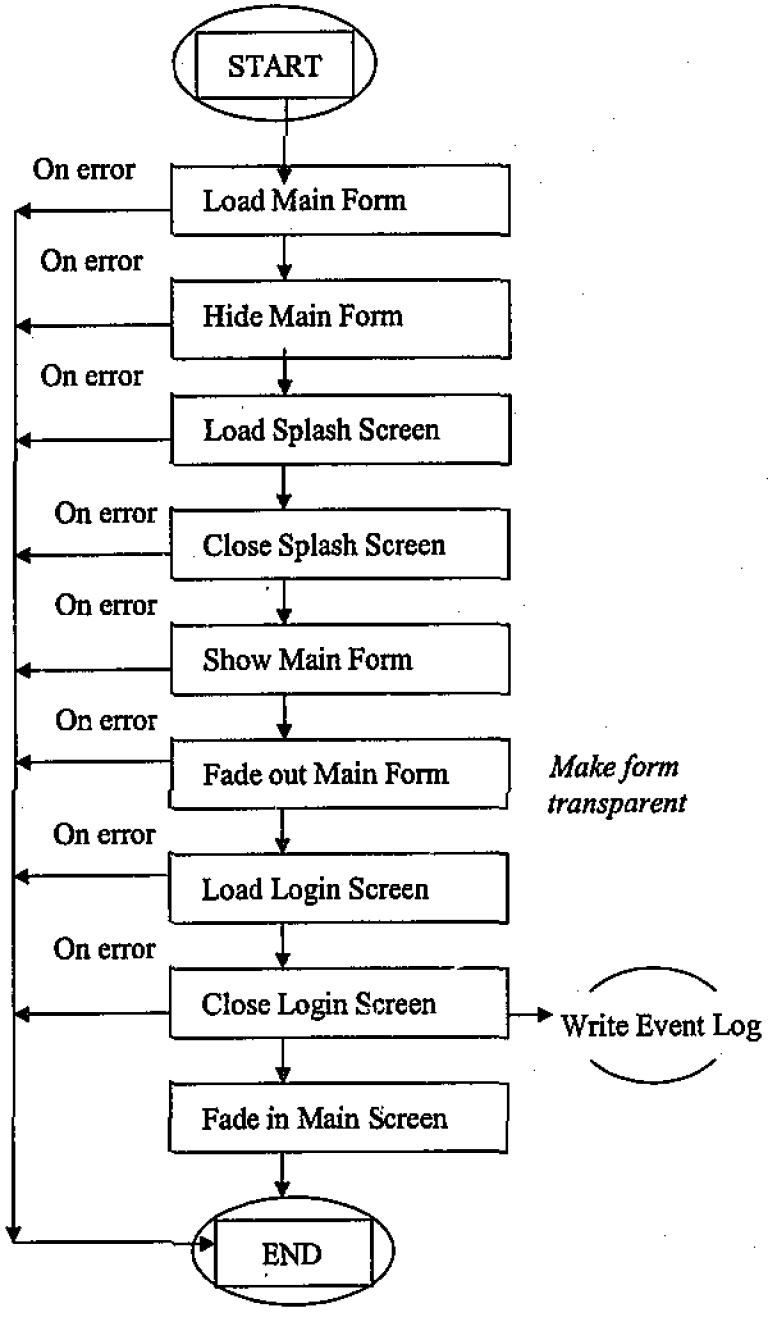



*Figure K.1* Application start-up procedures

# APPENDIX L: PROCEDURE LISTINGS

All procedures (source code) can be found on the CD-ROM accompanying the thesis copy held by Dr. Donngguang Li, MOUNT LAWLEY CAMPUS, email: d.li@ecu.edu.au.

## APPENDIX M: INSTRUCTIONS

To test the system, Visual Studio .NET should be installed on Windows XP Operating System.

All program files were installed and tested under the *c:\Wound Measurement\wound* directory on Windows XP.

To log on to the system:


| |
|---|
| Name: Administrator<br><br>Password: manager |

# DEFINITION OF TERMS

**Assembly**

An assembly is the primary unit and is composed of a manifest and one or more modules. The manifest contains information about the identity of the assembly. (Conard, Dengler, Francis, Harvey, Hollis, Ramachandran, Schenken, Short and Ullman, 2000, p. 37)

**Bitmap**

The native Windows bitmap file format, also known as the device-independent bitmap. (Petzold, 2003, p.475)

**Boolean**

A variable data type that allows the storage of a Boolean value. (Salvage, 2003, p.146)

**Button Control**

A control that acts as an action button. When depressed, the attached code will execute. (Salvage, 2003, p.51)

**Case**

A keyword that indicates the individual cases of a Select Case statement. (Salvage, 2003, p.146)

**Chronic**

1. lasting for a long time; happening continually
2. having had a disease or a habit for a long time

(Oxford Advanced Learner's Dictionary, 1995, p.197).

**Database**

A computer system designed to optimize the storing and accessing of large amount of data. (Salvage, 2003, p.12)

**Data Grid**

A control allowing the display, editing, addition, and deletion of data stored in a data set. (Salvage, 2003, p.435)

**Data Set**

A temporary representation of data from a database stored in the computer's memory. (Salvage, 2003, p.435)

**Double**

A variable data type that allows the storing of a decimal value with up to 14 digits of accuracy. (Salvage, 2003, p.86)

**Gif**

Format was developed in the late 1980s for the use on CompuServe(an early online information service) and remains one of the two most popular graphics formats on the World Wide Web. (Petzold, 2003, p.475)

**Image**

A stored description of a graphic picture, either as a set of brightness and colour values of pixels or as a set of instructions for reproducing the picture. (Microsoft Press Computer Dictionary, 1997, p.245).

**Image Processing**

The analysis, manipulation, storage and display of graphical images from sources such as photographs, drawings, and video. (Microsoft Press Computer Dictionary, 1997, p.245).

**Jpeg**

Pronounced "jay peg," JPEG stands for Joint Photographic Experts group, which is a collection of industry representatives who developed a family of compression techniques – some lossy, some lossless – specifically for continuos-tone still images. (Petzold, 2003, p.475)

**Menu Bar**

A serious of text shortcuts to commonly used routines in the development of an application. (Salvage, 2003, p.51)

**Module**

A module is either a DLL or an EXE Windows PE (Portable Executable). It contains Intermediate Language (IL – platform independent code).

(Conard, Dengler, Francis, Harvey, Hollis, Ramachandran, Schenken, Short and Ullman, 2000, p. 38)

**New**

A keyword used to create an object from a class. (Salvage, 2003, p.255)

**.NET**

Microsoft® .NET is a set of Microsoft software technologies for connecting information, people, systems, and devices.

.NET is infused into the products that make up the Microsoft platform, providing the ability to quickly and reliably build, host, deploy, and utilize connected solutions using Web services, all with the protection of industry-standard security technologies. (Microsoft Corporation, 2003).

**.NET Languages**

Languages such as Visual Basic .NET, Visual C++ .NET, C# and others.

**OleDbConnection**

An object that contains the specifications for the protocol to communicate with a database and any information pertaining to the connection. (Salvage, 2003, p.435)

**OleDbDataAdapter**

An object that provides the methods to transfer data from the database to the data set. (Salvage, 2003, p.435)

**Operand**

An entity upon which operators will perform their action. (Salvage, 2003, p.86)

**Perpendicular**

At an angle of 90° to another line or surface. (Oxford Advanced Learner's Dictionary, 1995, p. 862).

**Pixel**

One spot in a rectilinear grid of thousands of such spots that are individually "painted" to form an image produced on the screen by a computer or on paper by a printer. A pixel is the smallest element that display or print hardware and software can manipulate in creating letters, numbers, or graphics. (Microsoft Press Computer Dictionary, 1997, p.367).

**Public**

Defines a variable that is visible throughout the entire application

**Regression**

Regression is a statistical method used to predict values based on relationships in existing data. By analysing how a single dependent variable (y) is affected by the values of one or more independent variables (x), you can predict what y will be given x.

**Short**

A variable that allows the storing of an integer value between -32,768 and 32,767. (Salvage, 2003, p.86)

**Single**

The smallest variable data type that allows the storing of a decimal value with up to 6 digits of accuracy (Salvage, 2003, p.86)

**SQL**

Structured Query Language (SQL) was developed in the early 1970s at IBM for use with relational databases. VB.NET uses a version of SQL that is compliant with ANSO-92 SQL. (Schneider, 2003)

**Step**

The amount by which the FOR loop index is incremented on each iteration of the loop. (Salvage, 2003, p.301)

**Tiff**

The Tag Image File Format was originally developed by Aldus (creators of the popular PageMaker application) and Microsoft, and the specification is now owned by Adobe. (Petzold, 2003, p.476)

**Type**

A type is a template used to describe the encapsulation of data and an associated set of behaviours.

(Conard, Dengler, Francis, Harvey, Hollis, Ramachandran, Schenken, Short and Ullman, 2000, p. 38)

**Variable**

A way the computer stores a value in memory. (Salvage, 2003, p.86)

**Visual Basic.NET**

The version of Visual Basic following Visual Basic 6.0 (programming language) is Visual Basic .NET.

**Visual C++.NET**

The version of Visual C++ following Visual C++ 6.0 (programming language) is Visual C++ .NET.

**Wmf**

It is a format for a metafile, which is a collection of drawing functions (generally vector drawing functions) stored in binary form. (Petzold, 2003, p.475)

## References

Conard, J., Dengler, P., Francis, B., Glynn, J., Harvey, B., Hollis, B., Ramachandran, R., Schenken, J., Short S., Ullman, C. (2000). *Introducing .NET.* Birmingham: Wrox Press.

Dealey, C. (1994). *The care of wounds: a guide for nurses.* Boston: Blackwell Scientific Publications.

Deitel, H., M., Deitel, P.J. & Nieto, T.R. (2002). *Visual Basic .NET: how to program (2nd ed.).* New Jersey: Prentice Hall.

Evjen, B. & Beres, J. (2002). *Visual Basic .NET Bible.* New York: Hungry Minds.

Goldman, R.J. & Salcido, R., (2002). More than One-Way to Measure a Wound: An Overview of Tools and Techniques. *Skin & Wound Care, 15 (5), 236 – 243.*

Halvorson, M. (2003). *Microsoft Visual Basic .NET Step by Step.* Redmond: Microsoft Press.

Hess, C. T. (2002). *Wound Care: Clinical Guide (4th ed.).* Springhouse: Springhouse, Pa.

Krasner, D. (1990). *Chronic wound care: a clinical source book for healthcare professionals.* Pa.., USA: Health Management Publications.

Krouskop, T.A., Baker, R. & Wilson, M.S. (2002). A noncontact wound measurement system. *Journal of Rehabilitation Research and Development, 39(3), 337-345.*

Langemo, D.K, Melland, H., Hanson, D., Olson, B., Hunter, S. and Henly, S.J. (1998). Two-Dimensional Wound Measurement: Comparison of 4 Techniques. *Advances in Wound Care, 11(7), 337 - 343.*

Microsoft Corporation (2003). *What is .NET? [on-line].* Available at WWW: http://www.microsoft.com/net/basics/whatis.asp [2003, 10 May].

Microsoft Press Computer Dictionary (1997). (3rd ed.). Redmond: Microsoft Press.

MSDN (2003). *[on-line].* Available WWW: http://msdn.microsoft.com [2003, 20 November]

National Institutes of Health (2000). *Topical Application of a Protein Heals Wounds.* [on-line]. Available WWW: http://www.nih.gov/news/pr/oct2000/nidcr-01.htm [2003, 25 April].

Ovington, L., G. (1999). *Planimetry is best way to measure wounds [on-line].* Available WWW: http://www.woundcare.org/newsvol4n2/prpt1.htm [2003, 22 April].

Oxford Advanced Learner's Dictionary (1995). (5th ed.). Oxford: Oxford University Press.

Petzold, C. (2003). *Programming Microsoft Windows with Microsoft Visual Basic .NET.* Redmond: Microsoft Press.

Prosise, J. (2002). *Programming Microsoft .NET.* Redmond: Microsoft Press.

Salvage, J. (2003). *The Visual Basic .NET Coach.* New York: Addison-Wesley.

Schneider, I. D. (2003). *An Introduction to Programming Using Visual Basic .NET (5th Ed.).* New Jersey: Prentice Hall.

Sebesta, W. R. (2002). *Concept of Programming Language (5th Ed.).* Boston: Addison-Wesley.

Stephens, R. (2000). *Visual Basic Graphics Programming, Second Edition: Hands-on Applications and Advanced Colour Development.* New York: John Willey & Sons.

Sussman, C. (1997). Utility of the Sussman Wound Healing Tool in Predicting Wound Healing Outcomes in Physical Therapy. *Advances in Wound Care, 10(5), 75-83.*

Thomas, D. R. (1997). Existing Tools: Are They Meeting the Challenges of Pressure Ulcer Healing? *Advances in Wound Care, 10(5), 86-90.*

Xakelliss, G.C. & Frantz, R.A. (1997). Pressure Ulcer Healing: What is it? What influences it? How is it measured? *Advances in Wound Care, 10(5), 20-26.*