TestLink

Software for connecting laboratory equipment and automation of workflow.

TestLink
Mjukvara för att koppla samman labutrustning och automatisera arbetsflöde.

Martin Westberg

Faculty of Health, Science and Technology
Department of computer science
C-level thesis 15 hp
Supervisor: Donald F Ross
Examiner: Thijs Jan Holleboom
Date of opposition: 2015-01-15
Abstract

Software development is of great significance in manufacturing and process industry. Software can, for example, make production more efficient and monitoring easier. In paperboard industry testing the product, that is, paperboard, is important. Testing is done to make sure that the product meets the requirements of the customers who transform the paperboard to packaging products. In the Skoghall paperboard Mill operators make use of a range of different test methods. Since the work pace is high and the product is being continuously tested improvements of the workflow at the laboratory are always of interest.

The aim of this project is to develop a program that connects test instruments at Skoghall Mill’s test laboratory to the Mill’s network. The motivation is to automate part of the work process by removing the step of manually reading measurement values from the test instruments. The scope of this project includes developing software that connects one of the instruments (the Short Span Compression tester [1] or SCT) at the laboratory to the Mill’s network. This program, named TestLink, will work as a proof of concept for connecting instruments using a common interface. The software is constructed using a modular approach forming a platform where an additional instrument can be connected by creating a new module. The software is developed in Visual Studio using C# as the programming language. The software architecture follows the MVVM [2] (Model View ViewModel) design pattern paradigm and WPF [3] (Windows Presentation Foundation) is used to construct the graphical user interface. The result shows that time savings can be made when automating part of the work with the introduction of new software.
## Contents

1. Introduction .................................................................................................................. 9  
   1.1. Disposition ................................................................................................................. 12  
2. Background .................................................................................................................... 14  
   2.1. Introduction.................................................................................................................. 14  
   2.2. The paperboard industry – production and market .................................................... 15  
   2.3. Testing paperboard ..................................................................................................... 16  
   2.4. Working Methods ..................................................................................................... 19  
      2.4.1. Previous Working Methods .................................................................................. 19  
      2.4.2. Current Working Methods .................................................................................. 19  
   2.5. Future working methods ............................................................................................ 20  
   2.6. Tools for creating the program .................................................................................. 22  
3. Design .............................................................................................................................. 27  
   3.1. Introduction.................................................................................................................. 27  
   3.2. Gathering user requirements about the project. ....................................................... 27  
      3.2.1. User stories ......................................................................................................... 27  
      3.2.2. Work flow .......................................................................................................... 30  
   3.3. Designing the graphical user interface ....................................................................... 31  
   3.4. Communication with external devices. ...................................................................... 39  
      3.4.1. Input devices ....................................................................................................... 40  
      3.4.2. Output devices .................................................................................................... 43  
   3.5. Modularity .................................................................................................................. 44  
4. Implementation ............................................................................................................... 45  
   4.1. Introduction.................................................................................................................. 45  
   4.2. Software architecture overview .................................................................................. 45  
   4.3. More about MVVM .................................................................................................... 46  
   4.4. View classes – the graphical user interface ............................................................... 52  
   4.5. ViewModel classes – the logic behind the user interface .......................................... 54  
   4.6. An example to illustrate the parts of the program ..................................................... 56  
   4.7. Model classes – the data structures ......................................................................... 61  
   4.8. Communicating with external devices ....................................................................... 63
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.8.1. Input devices</td>
<td>63</td>
</tr>
<tr>
<td>4.8.2. Output devices</td>
<td>65</td>
</tr>
<tr>
<td>4.9. Testing</td>
<td>66</td>
</tr>
<tr>
<td>5. Results</td>
<td>67</td>
</tr>
<tr>
<td>5.1. Introduction</td>
<td>67</td>
</tr>
<tr>
<td>5.2. Evaluation of the program</td>
<td>67</td>
</tr>
<tr>
<td>5.3. Evaluation of the work process at the laboratory</td>
<td>70</td>
</tr>
<tr>
<td>5.4. Working methods in retrospect</td>
<td>71</td>
</tr>
<tr>
<td>5.5. Bugs and problems</td>
<td>73</td>
</tr>
<tr>
<td>5.6. Training</td>
<td>76</td>
</tr>
<tr>
<td>6. Conclusion</td>
<td>77</td>
</tr>
<tr>
<td>6.1. Summary of the main conclusions</td>
<td>77</td>
</tr>
<tr>
<td>6.2. Reflections</td>
<td>77</td>
</tr>
<tr>
<td>6.3. Future work</td>
<td>78</td>
</tr>
<tr>
<td>6.4. Concluding remarks</td>
<td>78</td>
</tr>
<tr>
<td>References</td>
<td>79</td>
</tr>
<tr>
<td>Appendix A. List of user stories</td>
<td>82</td>
</tr>
<tr>
<td>Appendix B. TextFileCreator.cs</td>
<td>86</td>
</tr>
<tr>
<td>Appendix C. Additional classes</td>
<td>88</td>
</tr>
<tr>
<td>Appendix D. User guide</td>
<td>89</td>
</tr>
</tbody>
</table>
List of figures

Figure 1 - Manual versus automated measurement........................................... 12
Figure 2 - Two examples of packaging products made from paperboard.......... 14
Figure 3 – Drawing of a paperboard machine .................................................. 15
Figure 4 – A SCT instrument ........................................................................... 17
Figure 5 - The sample stage of a SCT-instrument seen from above............... 17
Figure 6 - A ZD tensile tester ........................................................................... 18
Figure 7 - The sample stage of a ZD tensile tester instrument ....................... 18
Figure 8 --An overview of the system.............................................................. 21
Figure 9 – First draft of the graphical user interface ......................................... 22
Figure 10 - An illustration of the MVVM pattern............................................... 24
Figure 11 – An early draft the graphical user interface ...................................... 28
Figure 12 - The previous user interface ............................................................. 32
Figure 13 - An overview on how positions are composed. .............................. 33
Figure 14 - A draft of a new user interface...................................................... 34
Figure 15 - An illustration of an invalid measurement...................................... 35
Figure 16 - Portion of the user interface in its initial state ......................... 37
Figure 17 - Portion of the user interface where an invalid tambour id is entered.
......................................................................................................................... 37
Figure 18 Portion of the user interface where a correct tambour id and
signature is entered. Notice that the start test button is activated................. 38
Figure 19 – Header part of the user interface. Test is started......................... 38
Figure 20 - Header part of the user interface after send has been pressed.... 39
Figure 21 - An overview of all the devices and units that the program
communicates with. ......................................................................................... 39
Figure 22 - The barcode label is scanned directly into the text field representing
the tambour number. ...................................................................................... 41
Figure 23 - The barcode scanner used in conjunction with the program. ...... 42
Figure 24 - Changing the number of positions ............................................... 43
Figure 25 - An overview of how the modules are called via the IViewModel
interface ......................................................................................................... 46
Figure 26 – User interface of the compiled program from the XAML example
code above. .................................................................................................. 47
Figure 27 - The binding process in MVVM...................................................... 48
Figure 28 – The user interface of the compiled program from the source code
in the example above ...................................................................................... 51
Figure 29 – Databinding between elements in the user interface (View class)
and properties in the model class ................................................................. 52
Figure 30 - Separation of Views in the User interface ..................................... 52
Figure 31 - The compiled program from the code in the example above. ...... 54
Figure 32 - An illustration of the program architecture............................................... 55
Figure 33 - Databinding between two textboxes in the view class and their corresponding properties in the view model................................................................. 56
Figure 34 - The sequence for handling data from a test instrument......................... 57
Figure 35 - Databinding between the list in the modules model class and a datagrid in the modules view class................................................................. 59
Figure 36 – Complete measurement series values sent to the database for process data............................................................................................................. 60
Figure 37 - Data structure for storing measurement values.................................... 62
Figure 38 - The final graphical user interface for the SCT instrument.................. 67
Figure 39 - SCT instrument and test setup screen (right) for testing the program. ............................................................................................................. 71
Figure 40 - The datagrid before pressing the delete key........................................... 74
Figure 41 - The datagrid after pressing the delete key............................................. 74
List of tables

Table 1 - The protocol for a byte array from the SCT instrument. .................. 40
I would like to thank my supervisor Donald F Ross at Karlstad University for proofreading, help with language and inspiration when writing this thesis. I would also like to thank my supervisor Johan Lundin at Altran for help with planning and implementing the software written in this project. I am also very grateful for all the help from Stora Enso Skoghall Mill. Thank you Peter Kvitle, Kajsa Roos, Kjell-Åke Andersson and the staff at test laboratory for all valuable input! Thank you Jenny Gustafsson for proofreading and help with structuring.
Details about the production process and procedures at Skoghall Mill, both figures and text, have been anonymized in this thesis.
1. Introduction

Software development is important in the manufacturing and process industry. Today software and computers are often an integrated part of industrial processes. However, there is still room for improvement in different parts of the production process. One illustration of this is the German government’s initiative “Industry 4.0” [4]. The aim of the initiative, first launched in 2011, is to computerize the manufacturing industry in a similar way that the idea behind “internet of things” [5] connects everyday items to the internet. Connecting parts of a manufacturing process to the internet or an intranet makes it easier to track use of resources and to follow logistic processes. Another advantage of further computerizing manufacturing processes is that it can contribute to a better and more efficient work environment. The controls of different machines can, for example, be integrated and operated from a central control room.

Another part of an industrial process that can be improved by computerization and automation is the test stage. Most of the products created in the manufacturing and process industry are usually tested. The testing can be done for example by measuring the tensile strength of a material, the color of a fabric or the tint of a windshield for a car. The purpose of testing is to check that the product meets its specification (a piece of paperboard must for example be able to withstand a certain amount of force before it tears). Often when testing a product or material, whether it be plywood or fabric or paperboard, a variety of equipment is used to test different characteristics. One challenge is to connect the different testing instruments to the production plant’s network. This challenge comes from the fact that different test instruments might have different ways of communicating with the outside world. One type of instrument may connect via a serial port [6] while another type might connect via USB port [7]. Even if two instruments share the same type of connection
they may still use different protocols to arrange the data sent over the connection. Thus, there is no standard way to connect the test instruments to a network.

The aim of this project is to automate part of the work process at a test laboratory of a paperboard mill. The goal of the project is to create a program that acts as a common interface between test instruments at the laboratory and the paperboard mill’s database for process data. All measurement values produced at the mill are stored in this database. Values stored in the database are later used to supply information about the quality of the paperboard to the mill’s customers.

The program should also be able to display measurement values to the user in a coherent way. The program created in this project will be able to communicate with the SCT test instrument which is one of the instruments in the test laboratory. This program will work as a proof of concept for connecting the test laboratory’s instruments to the database for process data. Further, the ambition with the development of the program is that it shall work as a platform where additional instrument can be connected to the database in the future. This provides all the instruments of the laboratory with a common interface for communicating with the database. By introducing the program the step of manually reading the results from the test instruments can be removed. Instead the results are written directly to the database by the program, as seen in figure 1.
1.1. Disposition

The thesis consists of six chapters. After this introduction follows chapter two which gives the background and context of this project. Chapter two describes the paperboard industry in general and how paperboard is tested and why testing is of great significance in the production of paperboard. Further, chapter two covers the motivation behind this project and how the introduction of new software can automate, and improve, part of a work process. The chapter also gives an introduction to tools and design patterns used when creating the software. The following chapter three discusses the design of the software and the motivation behind the design. This chapter explains what kind of devices that the program has to communicate with and how the specifications of the software have been derived from its future users, that is, how information has been gathered from the operators in the laboratory. Chapter three also covers an analysis of the work flow in the laboratory. Chapter four covers how the design of the software is implemented. The chapter describes software architecture, code, classes and communication. Chapter four also illustrates how the different parts of the program are tied together.

The result of the project is presented in chapter five. The chapter consist of an evaluation of the software developed, that is, the program, as well as an evaluation of the work process at the laboratory. Chapter five also discusses the
working methods used in this project in retrospective. Lastly the chapter discusses some bugs and problems during the project. Chapter six concludes the thesis, summarizes the main conclusions and discusses some reflections made during this project. The concluding chapter also includes a section that covers potential improvements and expansions of the software.
2. Background

2.1. Introduction

This chapter describes the context for the project and gives a brief background about making paperboard and the paperboard industry in general. Paperboard is a raw material that can be transformed to packaging products, for example milk cartons. Figure 2 shows two examples of packaging products.

![Figure 2 - Two examples of packaging products made from paperboard.](image)

This chapter also discusses how and why paperboard is tested. Further an overview is given of some of the instruments at the test laboratory at Skoghall Mill and how this project can contribute to improving the workflow at the laboratory by automating part of the work process. The chapter finally introduces the tools, programming languages and methods used in the project.
2.2. The paperboard industry – production and market

The production plants in the paperboard industry in Sweden produce a variety of types of paperboard for a large number of products. The main raw material of paperboard is wood. The wood is first chopped and boiled with chemicals to produce pulp. The pulp has a semi-liquid texture somewhat like porridge. The pulp is then diluted with water and sprayed on a moving mesh at the wet end of a paperboard machine. Some of the water is drained from the mesh right away and the rest of the water is evaporated throughout the dry end of the paperboard machine. A typical production rate of a modern paperboard machine is around 1000 meters per minute. The production is continuous and the finished paperboard is rolled up on a tambour\(^1\) at the end of the machine. Figure 3 depicts a drawing of the basic components of a paperboard machine.

![Figure 3 – Drawing of a paperboard machine](image)

Paperboard is a very broad term since there are a number of different types of paperboard. A production unit for paperboard has a number of parameters to adjust to alter the characteristics of the paperboard. One example is that the degree of grinding of the pulp changes characteristics such as the internal strength of the paperboard. Another example is the ability to change the thickness of the paperboard by varying the amount of pulp used. The uses for different types of paperboard can vary widely. While one type of paperboard may be resistant to liquids another type may be suited for high quality printing.

---

\(^1\) A tambour is the drum where the paperboard from a paperboard machine is wound onto. A tambour is eight meters in width and contains several thousand meters of paperboard.
The one aspect that they all have in common is that they are used for packaging other products. While the paper industry is in a recession [8], due to the fact that more and more people read news and articles on the internet and consequent fall in demand for newsprint, the paperboard industry is different. The consumption of paperboard is high thanks to the variety of products using paperboard for packaging and protection. The competition for market share for packaging mostly comes from plastic and glass manufacturers.

2.3. Testing paperboard

When we pick up a carton of milk in the store we seldom think of all the testing and measuring that the paperboard in the carton has undergone before it ends up on the shelf. Testing the paperboard is highly important however. If there, for example, is not enough glue in the paperboard the carton may leak. Furthermore if the paperboard is not stiff enough the carton may collapse when you try to lift it or when it is stacked under transportation. To prevent this, a number of tests, both physical and chemical, are carried out to make sure that the paperboard is up to standard for its purpose. Some of the data about the paperboard is gathered directly in the production process. A variety of sensors are placed along the paperboard machine to continuously record properties like moisture levels and optical characteristics of the paperboard. After the production phase a number of samples from the newly produced paperboard are taken to the laboratory of the production mill for further testing.

A typical test laboratory at a paperboard mill has a number of instruments to test and measure the paperboard. An example of testing equipment is the SCT (Short span Compression Test) tester [1] (Figure 4). The program developed will act as an interface between the SCT-instrument and the database for process data. The implementation of the program will serve as a proof of
concept for connecting test instruments in the laboratory to the database for process data.

The SCT instrument measures the compressibility in the length direction of the paperboard. To use the instrument 15 mm strips are cut from the paperboard. The strips (marked with beige color in the figure) are placed between two clamps of the sample stage as seen in figure 5. The clamps hold and compress the strips according to the arrows in figure 5. The result from the measurement represents the force necessary to compress a strip a predefined number of millimeters. With the results from SCT testing it is possible to predict the stacking capability of packages produced from the paperboard that the sample was taken from.

Another example of paperboard testing is the ZD Tensile tester [9] shown in figure 6.
The purpose of ZD tensile testing is to measure the internal strength in the thickness direction of the paperboard. The internal strength is relevant when the paperboard is folded into packages. Figure 7 shows the measurement procedure. A double-sided adhesive tape is applied to front and the back surface of a paperboard sample. The taped board sample is placed between two metal plates which are pressed together. The result from the tensile tester represents the force required to tear the paperboard sample apart in the direction of thickness.

Apart from the test methods recently described, a typical test laboratory in the paperboard industry uses about a dozen other instruments to test the paperboard. The results from the tests have a number of purposes. One example is process control. Every type of paperboard has a specification (for instance thickness and weight per square meter) that the produced paperboard must live up to. If the tests show that the paperboard does not meet the specifications, adjustments to the production process can be made. This is not such a simple task as it sounds. Often when adjustments are made to change one property other properties may be affected as well. Another goal of testing is to supply information to the customers who buy the paperboard to convert it
to packaging products like milk cartons and food containers. This information is vital to the customers since their converting machines\(^2\) require a paperboard with certain characteristics in order to function optimally. This, of course, also affects the end product. A third goal is to build up an archive of production data which is useful in research and development of production of paperboard.

2.4. Working Methods

2.4.1. Previous Working Methods

The test laboratory at the Skoghall Mill has a number of instruments to test the paperboard produced at the mill. Previously some of the instruments were connected to Skoghall’s database for process data. The purpose of the database is to store and organize measurement values for all the measured properties of the paperboard. The connection between the instruments and the system was made possible with a separate program for each instrument. Each program allowed measurement values to be transferred from the instruments to the database and also made it possible for the operators to monitor the test sequence.

2.4.2. Current Working Methods

However, over time some of the old instruments have been replaced with newer and more modern versions. When new instruments were introduced they had a different communication protocol. The programs could not handle the new protocol and the operators at test laboratory now had to manually enter measurement values, printed on receipts by the instruments, in a program connected to the database. There are several disadvantages to this. The most obvious one is that it takes extra time to manually enter the values but it also

\(^2\) A converting machine converts paperboard to packaging products such as milk cartons and food containers.
increases the chance that an operator makes a mistake when the values are entered.

2.5. Future working methods

The purpose of this project is to create a new program for the SCT-instrument (see Figure 4 and 5) at the test laboratory. This program will act as a link between the instrument and the paperboard mill’s database for process data and have a graphical user interface where the operators can monitor the measurements. The program shall work as a proof of concept for a platform that can accommodate additional instrument like the ZD tensile tester (see Figure 6 and 7) and other test instruments at the laboratory. New instruments can be added by simply writing a module (covered in section 3.5) for the specific instrument. One disadvantage with the previous programs used at the laboratory was that they were individually written to suite a specific test instrument, requiring a re-write of the program if the instrument was upgraded or the construction of a new program when a new test instrument was introduced. This disadvantage is overcome by creating a modular program that, with new modules, can be able to handle all the instruments at the test laboratory. An additional advantage with creating a modular program is that if a test instrument is upgraded, it can still be used with the program as long as the module for the instrument is updated. The program connecting the SCT-instrument will, apart from transferring measurement values, work as a prototype for a common interface program that can be used for additional instruments in the future. Figure 8 shows an overview of the system including the main program (pink dotted line) and the components with which it communicates.
As seen in figure 8 the test instruments communicate, depending on how modern they are, either over the RS-232 [6] serial port or an Ethernet [10] port (the SCT-instrument in this project communicates with RS-232 serial port).

Further the program must be able to read an id number from a barcode reader. The id number corresponds to the specific production number of a tambour. When a series of measurements for a tambour is complete the program is responsible for sending the values to Skoghall’s database for process data. The operator should also have an option to print the values on a printer on the local network.

One important component of the program is the graphical user interface (GUI). The purpose of the graphical user interface is to display measurement values as they are sent from an instrument. If a measurement value seems invalid the graphical user interface should alert the user, as seen in figure 9, by marking the value with a blue bold crossed over font. The operator should be able to edit the value by making a new measurement. It is important that the
graphical user interface is easy to read and not cluttered as the workload in the laboratory is high and several people are going to use the program.

![First draft of the graphical user interface](image)

**Figure 9 – First draft of the graphical user interface**

### 2.6. Tools for creating the program

When designing and creating the graphical user interface of a desktop application in Microsoft Visual Studio using C# usually either Windows Forms [11] or WPF [3] (Windows Presentation Foundation) is used. In this project the WPF toolkit is used to construct the graphical user interface. In WPF all graphical elements are represented by XAML [12] (Extensible Application Markup Language) code. Microsoft states the following: [12]

XAML is a declarative markup language. As applied to the .NET Framework programming model, XAML simplifies creating a UI for a .NET Framework application. You can create visible UI elements
in the declarative XAML markup, and then separate the UI
definition from the run-time logic by using code-behind files, joined
to the markup through partial class definitions. XAML directly
represents the instantiation of objects in a specific set of backing
types defined in assemblies. This is unlike most other markup
languages, which are typically an interpreted language without such a
direct tie to a backing type system. XAML enables a workflow
where separate parties can work on the UI and the logic of an
application, using potentially different tools.

When represented as text, XAML files are XML files that generally
have the .xaml extension. The files can be encoded by any XML
encoding, but encoding as UTF-8 is typical.

Because XAML is based on XML, it is easy to read for humans. The following
lines of code demonstrate how a button with the text “Save” is added to an
empty WPF window:

```xml
<Window x:Class="Demonstration.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Button Content="Save" Height="25"
                Width="100" />
    </Grid>
</Window>
```

The use of XAML to create the graphical user interface makes user controls like
buttons easy to customize. To add an image to a button, for example, the
XAML code for the button in the previous code example can be modified to
the following:

```xml
<Button Height="25" Width="100">
    <StackPanel>
        <Image Source="buttonBackground.png" />
        <TextBlock Text="Save" />
    </StackPanel>
</Button>
```
WPF is suitable for use in conjunction with an architectural design pattern called MVVM (Model View ViewModel) [2]. The MVVM design pattern endorses a clear separation between the graphical user interface, logic and data. The responsibilities of the parts of the design pattern are as follows:

- **The View:**
  The view is responsible for displaying all the elements in the graphical user interface of the program. Ideally, the View should be written exclusively in XAML without any logic.

- **The Model:**
  The model is responsible for storing the data of a program. It could be as simple as a list containing strings but it could also be a connection and logic for communicating with a database.

- **The ViewModel:**
  The ViewModel acts as a communicator between the View and the model. It should expose properties from the Model that the View can bind to and also expose commands containing logic that the view can utilize.

Figure 10 illustrates the MVVM pattern.

![Figure 10](image)

*Figure 10 - An illustration of the MVVM pattern*

To communicate between the different parts of an MVVM-WPF application databinding [13] is used. Databinding makes it possible to bind an element, like
a textbox in the View, to a property [14] in the Model. Properties can be seen as getters and setters for variables. Microsoft explains properties in the following way [14]:

A property is a member that provides a flexible mechanism to read, write, or compute the value of a private field. Properties can be used as if they are public data members, but they are actually special methods called accessors. This enables data to be accessed easily and still helps promote the safety and flexibility of methods.

The following lines of code describe how a property called name is created and called:

```csharp
class Person
{
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}
class Program
{
    static void Main()
    {
        Person p = new Person();
        p.Name = "Alf"; //set the name of p
        String personName = p.Name; //get the name of p
    }
}
```

When using databinding, the binding can be set up to update the property in both directions. If the value in the Model changes this is reflected in the View and if the user changes the value in the View this is reflected in the Model. It is also possible to use databinding to bind an element in the View to specific command in the ViewModel.
As previously mentioned, the View is created with XAML. The ViewModel and the Model are written in C# (pronounced C sharp) [15] using Microsoft’s .NET [16] framework. C# is an object oriented programming language created by Microsoft and has similarities with JAVA in programming style. Microsoft Visual Studio 2012 [17] is used as tool for writing the program. The test laboratory at Skoghall Mill had no preference regarding which platform to use when building the program. The Microsoft platform was chosen simply because of previous experience with it and the knowledge that it would fit the needs of the program. Particularly, the C# language has good support for communication over the serial port and lots of documentation thanks to its large user base.
3. Design

3.1. Introduction

This chapter discusses the design of the program. The first two sections cover how the user requirements for the project have been gathered. Section 3.3 covers how the graphical user interface has been designed and what factors that has been taken into consideration when doing so. Section 3.4 contains information about the devices that the program has to communicate with and what is required by the program to carry out the communication. Section 3.5 discusses an important feature of the program, namely modularity. The program should be easily expandable to connect to different instrument and act as a common interface against the database for process data for all desired instruments at the testing laboratory. A new instrument should be able to be added to the program by simply writing a new module.

3.2. Gathering user requirements about the project.

The following two sections discuss how the user requirements for the project were gathered. Section 3.2.1 covers the user stories\textsuperscript{3} from the interviews with the staff at the test laboratory. Section 3.2.2 describes an analysis of the workflow when using the test equipment at the laboratory.

3.2.1. User stories

To gain an understanding of what the customer requires, some of the operators at the test laboratory were interviewed. After performing some initial discussions, the operators were showed a draft of a graphical user interface (Figure 11).

\textsuperscript{3} A user story is a requirement about functionality or design of the program written in everyday language.
Figure 11 – An early draft of the graphical user interface

The appearance of the draft was loosely based on a previous program that was used to transfer measurement values from the instruments to the database for process data. The idea of basing the draft on a previous program was so that the users would feel familiar with the graphical user interface. After showing the draft to half a dozen operators and the management of the test laboratory, the operators were interviewed in detail. These interviews led to a number of user stories. The following list describes some of the user stories (see appendix A for a complete, numbered, list of user stories):

- User story 15:

  As a user I would like to be able to send measurement values from an instrument to the database for process data so that I do not have to manually enter the values in the database.
• User story 3:
  As a user I would like to be able view all individual measurement values of a measurement series at the same time so that I can easily detect any value that does not seem valid.

• User story 21:
  As a user I would like each measurement series to have as few steps as possible so that the time interacting with the program is kept at a minimum.

• User story 27:
  As an IT administrator I would like that the program is easily able to adapt to new business systems so that if the old system is changed to a new one the program can function with a minimal number of modifications.

• User story 23:
  As a user I would like to have validation of the sample ids that are entered in the system so that I only can enter allowed characters with an allowed format.

• User story 31:
  As a user I would like that measurement values that seem unreasonable (zero and negative numbers) are presented with a bold blue text in the user interface so that the unreasonable value is easy to spot and remove.

As seen in the list above some of the user stories are very general while others are more specific. When the list of user stories was finished a meeting was held with the management and my supervisor to determine how to prioritize implementation of the user stories. The priorities are partly based on the need for the specific functionality of a user story and partly on an approximation of the time it would take to implement a specific user story. During the course of this thesis, meetings with the management of the test laboratory where held. Each meeting, which took place twice a month, included a demonstration of the program and discussions of what the next step should be. Discussions were also held with the operators of the laboratory to make sure that they would also be included in the development process. The discussions, both in the meetings
and in discussions with the operators, led to a number of new user stories along the way. The full list of user stories can be found in appendix A. All implemented user stories are marked with a green square in the rightmost column and the unimplemented are marked with no color at all.

### 3.2.2. Work flow

Another important aspect when designing the program (and particularly the graphical user interface) was to analyze the workflow when using an instrument. What is, for example, the first thing a user does when starting a new measurement series? And is the sample sizes of a measurement series always the same or does it vary between different types of paperboard? The following eight steps describe the workflow, based on the user stories and discussions with the operators, in detail:

1. The first thing a user does when beginning a measurement series is to start the session and scan a barcode on a label, with a barcode scanner. The label is printed in another part of the production process and represents the id of the sample. If the label is missing the user should be able to manually enter the id.

2. The user enters their signature containing four characters. All users at the laboratory have a unique signature consisting of four characters (a-z,â,ä,ö not case sensitive).

3. After the id and signature have been entered the user must decide how many values that the measurement series consists of.

4. The user activates the measurement series in the program. This means that the user is finished with all the information concerning the identity and the format of the sample. At this point the user should not be able to change any information about the identity and format.

5. The user starts the instrument and feeds it with sample material.
6. The user monitors the measurement values as they appear in the graphical user interface. The user should at any time be able to cancel the test and start a new one.

7. When the test sequence is done the user should be able to review all the measurement values. If a measurement value is outside the specification limits the user can manually enter a new value based on a new measurement.

8. When the user has reviewed the test sequence and all values are correct the user should be able to send the measurement series to the database for process data. This should lock the measurement series (deactivation of buttons and user controls) so that the session is not sent, by accident, more than once. The user should have the ability to print the measurement series to a printer if necessary.

The user stories and the data from the workflow analysis are used as a specification of what the software should live up to.

3.3. Designing the graphical user interface

When starting the work with the graphical user interface the operators were asked what they thought about an old program that is connected to one of the other instruments. Figure 12 shows what the previous graphical user interface looked like:
They responded that the old program and its graphical user interface worked fine but they missed the ability to view all the measurement values from a measurement series at the same time. As seen in figure 12 the graphical user interface of the old program only showed the measurement values for one position at the time (position A1). A position is a subset of the whole measurement series. A position represents a portion of a newly produced tambour of paperboard. For confidentiality reasons the exact meaning of a position is left out of this thesis. What can be said is that each position maps to part of a newly produced tambour. When testing a tambour it is not enough to test a sample at a single position from the tambour (a tambour is very large, eight meters in width). The tambour is therefore divided into a number of positions. Positions is an important concept in the test laboratory since some types of paperboard requires more positions to be tested while other types require less. A measurement series may contain a number of positions and the

Figure 12: The previous user interface
number of positions may vary. Each position in turn consists of a number of individual test strips, cut out from the paperboard at the specified position, each one representing an individual value. Figure 13 shows a measurement series containing four (hypothetical) positions: A1, A2, B1 and B2. Each positions containing three individual measurement values.

![Diagram of positions and test strips]

*Figure 13 - An overview on how positions are composed.*

When the first test strip of three from position A1 is tested in the SCT tester the value produced should appear on the first row in the A1 column in the user interface as seen in figure 13. The second test strip should appear on the next row in column A1 and so on. The average value from the three test strips represents the value for the position. When all positions are filled with values, the sample is ready to be sent to the database for process data.

The request from the operators, to be able to see all positions at the same time, formed the following user story:
As a user I would like to be able view all individual measurement values of a measurement series at the same time so that I can easily detect any value that does not seem valid (user story 3).

The user story created a base for the look of the interface and how much information that was going to be displayed. Figure 14 shows the draft of the user interface that was produced, showed and discussed with the operators:

![Draft of a new user interface](image)

*Figure 14 - A draft of a new user interface.*

The main advantage of displaying all positions at the same time is that it makes it much easier for the operators to identify and edit values that do not seem valid. In the graphical user interface of the old program the operators could only see the position currently being tested. If, for example position A1 was tested, the operators could only see the three values representing the A1 position. To further assist the user to identify non valid values, the following user story has been implemented:
As a user I would like that measurement values that seem invalid (zero and negative numbers) are presented with a bold blue text in the user interface so that an invalid value is easy to spot and remove (user story 31).

Figure 15 shows a portion of the user interface and how the zero and negative values are marked with a bold blue strikethrough font:

![Figure 15](image)

Another user story that the operator particularly stressed was important was:

As a user I would like each measurement series to have as few steps as possible so that the time interacting with the program takes as little time as possible (user story 21).

The work pace at the laboratory is high and the operators do not want to spend unnecessary time on a program that is not simple to use. A source of inspiration when designing the user interface is Steve Krugs [18] *Don’t make me think*. Krug’s first law of usability states [18, p. 20]:

People often ask me:

“What’s the most important thing I should do if I want to make sure my app or site is easy to use?”

The answer is simple. It’s not ”Nothing important should ever be more than two clicks away” or “Speak the user’s language”
or “Be consistent.”

It’s...

“Don’t make me think!”

It is important that the addition of the program contributes to the work environment rather than the opposite. It is also important, in this case, to try to avoid irritating elements such as dialogue boxes popping up all the time. It is however important to prevent the users from carrying out certain operations as the following user story states:

*As a user I would like to have validation of the sample id’s that are entered in the system so that I only can enter allowed characters with an allowed format (user story 23).*

It would, for example, be bad if the user tries to transfer an empty or partially empty measurement series to the database for process data. It is also a good idea to check whether the entered sample id’s and signatures to be entered into the database have the correct format and use only valid characters (the sample id is used when inserting the measurement values in the database and the signature represents the user carrying out the test). To avoid using dialogue boxes, user input errors and notifications to the user are handled with activation and deactivation of controls. In this way information about the state of a test sequence is available to the user just by looking at the screen. When creating a new test the initial state of controls makes it impossible to start the test sequence until all necessary information about the sample is entered in a correct way. The user is also prompted to enter the required values as seen in Figure 16.
Figure 16 - Portion of the user interface in its initial state.

As seen in figure 16 all buttons except “Nytt prov” (new test) and ”Skriv ut” (print) are deactivated (“Skriv ut” is always active since it may be useful to print partially filled measurement series in some instances). This makes it impossible for the user to start a new measurement series without entering correct details. If the user enters a sample id that does not have the correct format⁴ as seen in the figure 17 the error text displays the message Felaktigt tambour-ID (invalid tambour id), informing the user that the format of the id is not valid:

Figure 17 - Portion of the user interface where an invalid tambour id is entered.

⁴ The correct format is xyyyy where x represents a number between seven and eight and each y represents a number between zero and nine. Ex. 712345 is correct 412345 is incorrect.
The “Starta provning” (start test) button remains inactive to prevent the user from starting a new test with a sample id that is not valid. When the user enters a valid sample id and a valid signature, the button to start the test is activated as seen in figure 18:

![Figure 18 Portion of the user interface where a correct tambour id and signature is entered. Notice that the start test button is activated.](image)

The user can now start the test. As soon as the test is started the text fields for “Tambournummer” (sample id) and “Signatur” (Signature) are locked as seen in the figure 19. This prevents the user from (by mistake) changing the id or signature during the test. The user is also prevented from starting a new test until the current test is done or the user cancels the test by clicking “Avbryt” (cancel).

![Figure 19 – Header part of the user interface. Test is started.](image)

When a test is complete the “Skicka” (send) button becomes activated. By clicking the button the user sends the test results to the database for process data. When the “Skicka” button is clicked it becomes deactivated. This prevents the user from sending the same test sequence more than once. The test results are still visible and printable if the transmission of the test results failed as seen in figure 20.
3.4. Communication with external devices.

The following two sections describe the external devices that the program has to communicate with. Section 3.4.1 describes the input devices and section 3.4.2 describes the output devices. Figure 21 shows all the devices that the program has to communicate with. The arrows indicate the direction of the communication:

![Diagram showing communication with external devices]

**Figure 21** - An overview of all the devices and units that the program communicates with.
3.4.1. Input devices

1. Test instrument.

The program is designed to capture measurement values from a SCT instrument. The measurement values are sent over the RS-232 serial port. The values are sent as a byte array. Each byte in the byte array contains two hexadecimal characters each one representing an ASCII character. The value 3.7 is for example represented as 0x30 – 0x2C – 0x37. Apart from the measurement value itself the byte array also contains characters for start of text (0x02), end of text (0x03), new lines (0x0A), instrument id and group separators. To make sure that the byte array is intact when it arrives at the other side of the serial port from the instrument a checksum is appended at the end of the byte array. The checksum is calculated by adding all characters from start of text to end of text. Table 1 shows the protocol for an array from one of the test instrument

<table>
<thead>
<tr>
<th>Byte</th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 1</td>
<td>STX (0x02)</td>
<td>Start of text (1 byte)</td>
</tr>
<tr>
<td>Byte 2-5</td>
<td>Instrument code (4 bytes)</td>
<td></td>
</tr>
<tr>
<td>Byte 6</td>
<td>Not used (1 byte)</td>
<td></td>
</tr>
<tr>
<td>Byte 7</td>
<td>Not used (1 byte)</td>
<td></td>
</tr>
<tr>
<td>Byte 8-n</td>
<td>GS(0x1d)</td>
<td>Group separator (1byte)</td>
</tr>
<tr>
<td></td>
<td>SP(0x20)</td>
<td>Space (1 byte)</td>
</tr>
<tr>
<td></td>
<td>SP(0x20)</td>
<td>Compensated measurement value (4 bytes)</td>
</tr>
<tr>
<td></td>
<td>SP(0x20)</td>
<td>Space (1 byte)</td>
</tr>
<tr>
<td></td>
<td>SP(0x20)</td>
<td>Moisture level (3 bytes)</td>
</tr>
<tr>
<td></td>
<td>SP(0x20)</td>
<td>Space (1 byte)</td>
</tr>
<tr>
<td></td>
<td>SP(0x20)</td>
<td>Moisture curve (4 bytes)</td>
</tr>
<tr>
<td></td>
<td>ETX(0x03)</td>
<td>End of text</td>
</tr>
<tr>
<td></td>
<td>CR (0x0d)</td>
<td>Carriage return (1 byte)</td>
</tr>
<tr>
<td></td>
<td>LF (0x0a)</td>
<td>Line feed (1 byte)</td>
</tr>
</tbody>
</table>

Table 1 - The protocol for a byte array from the SCT instrument.
2. Barcode scanner.

The barcode scanner is used to identify the ID of a paper board sample. The barcode scanner is connected with a USB cable and is identified, and interpreted by the computer, as a keyboard. In other words, if a code representing the numbers 1, 2, 3, 4, 5 is scanned it is interpreted in the same way as if a user had used a keyboard and pressed the buttons 1, 2, 3, 4, 5. This means that when a bar code is scanned the destination of the result is dependent on which application is currently active. If, for example, a barcode is scanned and a word processor is active the result from the scanner will be printed as text in the word processor. For this reason the scanned code from the barcode scanner is captured directly in the text field representing the tambour id as seen in figure 22.

![Barcode scan](image)

*Figure 22: The barcode label is scanned directly into the text field representing the tambour number.*

The barcode scanner shown in figure 23 was obtained to meet the specifications from the management of the test laboratory.
3. User input from the graphical user interface

Input is also collected from the graphical user interface. If the barcode label is missing or unreadable for a measurement series, the user must be able to manually enter the id for the measurement series. The user must also be able to supply a signature that is saved along with the measurement series data. Both the id and signature are entered in text boxes with a standard keyboard. The user can also enter additional information, such as number of positions to be tested, with the help of radio buttons. The number of positions to be tested is flexible (the number of positions to be tested varies between different types of paperboard). By clicking the radio buttons under “Antal positioner” (number of positions) the user can change the number of positions to be tested. Figure 24 shows an example where one position is selected and the number of positions to be tested is changed to A1 and B1. If the numbers of positions were set to two, the numbers of positions to be tested would be changed to A1, A2, B1 and B2. If three positions were selected the number of positions would be A1, A2, A3, B1, B2 and B3.
Finally the user can use a button to start, cancel, create a new measurement series, print measurement values and send measurement values to the database for process data.

### 3.4.2. Output devices

1. Database for process data.
   
   The measurement values from the instruments are stored in a database containing process data. Rather than accessing the database directly, text files are used to transfer the values. The text files are formatted in a specific way and, in addition to the measurement values, contain sample ID, signature, date and time.

2. Printer
   
   In order to get a paper copy of the measurement values the program has a print functionality. The printer is accessed through the local network.

3. Output to the graphical user interface
   
   The graphical user interface displays the measurement values that are sent from the test instrument.
3.5. Modularity

One of the goals with this program is to make it expandable to work with several different instruments. It should be easy to add new instruments, if needed, to the program. To accomplish this, each instrument is given a separate module in the program. In this project, a module for the SCT instrument has been developed. The module is responsible only for the functionality of the particular instrument that it represents. As much of the functionality that is shared between all instruments, such as user input of id and signature and reading raw data from the test instruments, is kept outside the modules. This way code duplication is kept down.
4. Implementation

4.1. Introduction

This chapter discusses the implementation of the program. Section 4.2 gives an overview of the software architecture. Section 4.3 contains more information, as well as a few code examples, about the MVVM design pattern and serves as an introduction to the subsequent sections. Section 4.4 discusses the view classes (the graphical user interface classes) of the program. Section 4.5 contains information about the view model classes (the classes behind the logic of the program). Section 4.6 contains an example of how a measurement value is transferred from the SCT test instrument to the program. Section 4.7 contains information about the model classes (the data storage classes of the program). Section 4.8 explains the implementation of the communication with external devices. The section is divided into section 4.8.1 which describes implementation of communication with input devices and section 4.8.2 which describes implementation of communication with output devices. Finally the last section (section 4.9) briefly discusses some of the testing methods used when creating the program.

4.2. Software architecture overview

The program is designed to be expandable to a number of different test instruments besides the SCT-instrument. The main purpose of the program - to capture measurement values - is however the same for all instruments. Although the purpose of the program is the same for all instruments the way that the data is collected and extracted from the instruments varies between the instruments. There are also variations concerning aspects like how the data is transferred from the program to the database for process data. To reuse as much code as possible each instrument has a separate module containing functionality for the instrument in question. The non-instrument specific code, like the logic behind components in the graphical user interface, is shared
between the instruments in a common part of the program. Since all modules have the same objective they all implement a common interface called IViewModel. This way the main program can call the same methods in a module regardless of what type of instrument it is. Figure 25 shows the idea behind how the modules for different instruments can be stored in a list and how the main program then can then call the currently activated module.

![Diagram](image)

*Figure 25 - An overview of how the modules are called via the IViewModel interface.*

### 4.3. More about MVVM

One goal with MVVM is to separate the graphical user interface from the logic of the program as much as possible. Nevertheless, the Views (the graphical user interface), ViewModels (the logic of the program) and models (the data structures and the data storage) must somehow be able to communicate with each other. To solve this, Commands [19] are used for communication. Commands are created in the ViewModels as properties (see chapter 2.5). The View can then subscribe to the commands using databinding. To be able to use
Databinding the View classes need a datacontext [20]. A datacontext can be described as an object that the View (graphical user interface) can communicate with. Usually the datacontext of a View is set to a ViewModel that is responsible for the logic behind the View. Setting the datacontext to a ViewModel means that the View can be bound to all the properties of a ViewModel. The following code shows an example on how databinding is used to bind a button in a View to a Command property in a ViewModel:

```xml
<Window x:Class="View"
    // The view model is added as a resource from the local namespace
    <Window.Resources>
        <local:ViewModel x:Key="ViewModel" />
    </Window.Resources>
    DataContext="{StaticResource ViewModel}"
    <Button Command="{Binding StartCommand}" Content="Start" />
</Window>
```

A reference to the ViewModel class is added as a resource to the View class on lines three to five. On line six the datacontext of the View is set to the reference of the ViewModel. This way the View has access to the ViewModel class. It is now possible to databind the button (line seven) to the property StartCommand in the ViewModel class. The content of the button represents the text that the button should display and is set to “Start”. The code above creates a window containing only a single button with the text Start as seen in figure 26.

*Figure 26 – User interface of the compiled program from the XAML example code above.*
The Views datacontext is set to ViewModel defined below. The buttons command is bound to a property called StartCommand in the class:

```csharp
1 class ViewModel
2 {
3 4 private ICommand startCommand;
5 public ICommand StartCommand
6 {
7    get
8    {
9        if (testAddValueCommand == null)
10           { testAddValueCommand = new RelayCommand(
11             p => this.startMethod(), //calls start method
12             p => true; //condition whether command is active or not
13           )
14           return testAddValueCommand;
15    }
16  }
17 18 private void startMethod()
19 {
20    //some code for starting
21    }
```

The Command property and how commands work is not explained in detail. The important part of the Command property is the code on line 12 where the command is set to call the startMethod() on line 19. The View can now access the StartCommand which in turn can call the startMethod in the class. Figure 27 shows the binding process:

![Figure 27 - The binding process in MVVM.](image-url)
Another useful feature of databinding is the ability to bind collections such as lists or simple properties to elements in the user interface. The following code example shows how to bind a list of strings in a model class to a datagrid [21] in a View class:

The model class:

The model is a simple class containing the data to be presented. It contains two properties: a list of person objects and a model type. Later in this example the View is going to bind two these two properties. The person class is an inner class that represents a person.

```csharp
namespace DatagridDemonstration
{
    public class Model
    {
        public List<Person> PersonList { get; set; }
        public string ModelType { get; private set; }

        public Model()
        {
            ModelType = "Persons model";
            PersonList = new List<Person>();
            PersonList.Add(new Person("Sara", 27));
            PersonList.Add(new Person("Olle", 34));
            PersonList.Add(new Person("Doris", 32));
        }

        public class Person
        {
            public string Name { get; set; }
            public int Age { get; set; }

            public Person(string name, int age)
            {
                Name = name;
                Age = age;
            }
        }
    }
}
```

The view model class:

The view models responsibility is to expose the model so that the view can bind to the model’s properties. The following lines of code show the ViewModel class:
namespace DatagridDemonstration
{
    public class ViewModel
    {
        public Model Model { get; set; }

        public ViewModel()
        {
            Model = new Model();
        }
    }
}

On line five a Model object is created as a property. The model is instantiated in the constructor on line nine.

The view class:

The view can now bind its elements to the properties in the Model. The lines of code below illustrate the concept of databinding components in the graphical user interface (the View class) to properties in the Model class (see also Figure 29). A reference to the view model is created as a resource on line nine. Then the grid's datacontext is set to the ViewModel on line 13. This means that all child controls of the grid can reference the ViewModel. The grid is a layout control (such as a stackpanel or dockpanel) where different child controls (such as labels and text boxes) can be placed. On line 16 a label is created. The label's content is bound to the ModelType property in the model. The binding keyword on line 16 simply means that the content of the label is subject to databinding. In plain text this means: set the content of the label to the ModelType property of the Model (or Model.ModelType). On line 18 a datagrid is created. A datagrid is an Excel-like control (see figure 28) for presenting data to the user. The datagrid is a flexible control that adapts to the number of elements in the collection that it binds to. If, for example, a datagrid is bound to a collection with three elements, the datagrid will display three cells where each cell contains one of the elements of the collection. The binding of the data grid is set to the persons list in the model on line 18.
The graphical user interface of the program from the compiled code is shown in figure 28.

Databinding makes displaying information from collections or single variables very easy. As seen in figure 28 above, when binding a collection to a datagrid, even the column names for name and age are created automatically. Figure 29 shows the concept behind databinding between elements in the view (the graphical user interface) and properties in the model:
4.4. View classes – the graphical user interface

The view classes represent the graphical user interfaces in the program. Each View class is responsible for displaying a defined part of the functionality to the user. All View classes are written purely in XAML (no C# code). The graphical user interface is separated into two distinct parts, as seen in figure 30.
One part (CommonView – pink dotted line) representing all the common controls of the program, and one part (ModuleView – blue dotted line) representing the active module. The following list describes the View classes.

- **CommonView**
  This class represents all the common functionality for the program. All buttons and field for inputting initials and ID:s are placed here. All buttons and input fields are databound to the CommonViewModel.

- **ModuleView**
  The ModuleView is responsible for displaying measurement values for the active instrument. The elements of the ModuleView are bound to the active modules Model class. The measurement values for the positions are each displayed in a customized data grid.

To position the components in the graphical user interface stackpanels, and to some extent, dockpanels and grids [22], are used. Because it is hard to fit the XAML code from the actual program in to a word document, the following piece of code illustrates a simplified version of how elements are laid out with XAML using stackpanels:

```xml
<StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
  <StackPanel Orientation="Vertical">
    <StackPanel Orientation="Horizontal" HorizontalAlignment="Right" Margin="0">
      <Label Content="Tambournummer" />
      <TextBox Width="100" BorderBrush="Black" />
    </StackPanel>
    <StackPanel Orientation="Horizontal" HorizontalAlignment="Right">
      <Label Content="Signatur" />
      <TextBox Width="100" BorderBrush="Black" />
    </StackPanel>
  </StackPanel>
  <Button Padding="5" Margin="5" VerticalAlignment="Center">Skicka</Button>
  <Button Padding="5" VerticalAlignment="Center">Nytt prov</Button>
</StackPanel>
```

When adding an element to a stackpanel the elements are stacked from top to bottom (vertical orientation) or left to right (horizontal orientation). Stackpanels
can also, like in the code above, be nested. The outer stackpanel between lines 1-14 contains several nested stackpanels inside it. Figure 31 shows how the stackpanels are distributed in the user interface of the program compiled from the code above. The dotted lines illustrate how the labels and text fields are laid out using stackpanels.

Figure 31 - The compiled program from the code in the example above.

4.5. ViewModel classes – the logic behind the user interface

The ViewModel classes are responsible for the logic behind the View classes (and the logic of the program in general). Each View class is supported by a ViewModel class. The communication between the Views and the ViewModels is done using Commands. The following list describes the ViewModel classes:

- **CommonViewModel:**
  The CommonViewModel is responsible for the logic behind the CommonView graphical user interface. The class exposes all the commands that the elements, such as buttons and text fields, in the CommonView class can bind to. The class also contains methods that correspond to each command. For example an addValue command calls an addValue method.

- **ModuleViewModel:**
  The ModuleViewModel is responsible for displaying and handling data from a specific instrument.
- **IViewModel:**

  All ModuleViewModels that represent a specific instrument implements the IViewModel interface. This way all modules must contain all the methods necessary for communicating with a test instrument.

Figure 32 shows the architecture for a simplified version of the program. To make the diagram as clear and readable as possible it demonstrates the principle behind the architecture rather than the architecture for the complete program.

![Diagram](image-url)

*Figure 32 - An illustration of the program architecture.*

Figure 32 shows how the CommonViewModel calls the currently active ModuleViewModel (the SCTViewModel) via the interface IViewModel and how the ValueList in the ModuleModel is bound to the datagrid in the CommonViewModel. Notice that the graphical user interface is divided into
two different classes, CommonView (pink dotted line) and SCTView (blue dotted line), each one databound to the common part of the program and the module part respectively (see also figure 30). Most of the methods that the interface requires to be implemented are left out for readability.

4.6. An example to illustrate the parts of the program

The pages in this section contain an example of how a test sequence with the SCT instrument is handled by the program.

The user first enters the tambour id and signature in the user interface as seen in figure 33:

![Diagram showing databinding between two textboxes in the view class and their corresponding properties in the view model.](image)

*Figure 33 - Databinding between two textboxes in the view class and their corresponding properties in the view model.*

The signature and tambour id (Tambournummer) labels in the CommonView are bound to the TambourId and Signature properties in the CommonViewModel. The signature and tambour id are validated. When the signature and tambour id are valid, the next step is to create a new ModuleViewModel of SCT type. When creating the new module the signature
and tambour id are passed to the module. The module now has the necessary information about the identity of the test sequence. When the module is created a help class called PortReader is activated in the CommonViewModel. The PortReader’s responsibility is to listen for incoming data at the serial port. More information about the PortReader class can be found in chapter 4.5.2.1. When new data arrives at the serial port (see figure 34), the PortReader class reads the data and converts it to a byte array. The class then signals the method addValueToCurrentModule() method in the CommonViewModel to execute. The addValueToCurrentModule() method calls the getLatestByteArray() on the PortReader class and sends the returned byte array to the SCTViewModel. The byte array sent to the SCTViewModel is the raw data from the instrument. It is the SCTViewModel class’s responsibility to extract the measurement value from the byte array.

![Diagram](image)

*Figure 34 - The sequence for handling data from a test instrument.*

Every byte array collected from the instrument has a specified structure (described in chapter 3.4.1) that includes a checksum. The algorithm for calculating the checksum is done by adding all bytes from STX (start of text) to
ETX (end of text). The addValue(byte[] valueArray) method in the SCTViewModel class’s responsibility is to control the checksum of the byte array and to extract the measurement value. The following pseudo code illustrates the functionality of the method as well as a method for controlling the checksum:

```
addValue(byte[] valueArray)
{
    if isChecksumOk(byte[] valueArray)
        Extract the measurement value (represented by byte 8-12 in the byte array)
        Combine the byte to a hexadecimal string
        translate the hexadecimal string to a base 10 decimal value
        Add the value to the model
    else
        do nothing //let the user do a new measurement
}

isChecksumOk(byte[] valueArray)
{
    Extract the byte representing the checksum from the byte array and translate them to a hexadecimal value
    Calculate the sum of the hexadecimal representation of the bytes in the array from STX to ETX.
    if the calculated checksum matches the checksum provided by the instrument
        return true
    else
        return false;
}
```

Thus, if the calculated checksum of the byte array matches the checksum supplied from the instrument the measurement value is accepted and the value is added to the list of measurement values in the SCTModel class (described more in detail in chapter 4.7). Since the list of measurement values (ValueList)
in the SCTModel is data bound to the datagrid in the SCTView, as soon as a value is added to the list it will also appear in the user interface. Figure 35 illustrates the databinding between the Model class and the View class:

As an example the value 5.3 is added to the list in the model in figure 35. When the value (originally from the instrument) is added to the model, the same value appears in the SCTView user interface. The databinding works in both directions: if the user changes or deletes a value in the user interface, this is reflected in the SCTModels ValueList. When the grid is filled with measurement values the values are ready to be sent to the database for process data. As seen in figure 36, when the last value in the series (4.5 in the figure), the user can send the values from the measurement series to the database for process data.
Figure 36 – Complete measurement series values sent to the database for process data.

The transfer of the values to the database is done by creating text files. The average values from the positions are placed in a text file and formatted in a specific way. The text files created by a help class called TextFileCreator (see section 4.8.2.1) are placed in a specific folder which is scanned on a regular basis by another program (created by the IT division at Skoghall Mill) that reads, and extracts the values, from the text file.

The example above does not, as previously mentioned, contain all the methods required to be implemented by the IViewModel interface for the sake of the clarity of the example. The following list contains all the methods that are required to implement the interface:

- void addValue(byte[] valueArray)
  
  Adds a new value to the currently active module in the form of a byte array containing the raw data from the serial port. The class implementing the interface is responsible for extracting a measurement value and to compare checksums for validity of the array.
• **void activateSession()**
  The method activates the measurement series and prepares the module for adding new measurement values.

• **void cancelSession()**
  Method for cancelling the session.

• **bool areListsFull()**
  Method for checking if all the lists are filled up with values. In other words if the measurement series is finished.

• **void changeNumberOfPositions(NumberOfPositions numberOfPositions)**
  The method changes the number of positions to be tested (changes the number of measurement values of a sample). The NumberOfPositions parameter is an enum help class for determining a predefined number of measurement values.

• **bool IsSessionActive()**
  Method for checking if the session is active or not.

• **bool IsSessionUsed()**
  Method for checking if the session has been used or not. The method is utilized by the graphical user interface logic.

• **void sendValuesToDatabase()**
  Method for sending the measurement series to the database for measurement values.

### 4.7. Model classes – the data structures

In MVVM the model classes are responsible for storing the application’s data such as data structures and database connections. The model classes store all the measurement values produced by the test instruments. Each position in a measurement series is composed of a number of individual measurement
values. For example, the first position represents the average of the first three values sent from the instrument, the second represents the average of values four to six and so on. Because the values from the instrument are sent in a sequence without the instrument knowing where a value belongs, a custom data structure is used. Each position stores the values in a list [23] where the values are represented in a custom class called DoubleItem. The DoubleItem contains, besides the actual measurement value, information about how the value should be presented in the graphical user interface. More information about the DoubleItem class can be found in Appendix C. The data structure has functionality for adding a single measurement value. If there are no other values present, the value is placed in the first slot of the first position. If there is already a value present at the first slot in the first position, the added value is placed in the second slot of the first position. If there are three values present in the first position and a new value is added, the value is placed in the first slot of the second position. Each position is represented by a list in the data structure. Figure 37 illustrates how the values of sequence 5.3 – 5.7 – 5.0 – 4.9 are placed in the custom data structure in the model:

Figure 37 - Data structure for storing measurement values.
The data structure also has functionality for calculating and returning average values for each position.

4.8. Communicating with external devices

The following two sections describe the implementation of the communication with the external devices that the program has to communicate with. Section 4.4.1 describes the input devices and section 4.4.2 describes the output devices.

4.8.1. Input devices:

4.8.1.1. Barcode scanner:

It is important that the scanned code for sample id has a correct format. If the format is not correct the database for process data cannot recognize the id of a specific sample and no data will be transferred. It is also important that no characters are present. To check validity of a scanned barcode a regular expression [24] is used. The following regex pattern is used to control a sample from the board machines:

```
^[7-8]{1}[0-9]{5}$
```

The pattern matches a string with the format xyyyyy where x represents a number between seven and eight and each y represents a number between zero and nine. In other words the patterns match a string that consists of five characters where the first character has to be seven or eight and the remaining characters have to be between zero and nine. The x-number represents the origin of the sample (8 = board machine eight and 7 = board machine seven) and the y number represents the id of the sample.
4.8.1.2. Test instrument:
The following section will cover the communication with the SCT instrument (described in chapter two) which is one of several instruments used at the test laboratory. The SCT instrument communicates over the RS-232 port. The .Net framework includes a serial port class [25] for communication over the serial port. The following line of code creates a new instance of the serial port class:

```csharp
SerialPort serialPort = new SerialPort(COM1, BAUD_RATE, PARITY, DATA_BITS, STOP_BITS);
```

The program uses a dedicated class (PortReader.cs) to handle all serial port communication with the instruments. The class is kept general by only processing the raw data that each instrument sends. The class is unaware of the contents of the data. The only responsibility that the class has is to read the bytes from the serial port place the bytes in a byte array and pass the array along to a receiving class for further processing. It is the receiving class’s responsibility to calculate and compare checksum and extract the measurement value from the raw data. The .Net serial port class has an event (DataReceived) that signals that new data has arrived at the serial port. The following line of code demonstrates how to connect a method called comPort_DataReceived to the DataReceived event:

```csharp
serialPort.DataReceived += comPort_DataReceived;
```

The comPort_DataReceived method’s job is to extract the bytes received from the serial port and place them in a byte array. The following lines of code show the implementation for the serialPort_DataReceived method:

```csharp
void comPort_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
    Thread.Sleep(100); //wait for all data to arrive
    int bytesToRead = serialPort.BytesToRead; //number of bytes to read
    byte[] byteArray = new byte[bytesToRead]; //create byte array
    serialPort.Read(latestByteArray, 0, bytesToRead); //reads the available bytes from the serial port.
}
```
4.8.1.3. User input:

To control that the signature is correct (four characters a-z,å,ä,ö) the following regular expression pattern is used:

`^[a-zA-ZåäöÅÄÖ]{4}$`

The pattern matches a string consisting of four characters where all the characters has to be between a-z,å,ä,ö (not case sensitive).

4.8.2. Output devices

4.8.2.1. Database for process data

To send a finished measurement series to the database for process data, text files are used. A text file, containing the result for a measurement series, is created and then placed in a specific folder and picked up by a program written by the IT department at Skoghall Mill. The program receiving the files interprets them and makes sure the values end up at the specified place in the database. The text files, containing comma separated columns, are formatted in a specific way with individual codes for each instrument. The following section describes the contents of a created text file:

```
1,sc_812345,txt,2014-11-25 20:35:06
3,7,12345,0,0,0
9,MAWE
11,1,0
12,7Two78,0,1,5.30
12,7Two77,0,1,4.70
```

Each row starts with a character indicating what type of row it is. The first row contains the name of the text file (sc_812345), the file ending (.txt) and the date. The second line contains the paper board machine that the sample is from along with the id of the sample. The third line contains the signature. The fourth line can be ignored. The fifth and sixth lines represent the number of
positions (two) and measurement values 5.21 and 3.79. The numbers 78 and 77 are used two identify the destination for the measurement values in the database for process data. Appendix B contains the source code for the complete class.

4.8.2.2. Printer

The user should have the ability to send the values from a measurement series to a printer. For this purpose a separate printer class has been created. The class is kept general to follow the modularity principle of the program. The printer class uses flow documents [26] to create the documents to be printed. The following lines of code illustrate how a flow document is constructed:

```csharp
2 flowDocument.Name = "PrintedDoc";
3 Run text = new Run("Example text");
4 Paragraph paragraph = new Paragraph(text);
5 flowDocument.Blocks.Add(paragraph);
```

The flow document is created on line one. The name of the flow document is set on line two. On line three a Run [27] object is created. An object of the run class is used to store inline text. The paragraph [28] created on line four is an object that is used for grouping text objects (such as run objects). On line five the paragraph is added to the flow document which now is ready to be printed.

4.9. Testing

Apart from the debugging done during the development phase of the program a few unit tests [29] were created. Unit tests were primarily used when creating the data structures and when creating a method for calculating the checksum of the byte arrays from the SCT instrument.
5. Results

5.1. Introduction

This section summarizes the work of this project. Section 5.2 discusses the result and evaluation of the program and section 5.3 covers the result and evaluation of the work process. Section 5.4 gives a description of how the work developed and the working methods in retrospect. The next section (5.5) discusses problems encountered during this project and describes some of the bugs in the program that have been dealt with. Section 5.6 finally covers the training that the operators have been given.

5.2. Evaluation of the program

The program is currently being evaluated at the test laboratory. The focus for this project has been on one of the instruments namely the SCT instrument (see Figure 4). Figure 38 shows the user interface of the finished program.

![Figure 38 - The final graphical user interface for the SCT instrument.](image)
Most of the highest prioritized user stories have been implemented. Most importantly the core functionality, to transfer values from the SCT-instrument to the database for measurement values has been implemented. The user can also view measurement values from all positions in a test sequence at the same time in the graphical user interface, something that was not possible in the previous programs used at the test laboratory. The request from the operators at the test laboratory to differentiate invalid values in the user interface with a font that stands out from valid values has also been implemented. Another design requirement by the operators was to make the program as easy as possible to use. This has been done by studying the workflow at the laboratory and to have continuous discussions with the operators of the test laboratory throughout the development process of the program. To minimize the number of clicks and number of steps when measuring a test sequence, inactivation of buttons and other controls in the graphical user interface is used in favour of dialogue boxes.

The program has been designed, as planned, to be able to include additional instruments in the future. The test laboratory now has a platform for connecting additional instruments to the database for process data. This has been accomplished by keeping as much as possible of the common functionality that can be shared with other instruments in a separate part of the program and the code used for instrument specific implementation is a separate switchable module. In this way the code required to add a new instrument is kept to a minimum. The major tasks when writing a module for a new instrument are the following:

- Functionality for extracting a measurement value and calculating the checksum from the byte array sent from a different instrument. The Byte arrays from the different instrument are totally different from each other.
• If an instrument uses an Ethernet connection instead of a RS-232 serial port connection the class for reading from the serial port will have to be replaced with a class with functionality for communicating over Ethernet.

In addition to the tasks above a few more minor adjustments have to be made in order to connect a new instrument. One example of additional work is that the definition of what an invalid value is might be different with a different instrument. While, for example, negative values are invalid for the SCT instrument, it may be perfectly valid for other instruments. This is done by defining what makes a value invalid in the DoubleItem class (see Appendix C) that stores individual measurement values. Another example of additional work is to add the identification codes tied to the instrument to be added. The codes are used when sending values to the database for process data to identify the instrument in question.

Some of the user stories have been left out of the program for time reasons. One example of an unimplemented feature is the ability to show how much a measured value deviates from the target specification in the graphical user interface. This user story was down prioritized at an early stage in the project. The main reason for this was that it was not possible to communicate directly with the database that contains the target values. A new program that would act as an interface against the database for target values was necessary to solve the problem. This would have taken too much time from the rest of the project. It was decided, in collaboration with the management of the test laboratory, that if this user story was to be implemented, it would be in a later version of the program. To conclude, most of the highest prioritized specifications of the program have been met: Of a total of 38 user stories, 31 or 82% have been implemented. All the user stories, implemented and unimplemented, can be found in Appendix A
5.3. **Evaluation of the work process at the laboratory**

The operators and management at the test laboratory are pleased with the result. The introduction of this program makes the work at the test laboratory more efficient. Using this program saves time. The time saving each day depends on how many tests are carried out on that particular day but anywhere from 10 – 40 minutes/day depending on the workload. It also takes less time from when the paperboard is produced until the values arrive in the database for process data which is important for the production process as a whole. This is important because the sooner the results become available to the operators of the board machine, the sooner the machine operators can make adjustments if the measurement values deviate from the target specification. This in turn can help Skoghall Mill with cost savings on production material (and finished product).

Apart from the time saving, the operators were pleased not to have to manually read the measurement values from the receipts from the instrument and enter them into the system. The fact that the operators will not have to enter the values manually with a keyboard helps avoiding stress to shoulders and elbows. Thereby the introduction of the program also improves the work environment. Lastly, by removing the step of manually entering the measurement values into the database for target values, the risk of adding wrong numbers by mistake has been reduced.
5.4. Working methods in retrospect

The task of writing the program has mostly been done outside of the laboratory. The reason for this is that the placement of the test instrument does not permit space for an extra work desk. To solve this, a computer was placed next to the instrument on a regular bench for the purpose of testing and modifying the program when connected instrument. Figure 39 shows the placement of the SCT instrument to the left and the screen of the test computer to the right:

![Figure 39 - SCT instrument and test setup screen (right) for testing the program.](image)

To share the code between different computers, and managing version control, Microsoft Team foundation server [30] has been used. This methodology has been developed during the course of this project and has worked well. Usually a piece of the program was written outside the laboratory and when the part was
finished it was tested in the laboratory against the SCT instrument. To be able
to do this the instruction manual and the communication protocol for the
instrument have been carefully studied. The data stream from the instrument
has also been well documented to be able to simulate the instrument when
outside the laboratory.

An agile [31] approach has been used when designing and developing this
program. The work has been divided in two weeks sprints. Every other week a
sprint planning meeting was held with my supervisor at Altran. The weeks
between the sprint planning, meetings were held with my supervisor to solve
problems and report work progress.

When writing this program some of the larger more general user stories like,

*As a user I would like to be able to send measurement values to the database for process data
so that I do not have to manually enter the values in the database (user story 15)*,

have been implemented simultaneously with other user stories during a number
of sprints. Other, smaller, user stories like,

*As a user I would like to have validation of the sample id:s that are entered in the system so
that I only can enter allowed characters with an allowed format (user story 23)*.

have been implemented during a single sprint focusing almost entirely on the
user story to be implemented.

Great care has been taken to include the operators that are going to use the
program as much as possible. The discussions with the operators have proven
very valuable. One example is when designing how invalid measurement values
should be presented to the user. The first draft of the design was not clear
enough to most of the operators. After the discussion a new, clearer, design was
presented and the operators were satisfied. Often when approaching an operator with a question, after a while, the discussion has steered into a completely different direction and new aspects, previously not thought about, have been discovered. This shows the importance of communication with the customer and the author’s opinion was that involving the operators in the production process will make them more positive to the introduction program. This also makes the work more interesting and rewarding. The discussions with the person responsible for the instruments at the test laboratory have also been very informative. He has many times been able to point me in the right direction and tell me who to speak with when I needed technical help from the IT department and other departments of Skoghall Mill. It has also been great to have the support and expertise from him when trying out different solutions when implementing the program.

5.5. Bugs and problems

A few bugs have been found in the testing the program. One bug that was found in the SCTView class was detected quite late in the development process. The bug presented itself when pressing the delete key, to remove a measurement value, when the cell for the value was selected in the datagrid. The delete key caused the whole cell to disappear rather than the measurement value. Figure 40 illustrates the datagrid before pressing the delete key with the bottom left cell selected.
Figure 40 - The datagrid before pressing the delete key.

Figure 41 illustrates the datagrid after pressing the delete key.

The solution to the problem was rather easy. The only change that had to be made was to set the datagrid built in property “CanUserAddRows” (which was set to true by default) to false. The following line of code shows the original XAML code for the datagrid:

```xml
<DataGrid ItemsSource="{Binding Valuelist}"

When the “CanUserAddRows” property was set to false the as seen on the following line of XAML code the problem was solved.

```xml
<DataGrid ItemsSource="{Binding Valuelist} CanUserDeleteRows="False"/>
```
One of the problems or at least a time consuming part of this project has been understanding and working with the graphical user interface in WPF. Much time has been spent figuring out how to customize datagrids, error labels and general design. Another time consuming task was to understand the theory behind the MVVM design pattern and how the communication between the different parts of the program takes place.
5.6. Training

To introduce the program to the operators of the laboratory introductory training sessions has been held. The features and workflow of the program were demonstrated for the operators and they also had the opportunity to ask questions about the program. A manual, and a quick guide, for the program have been created to guide the user of the program if any questions should arise. The manual is accessible from the help menu of the program. The manual and the quick guide can be found in appendix D.
6. Conclusion

6.1. Summary of the main conclusions.

The aim of this project has been to develop software that connects the SCT instrument at the test laboratory at Skoghall Mill to the mill’s database for process data and to automate the workflow at the test laboratory. This has been achieved with the introduction of the program called TestLink. The reason for the introduction of the program is to save time by eliminating manual work. The operators previously had to manually enter the test measurements into a program connected to the database for process data which was time consuming. Automating this process saves time. Manually entering the values also introduced the risk of, by mistake, entering the wrong value. One other goal with this project was to develop a platform where additional instruments can be added. With this platform deployed at the test laboratory, additional desired instruments can be connected to a common program with a consistent graphical user interface that is already familiar to the operators. The program for the SCT instrument has worked as a proof of concept for connecting instruments at the laboratory to the database for process data. The management and the operators of the test laboratory might be interested in further developing the program to include more of the instruments in the laboratory.

6.2. Reflections

One point of discussion during this project is the choice of the MVVM pattern when designing the program and WPF as tool for constructing the graphical user interface. Using windows forms in conjunction with the MVP design pattern [32], for example, might have reduced the time spent on creating the graphical user interface. This way more time could have been spent on the rest of the program and maybe a couple of more user stories could have been implemented. On the other hand the use of WPF, MVVM and particularly the use of databinding have provided flexible design which will be useful for
expanding the program to include new instruments. I personally think that it was a great opportunity to learn some new techniques and to broaden my knowledge of the Microsoft platform.

6.3. Future work

One goal with this project from the start was to make the program easily adaptable to different instruments. This has, as previously mentioned, been achieved by placing as much of the instrument specific implementation in an own module. The test laboratory at Skoghall Mill is happy with the result and might be interested in further development for more modules so that the program can communicate measurement values from additional instruments. One feature that was left out of this project was the ability for the program to display target values in the user interface. This feature was in the software specification but it was down prioritized. The implementation of target values would help the operators at the test laboratory further and is also a possible candidate for a future update.

6.4. Concluding remarks

The testing laboratory now has a working platform for connecting the test instruments to the database for process data. The implementation for connecting the SCT instrument as a proof of concept has turned out well. As a next step, it would be interesting to develop modules for additional test instruments like the ZD tensile tester (described in chapter two). It would also be particularly interesting to develop a module for an instrument that uses an Ethernet connection to communicate instead of a serial port like the SCT instrument does. These tests would confirm the generality of the test platform.
References


## Appendix A. List of user stories

<table>
<thead>
<tr>
<th>Nr</th>
<th>Som:</th>
<th>vill jag:</th>
<th>så att:</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Användare</td>
<td>kunna avbryta en mätserie.</td>
<td>felaktig mätserie inte sparas i systemet eller skickas till databasen.</td>
<td>klart</td>
</tr>
<tr>
<td>2</td>
<td>Användare</td>
<td>kunna redigera enskilda mätvärden undre/efter en mätseries gång.</td>
<td>felaktiga mätvärden kan åtgärdas.</td>
<td>pågör</td>
</tr>
<tr>
<td>3</td>
<td>Användare</td>
<td>kunna se mätvärden för alla position i mätseryen samtidigt.</td>
<td></td>
<td>ej påbörjat</td>
</tr>
<tr>
<td>4</td>
<td>Användare</td>
<td>kunna skriva in initialer.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Användare</td>
<td>att ID-nummer för aktuell mätserie visas.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Användare</td>
<td>kunna starta program för mätresultat från skrivbordet på den PC som tillhör varje mätinstrument</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>IT-ansvarig</td>
<td>att systemet kan köras på den hårdvara som finns tillgänglig i labbet (se specifikation av datorer i kartonglab)</td>
<td>systemet inte behöver en separat dator eller via sina krav på hårdvara påverkar nätverk, skrivare eller andra datorer i labbet</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Användare</td>
<td>kunna först mäta alla A-remsor och sedan alla B-remsor.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Användare</td>
<td>tydligt kunna se om en mätserie är avslutat och skickad.</td>
<td>en redan inskickad mätserie inte skickas igen</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Användare</td>
<td>tydligt kunna se om en mätserie pågår.</td>
<td>inte en ny mätserie påbörjas mitt i en tidigare mätserie.</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Användare</td>
<td>kunna skriva in tambour-id för hand.</td>
<td>inmatning av ID kan ske även om streckkod saknas.</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>IT-ansvarig</td>
<td>att formatet på initialer som matas in kontrolleras.</td>
<td>det går att identifiera aktuell användare vid varje mätserie/ID</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Användare</td>
<td>kunna se vilket instrument som är kopplat till programmet.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nr</td>
<td>Roll</td>
<td>Krav</td>
<td>Beskrivning</td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>------</td>
<td>------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Användare</td>
<td>kunna lämna systemet för att utföra andra arbetsuppgifter utan att systemet tappar mätvärden eller mätresultat</td>
<td>Fikapaus! Eller andra arbetsuppgifter som dyker upp, gör inte att man behöver starta om sitt jobb</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Användare</td>
<td>kunna skicka mätvärden från mätinstrument till databas</td>
<td>inga manuella inmatningar av värden behöver ske.</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Användare</td>
<td>kunna läsa in tambour-ID med hjälp av en sträckcodsläsare.</td>
<td>ingen inmatning av ID behöver ske för hand.</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>IT-ansvarig</td>
<td>att textfiler med mätresultat formateras på ett visst sätt och sparas på ett specifikt ställe.</td>
<td>Databasen kan plocka upp och tyda filerna.</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Användare</td>
<td>att det finns en hjälpstext eller manual kopplad till programmet för att underlätta om det uppstår frågor</td>
<td>varje användare (ny eller gammal), kan få en överblick av systemets funktioner och en kort beskrivning för vilka steg som ska genomföras</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>IT-ansvarig</td>
<td>kunna utveckla systemet för att integrera fler mätinstrument som eventuellt tas in i labbet</td>
<td>systemet återanvänds och ger användarna ett konsekvent och igenkännande vad gäller funktionalitet och grafiskt gränsnitt</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Användare/IT-ansvarig</td>
<td>att systemet har en knapp för direktsupport vid problem</td>
<td>jag kan via en knappstryckning få hjälp om systemet stannat</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Användare</td>
<td>att det ska vara så få moment som möjligt under en mätserie.</td>
<td>så lite tid som möjligt går till knappstryckningar och liknande.</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Användare</td>
<td>att programmet ska vara så stabilt som möjligt</td>
<td>inga mätvärden försvinner på grund av att programmet kraschar.</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Användare/IT-ansvarig</td>
<td>att en warning visas om ett ogiltigt ID-nummer matats in.</td>
<td>mätserier med ogiltigt ID inte skickas till databasen.</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Användare</td>
<td>att systemets GUI är anpassat till de skärmstorlekar som finns vid varje stations</td>
<td>jag slipper scrolla eller klicka runt för att få en överblick av mätserier, idn eller knappar</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Användare</td>
<td>ha ett så tydligt GUI som möjligt</td>
<td>värden inte blandas ihop.</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Användare</td>
<td>kunna skriva ut mätvärden till en skrivare.</td>
<td>mätvärden lätt kan visas upp på papper.</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>IT-ansvarig</td>
<td>att funktionalitet för att skapa textfiler lätt kan anpassas.</td>
<td>programmet lätt kan anpassas om databasen uppdateras.</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>IT-ansvarig</td>
<td>att det finns dokumentation som beskriver hur systemet integrerar mätinstrument och databasen</td>
<td>vi enkla koppla upp fler mätstationer eller flytta labutrustning</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>IT-ansvarig</td>
<td>kunna anpassa systemets lagring av mätserier</td>
<td>systemet kan integrera andra verksamhets eller affärssystem om detta blir nödvändigt</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Användare</td>
<td>att systemet inte blockerar ett mätinstrument</td>
<td>om systemet krånglar eller krashar så kan mitt arbete fortsätta</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Användare</td>
<td>att ogiltiga mätvärden så som nollvärden och negativa värden markeras med en blå överstrucken text i det grafiska gränssnittet</td>
<td>de ogiltiga mätvärdena skiljer ut sig från de giltiga värdena och lätt kan identifieras.</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Användare</td>
<td>kunna se om mätvärden avviker från specifikation för mycket med hjälp av färgkoder i GUI.</td>
<td>felaktiga mätvärden lätt kan identifieras.</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Användare</td>
<td>ha en kort historik över tidigare mätserier.</td>
<td>en mätserie kan skickas igen om databasen ligger nere.</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Användare</td>
<td>att antalet positioner som ska provas i en mätserie styrs automatiskt.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>Användare</td>
<td>i vissa fall (för vissa instrument) kunna skicka mätvärden innan en mätserie är klar.</td>
<td>kartongmaskinens operatörer kan få resultat så tidigt som möjligt.</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Användare</td>
<td>inte bli beroende av mus vid inmatning eller kontroll av systemet.</td>
<td>programmet går att använda där bara tangentbord får plats eller finns</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>Användare</td>
<td>kunna köra systemet på en pekplatta model X</td>
<td>systemet kan användas även vid de stationer som inte har plats eller möjlighet för en PC</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Användare</td>
<td>kunna övervaka alla mätstationers aktuella och pågående mätserier från en annan dator, pekplatta eller mobil device på samma nätverk</td>
<td>jag kan hjälpa mina kollegor i labbet med råd från mitt kontor eller fikarum eller övervaka våra praktikanter när de kör mätserier</td>
<td></td>
</tr>
</tbody>
</table>
Appendix B. TextFileCreator.cs

public class TextFileCreator
{
    private const string FILE_FORMAT = "txt";
    private const string ID1 = "0";
    private const string ID2 = "0";
    private const string ID3 = "0";
    private const string UNUSED = "0";
    private const string ONE_STRING = "ONE";
    private const string TWO_STRING = "TWO";
    private const string COLUMN_SEPARATOR = ",";

    public TextFileCreator(BoardMachine boardMachine, string tambourNumber, InstrumentType
        nstrumentType, NumberOfPositions nrOfPositions, TestType testType, List<double> AList,
        List<double> BList, string signature, string ACode, string BCode)
    {
        string line1 = "1,"; //type one - File Header
        string line2 = "3,"; //type three - Sample area
        string line3 = "9,"; //type nine - User info
        string line4 = "11,"; //type eleven - Sample ID
        string line5 = "12,"; //type twelve - Sample result
        string line6 = "12,"; //type twelve - Sample result
        string fileName = "";
        string dateAndTime = "";

        if (AList.Count == ((int)nrOfPositions) && BList.Count == ((int)nrOfPositions))
        {
            //Create text strings from values list
            string AString = "";
            string BString = "";
            for (int i = 0; i < (int)nrOfPositions; i++)
            {
                AString += Convert.ToString(AList[i], CultureInfo.InvariantCulture) + 
                COLUMN_SEPARATOR; //replace , with .
                BString += Convert.ToString(BList[i], CultureInfo.InvariantCulture) + 
                COLUMN_SEPARATOR;
            }
            AString = AString.TrimEnd(’,’); //remove the last comma
            BString = BString.TrimEnd(’,’);

            //create file name
            fileName = createFileName(instrumentType, tambourNumber, boardMachine);

            //create date and time
            dateAndTime = DateTime.Now.ToString();

            //create line1
            line1 += fileName + COLUMN_SEPARATOR + FILE_FORMAT + COLUMN_SEPARATOR + dateAndTime;

            //create line 2
            line2 += (int)boardMachine + COLUMN_SEPARATOR + tambourNumber + COLUMN_SEPARATOR + 
                ID1 + COLUMN_SEPARATOR + ID2 + COLUMN_SEPARATOR + ID3;

            //create line 3
            line3 += signature;

            //create line 4
            line4 += (int)testType + COLUMN_SEPARATOR + UNUSED;

            line5 += createPositionString(nrOfPositions, boardMachine, ACode) + COLUMN 
                _SEPARATOR + UNUSED + COLUMN_SEPARATOR + (int)nrOfPositions + COLUMN_SEPARATOR + 
                AString;
        }
    }
}
line6 += createPositionString(nrOfPositions, boardMachine, BCode) + COLUMN_SEPARATOR + UNUSED + COLUMN_SEPARATOR + (int)nrOfPositions + COLUMN_SEPARATOR + BString;

string path = fileName + "." + FILE_FORMAT;

File.Create(path).Close();
TextWriter textWriter = new StreamWriter(path);
textWriter.WriteLine(line1);
textWriter.WriteLine(line2);
textWriter.WriteLine(line3);
textWriter.WriteLine(line4);
textWriter.WriteLine(line5);
textWriter.WriteLine(line6);
textWriter.Close();

private string createFileName(InstrumentType instrumentType, string tambourNumber, BoardMachine boardMachine)
{
    string fileName = "";
    switch (instrumentType)
    {
        case InstrumentType.SCT:
            fileName = "sc";
            break;
        case InstrumentType.ZStrength:
            fileName = "zs";
            break;
        default:
            break;
    }

    //add tambour number
    fileName += "." + (int)boardMachine + tambourNumber;
    return fileName;
}

private string createPositionString(NumberOfPositions nrOfPos, BoardMachine bm, string instrumentCode)
{
    string positionString = "";
    //add board machine
    positionString += (int)bm;

    switch (nrOfPos)
    {
        case NumberOfPositions.ONE:
            positionString += ONE_STRING;
            break;
        case NumberOfPositions.TWO:
            positionString += TWO_STRING;
            break;
        default:
            break;
    }

    positionString += instrumentCode;

    return positionString;
}
Appendix C. Additional classes

PortSimulator
The PortSimulator class simulates the serial port of the SCT instrument. It has been used when working with the program outside of the laboratory. The class can generate a measurement value within a given interval and a corresponding checksum with the same algorithm that the SCT-instrument uses. The simulation is done by creating data packets, containing the generated measurement value and a checksum, identical to the packets sent by the instrument (see 4.8.1.2). The values are then sent to the program in a similar way that the SCT instrument does.

TestLinkEnums
The class contains enums [33] used throughout the program. The enums are, for example, used to identify number of positions to be tested and type of instrument connected to the program.

DoubleItem
Each measurement value produced by the SCT-instrument is stored as an object created from the DoubleItem class. The class also holds information about how the value should be presented in the graphical user interface. This is controlled by storing different font formats for each value. This way the elements in the graphical user interface can bind its font to the property in a DoubleItem object. If a value is invalid (negative or zero value) the font format is set to have a bold blue crossed over format. The conditions to be met to be an invalid value, as well as how an invalid value should be presented are easily customizable.
Appendix D. User guide

Användarmanual TestLink

1. Inställningar för instrument

För att SCT-apparaten ska kunna skicka mätvärden till programmet krävs att ”report” på instrumentets knappsats är inställt på något av de två undre alternativen.

2. Öppna programmet

Om programmet inte redan är startat, dubbelklicka på ikonen på skrivbordet enligt figur 1

![TestLink ikon](testlink.png)

Figur 1 - TestLink i ikon.
3. Inmatning av data för provet

Börja med att klicka på knappen ”Nytt prov” och fyll i textfälten för tambournummer samt signatur i enligt figur (se figur 2).

![Figur 2 - Inmatning av tambournummer och signatur.](image)

Tambournummer kan antingen matas in med streckkodsläsare eller manuellt med tangentbordet. Vid manuell inmatning ska sex siffror skrivas in. Den första siffran representerar kartongmaskin och de fem efterföljande siffrorna representerar tambournummer (se följande exempel).

**Ex 1**

- Tambour 45890 från KM7 ska matas in som 745890
- Tambour 32989 från KM8 ska matas in som 832989

Nästa steg är att mata in signatur (fyra tecken). Programmet har en funktion för att kontrollera att tambournummer och signatur har korrekt format. Om formatet inte är korrekt visas en hjälptext med information om vad som är fel med inmatningen (se figur 3).

![Figur 3 - Felaktig inmatning av tambournummer och signatur.](image)
När tambournummer och signatur är korrekt inmatade aktiveras kryssrutorna för att välja om provet är ett extraprov samt antal positioner och knappen för att starta ett nytt prov aktiveras (se figur 4).

![Korrekta inmatade signatur och tambournummer](image.png)

Figur 4 - Korrekt inmatat signatur och tambournummer – kryssrutor val för antal positioner, extraprov och knappen för ”Nytt prov” aktiveras.

Om provet är ett extraprov ska det markeras med kryssrutan för extraprov (se figur 4). Om inmatning av tambournummer sker med streckkodsläsare markeras extraprov automatiskt om så är fallet. Antalet positioner som kan väljas beror på vilken kartongmaskin som valts. KM8 ger 1 samt 3 och KM7 ger alternativen 1 samt 2. Se alltid till att kontrollera att tambournumret är korrekt inmatat eller inskannat.

4. Starta en mätning

För att börja provningen, klicka på knappen ”Starta provning” (se bild 4). När knappen har klickats inaktiveras knappen ”Starta provning” samt alla inmatningsfält och kryssrutor. Inaktiveringen sker för att inte data om provet ska kunna ändras av misstag under provningens gång (se bild 5).
Figur 5 - Provningen har startats genom att användaren har klickat på ”Starta provning”. Notera att provet inte går att skicka i det här läget.

Nu är programmet redo att ta emot värden från SCT-instrumentet. Prova först alla A-remesor, sedan alla B-remesor. Om ett omrimsligt värde (nollvärde eller negativt värde) överförs från SCT-instrumentet markeras värdet med en blå färg och stryks över (se figur 6).

Figur 6 - Orimliga värden stryks över och exkluderas från medelvärdet.

När en mätserie är klar och alla positioner fyllts med mätvärden aktiveras knappen skicka (se figur 7).

För att skapa ett nytt prov efter att ett prov är färdigt, eller ett prov har avbrutits, används knappen ”Nytt prov”. När nytt prov klickas tas det aktuella provet bort och ett nytt tomt prov skapas och programmet är redo för att starta en ny mätning (se avsnitt 4).
5. Snabbguide

1. Se först till att SCT-instrumentet är rätt inställt (något av de två understa alternativen för ”report” på SCT-instrumentets knappsats ska lysa ).
2. Klicka på ”Nytt prov”. 
5. Välj antal positioner och om provet är ett extraprov eller ej (om tambournumret matats in med streckkod sköts markering av extraprov automatiskt).
6. Klicka på ”Starta provning”.
8. Om något enskilt värde behöver ändras, mät först klart alla positioner. Markera sedan rutnan för det värde som ska ändras och skriv in det nya värden (mät alltså ett nytt delprov och mata in det manuellt i TestLink).
9. När alla positioner är mätta, klicka på ”Skicka” och på ”Skriv ut”.
10. För att mäta nästa prov, klicka på ”Nytt prov”.

Figur 8 - Figur till snabbguide.