Expertise
and insight
for the future

Deniss Komarskis

Efore’s Mini Test System

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor’s Thesis

10 May 2021

Author Title Number of Pages Date	Deniss Komarskis Efore's Mini Test System 33 pages 10 May 2021
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Smart Systems
Instructors	Sami Sainio, Senior Lecturer Pasi Lauronen, Chief SW Architect, Efore Telecom Finland Oy

This thesis presents the research and realization of a program that controls connected test equipment, calculates the device's efficiency under test for different loads, and makes a report for the tester. There was a need for such a program for the testing team in the development division of Efore Telecom Finland Oy as it could save human resources and money for the company.

The main problem which should be solved is to automate actions that usually are done manually. One of these actions is making measurements for the device under test. This type of work consumes plenty of time because there is a need for many measurements.

The task was to create a program that can communicate with the necessary devices and control them with a bit of human influence. In addition to that, it should be easy to use for the tester. For the first version, it was enough to do essential functions. For example, they are measuring and set settings for testing equipment.

The need for this program reveals how time became more valuable for the implementation of different projects. In addition to that, automated actions can help to avoid errors, which can be done by a human when work is performed manually. If this program would demonstrate excellent results, then future development is guaranteed.

Keywords	Power efficiency, programmable instruments, SCPI, Python 3, test sequence, architecture

Contents

List of Abbreviations

1 Introduction 1

2 Theoretical background 3

2.1 Power efficiency analyzing 3
2.2 Communication interfaces 6
2.3 Programmable instruments 7
2.3.1 Chroma AC Source 61505 7
2.3.2 Chroma DC Electronic Load 63200 series 8
2.3.3 Zes Zimmer Power Analyzer LMG600 8
2.4 SCPI 8
2.5 Python 3 programming language 9

3 Employer's requirements 11

3.1 Instruments 11
3.2 Main features for software 12

4 Program implementation 14

4.1 Program architecture 14
4.2 Launching EMTS 16
4.3 DeviceList.info and how EMTS find instruments 18
4.4 Prepare parameters file 19
4.5 Main Test Sequence (MTS) 23
4.6 User interface 27

5 Results and discussion 29

5.1 Employer's feedback 29
5.2 Future improvements 29

6 Conclusion 31

References 32

List of Abbreviations

AC	Alternating current
API	Application programming interface
CLI	Command-line interface
DC	Direct current
DUT	Device under test.
EMTS	Efore's Mini Test System
GPIB	General Purpose Interface Bus
GUI	Graphical user interface
LAN	Local area network
MTS	Main Test Sequence
NI-VISA	National Instruments VISA
PC	Personal computer
R&D	Research and Development
RMS	Root mean square
SCPI	Standard Commands for Programmable Instruments
VISA	Virtual Instrument Software Architecture

1 Introduction

Testing is crucial for electronic devices because the modern world depends on them, and every failure could be critical as for smartphones as for large power stations [1]. However, regarding the damage that can be dealt with failure, the coverage of tests can vary from only to turn on and off the device to a complete set of long-term and extreme environmental tests. For example, a test with a control group performs the following way when all devices are not tested but only part of them selected by quality assessment personal. In contrast to this, there are cases when produced device must pass all prepared in advance tests to be allowed to ship to the customer.

This thesis is about optimizing at least some of the tests needed during Efore's product development stage. Efore is an international company that develops and produces highly efficient, reliable, and performant power solutions. Offered solutions vary from small rectifiers to large power cabinets.

There are many tests performed during development for any product in Efore's R&D team. In addition to that, clients of Efore usually have their requirements for quality and performance for the product. For example, the most common request from clients is that devices must be environmentally friendly, and one of the attributes which pertain to it is power efficiency.

This thesis aims to create a PC program that could decrease the required amount of human interaction and save time for repeatable tests performed during development for Efore's products. Specifically, the program should be used to do the following: automatically connect to the programmable instruments, send commands to them, and store received data.

During development, the program got the codename EMTS, and this name will be used in this thesis. It should be mentioned that a decision was made to use Python programmable language for this project because it is well-implemented to work with text lines [2]. There are five more chapters in this thesis for theory, more details about EMTS and results.

The second chapter will describe the theory for basics of power efficiency analyzing, Python programable language, and its packages used in this work. In addition to that, there will be a short introduction about programmable instruments supported by EMTS today.

Since this thesis was done for a company, the third chapter is dedicated to describing the requirements for EMTS. The requirements list was created by the R&D team of Efore. It should be noted that during development, many features were appended to the list of wanted features. However, this thesis presents only those features that have been implemented.

The fourth chapter will present and discuss program implementation itself. It is the main part of this thesis, and it describes solutions that were developed to fulfil the employer's requirements. In addition, some of the most challenging problems and their solutions are also discussed in this chapter.

The last two chapters are the results and conclusion. The work results are based on feedback from the R&D team. In addition to that, there are present future improvements that can be applied to this EMTS project. Finally, the conclusion will include final thoughts about the results and the summary itself.

2 Theoretical background

2.1 Power efficiency analyzing

One of the main factors for each power device as power supply, ac-dc or dc-dc convertors is power efficiency. Usually, the customer provides requirements and specific features that the device should perform. Possible features could be making measurements by device, remote control of the device, communicating with other devices, etcetera. However, this part of the thesis will explain the most essential and commonly required requirement. It is the power efficiency of the device.

Power is work that is done during some amount of time. [3, p 166] However, this chapter is about power for electrical systems. The power for electrical systems is the rate of the flow of the charge per second with the required force to move electrons. It is calculated as in Equation 1. In this equation, the force which is required to move electrons is called voltage. The rate of the flow of charge per second is current. [4]

Equation 1 Power in electrical systems

$$P = V \cdot I$$

The following example will demonstrate regular tasks for power efficiency analysis. For example, a customer asked for a power supply with one input and one output and with efficiency requirements presented in Table 1. In this table are present "Output load range" and "Efficiency" columns. The "Output load range" column illustrates the power load range, and in Equation 2, definite output load presented as $\sum P_{out}$. The "Efficiency" column represents device efficiency according to its output load.

Table 1 Imagined customer requirements

Output load range (W)	Efficiency
0 - 20	No requirement
>20 - 80	>60%
>80 - 160	>60% at 80W, linear increase to 80% at 160W
>160 - 320	>80%

Efficiency is calculated using Equation 2. In this formula, Total Efficiency is calculated by dividing total output by total input power. The received number will be between 0 and 1.

Then it should be multiplied by 100. It will be a percentage number that represents Total efficiency.

Equation 2 Total efficiency

$$Total\ Efficiency = \frac{\sum P_{out}}{\sum P_{in}} \cdot 100\%$$

It is a calculated summary of power outputs because it can be more than one for the DUT. However, in this case, the sum symbol is not needed because this device has only one output, as it was told above. The same applies to input power. The formula which is required to calculate power is dependent on the electric current type.

Equation 3 presents how to calculate power for DC and Equation 4 for AC. In these formulas, *V* stands for voltage and *I* for current values. PF is a power factor, and it is a ratio that has a range from 0 to 1 and is needed only for AC power measurement. [4]

Equation 3 DC power

$$P_{dc\ power} = V \cdot I$$

Equation 4 AC power

$$P_{ac\ power} = V \cdot I \cdot PF$$

Table 1 could be presented as a graph illustrated in Figure 1. The horizontal axis represents Output power from the device. The vertical axis is total efficiency which is calculated by the formula from Equation 2. In this table, it is easier to see the bounds of requirement. The device must work with efficiency above or equal "Efficiency line" to fulfil customer needs.

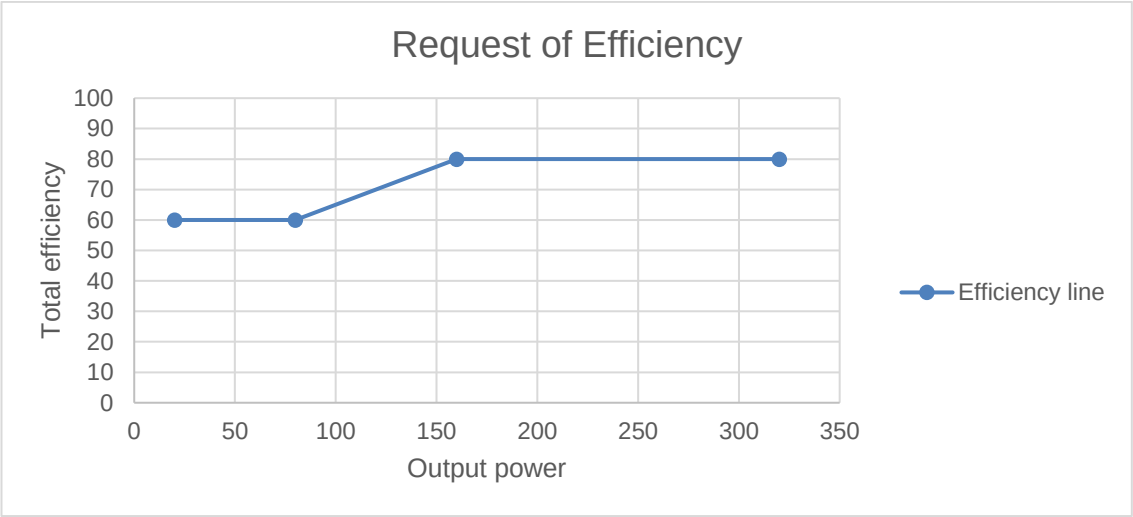

Figure 1 Graph for imaginable customer requirements

For the current example, the efficiency must be appropriate in the range from 20 to 320 watts. Everything that is beyond this range is not essential. In Figure 2, all efficiency measurements are above the requirement line. That means that efficiency for a device will fulfil customer's power efficiency needs.

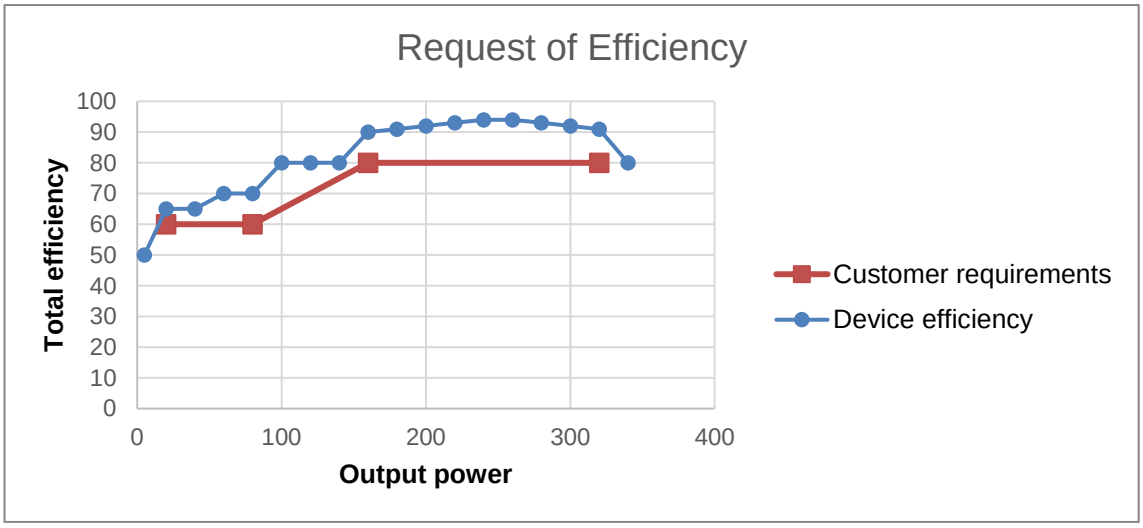

Figure 2 Example of successful fulfilment of customer requirements

Figure 3 is presented here as being opposed to Figure 2. Device efficiency at some point drops below customer requirements. Therefore the device is inappropriate for a customer, and it must be sent to rework.

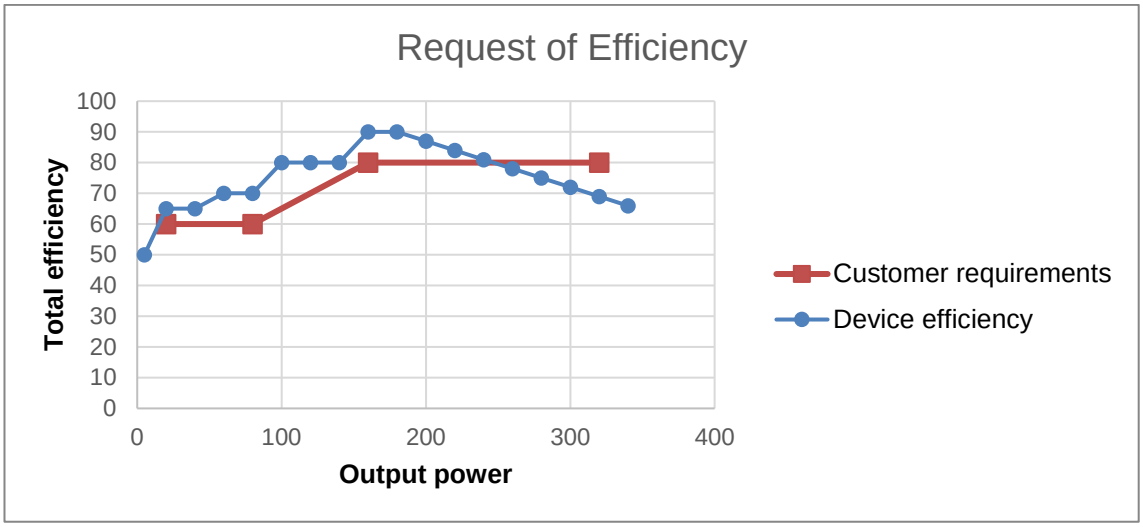

Figure 3 Example of failed fulfilment of customer requirements

Independently from the result the Figures 2 and 3 have valuable data for the customer. It has a reasonable amount of measurement points, information is presented conveniently, and every piece of information is captioned. A helpful improvement could be increasing the number of measurement points without spending many human resources, and it was a prime purpose of creating EMTS.

2.2 Communication interfaces

Manufacturer of test equipment produces their instruments with different communication interfaces. They allow a tester to connect test equipment between other tools or connect it to a computer to simplify interaction. The most common interfaces are RS-232 and GPIB. However, for this thesis purpose was used RS-232 and less common USB interfaces. Both are ideally suited for EMTS because it is easier to interact with them. In addition to that, they are supported by NI-VISA. [5]

NI-VISA is the particular driver, which is based on VISA specification. [6] With NI-VISA, it is not needed to implement each communication interface for each device. It is needed only to make a solution that sends SCPI commands to the device through the NI-VISA driver.

2.3 Programmable instruments

Many tools are in use for the development process, and programmable instruments are one of them. Programmable instruments are devices that can be controlled by using SCPI language. [7] Usually, all these instruments are controlled with the user's PC. These instruments have a communication interface to allow to do this. It depends on each manufacturer which communication interfaces the instrument will have. The most common interfaces are RS-232, RS-485 and GPIB. Manufacturers implement less USB and LAN interfaces.

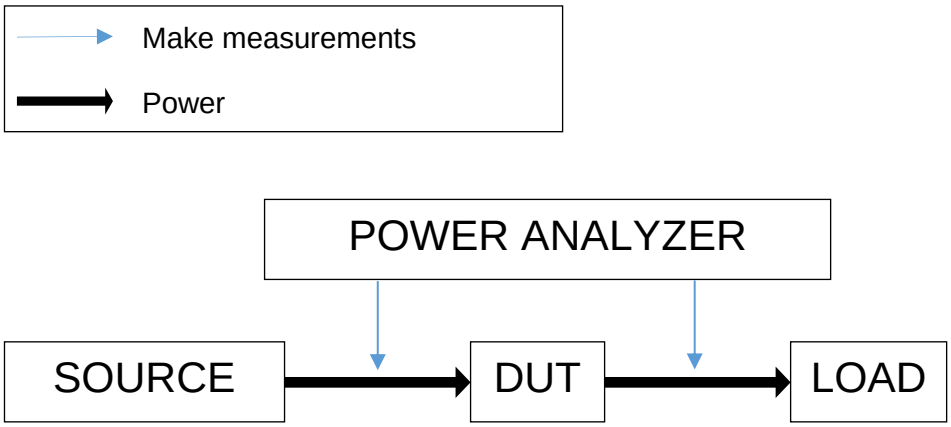

Figure 4 Concept of devices relationship

Programmable instruments can be sources, loads, or power analyzers. Figure 4 demonstrates the most common connection between them and DUT for power efficiency measurement. First, the DUT must be powered by the power source. It can be an AC source or DC source corresponding to the DUT input specification. Second, the DUT must have output connector(s) to which load(s) could be applied. The DUT output also can be AC or DC, depending on specifications. At last, there is a power analyzer for accurate measuring of input and output for DUT.

2.3.1 Chroma AC Source 61505

The Chroma AC source 61505 is a highly efficient AC power source that provides accurate measurement of power and sine wave output with low distortion. The instrument's digital signal microprocessor generates stable and accurate output voltage and frequency. The pulse width modulation design of the power stage allows for full volt-ampere into loads. The front panel has both a rotary pulse generator and keypad controls

for setting the output voltage and frequency. In addition to that, it is also possible to enable DC mode for this instrument. The display provides a complete operating state of the unit to the user. Remote programming and controlling are accomplished either through the GPIB bus or the RS-232C serial port. [8]

2.3.2 Chroma DC Electronic Load 63200 series

The Chroma DC Electronic Loads 63200 series are high power loads. This series's difference between instruments is variations on voltage, load current, and power ratings. However, features, functions, and remote options are the same for all loads of the 63200 series. [9]

This series suits EMTS because there are GPIB, RS-232 interfaces. In addition to that, almost all instruments operations can be done remotely by communication interfaces. Such operations are constant current mode, constant resistance mode, constant voltage mode, and constant power mode. [9]

2.3.3 Zes Zimmer Power Analyzer LMG600

The Zes Zimmer Power Analyzers are used widely in the electric and electronics industry. These instruments are innovative with useful modern features such as a power analyzer with independent sync on multiple channels, a power analyzer with a logical group concept, etcetera. Test labs use them to guarantee compliance with standards, and universities rely on them to train future generations of engineers and scientists. [10]

This instrument's key feature for EMTS is remote control via the RS-232 interface. In addition to that, there are vast amounts of commands that could be sent to the instrument, which provides almost complete control and receives various precise measurements [10]. However, EMTS does not use all possibilities of this instrument yet.

2.4 SCPI

SCPI standard was introduced in 1990 for IEEE-488.2 specification [11]. This standard was needed to create rules for manufactures on how end-user should communicate with

the device over GPIB. Later these rules spread over all interfaces as USB, RS-232 and similar.

SCPI is a standard for IEEE 488.2 programmable instruments commands. The manufacturer-specific instruction sets are based on this standard. For programmable instruments are also used IEEE 488.1 standard, which is responsible for control mode, command, and data messages transmission. The SCPI contains a command hierarchy, response formats, and a programming instruction set. Controller and the bus between the measuring device may also be other than IEEE 488, such as a serial, instrument, or field bus. SCPI commands are plain language, and they consist of ASCII characters. [11]

The most significant benefits of SCPI are simplicity and standardization. That means that it is easy to implement into programs and excellent interchangeability for instruments that are in use. For example, it is not needed to make changes in the program when a user needs to change measuring instruments.

Each manufactures, for particular series of instruments, has its specific variation of SCPI commands. Therefore, it is necessary to read documentation to learn about particular details and custom commands if such present. Usually, instruments from the same manufacturer have similar SCPI commands for one type of instrument. However, there are reserved commands by IEEE 488.1, which must be implemented identically for all instruments. Such commands start with the asterisk sign (*). There are examples of the most common commands in table 2.

Table 2 Asterisks SPCP commands for programmable instruments

SCPI command	Meaning
*IDN?	This command typically writes with a question mark because it is a query request to identify itself for an instrument.
*OPC?	Waits for all operations to be completed and causes the device to return bit on complete
*RST?	Reset device command

2.5 Python 3 programming language

Python was introduced by a Dutch programmer Guido von Rossum in 1990 [12]. Today Python developed into a powerful instrument and started spread over programmers and enthusiasts. This high-level programming language is extensively used in modern

applications. It can be used in web development, data science, desktop applications, and education [13].

Python's key features as high-level data structures, object-oriented programming, and elegant syntax, make it the ideal language for application development. Python is an interpreter that makes development faster because it is not needed to compile a source code. Moreover, Python has many open-source third-party packages, toolkits, and modules that solve some tasks. Table 2 illustrates the list of assets that were used for EMTS

Table 3 Used assets for EMTS

Module/library/toolkit name	Short description
NumPy	It is the library that provides tools to work with massive arrays and matrices.
wxPython	It is a Cross-platform GUI toolkit for Python programmers who needs to create applications with highly functional GUI. Moreover, it is free, and everyone can modify it because it is open source. [14]
PyVISA	It is the Python package that allows to control of any measurement devices and supports various communication interfaces.

Version 3.7.4 of Python was used for this thesis. It was the last version of Python at that moment.

It should be noted that Python has native support for the RS-232 interface. It is built in a package "serial". This package was used at the beginning of the project. However, during development, it was decided to move to another more advanced library, PyVISA.

3 Employer's requirements

The employer's request was to create software for the testing team in R&D. The main purpose of the software was to save working hours by automating interaction with devices and storing records to the file, and creating a graph if needed. Furthermore, EMTS should automate configuring and controlling of instruments like AC/DC-sources, power analyzer, and DC-loads. In addition, EMTS should include a graphical user interface and report generator. In other words, the target is to develop a system that saves working hours compared to manual PQ (Product Qualification) measurements.

3.1 Instruments

Table 4 List of the instruments which EMTS must support

Instrument name	Instrument Type
Chroma AC Source 61505	AC/DC source
Chroma DC Source 62000P series	DC source
Chroma DC Electronic Load 63600 series	Load
Chroma DC Electronic Load 63200 series	Load
Zes Zimmer Power Analyzer LMG600	Power Analyzer

Table 4 presents a list of the instruments that should be supported for the program's first version. Support for instruments means that it is possible to control the instrument by program and make measurements. It is recommended to implement as many measurements as possible which instrument is capable of doing.

Comparing instruments from table 3 was learned that it is possible to connect to them with a USB or COM port and communicate with them using NI-VISA. However, these instruments have slightly different controlling logic between each other. For example, to control Loads of 63200 series, it is needed to send a command to them which changes the instrument's mode to remote mode when all the rest instruments change their mode automatically when the first command is sent. That means that there was needed to learn how to standardize the program's logic with the instrument's control logic.

3.2 Main features for software

Employer prepared a list of which features had to be realized in the software. The list was divided into different priority features. Most of the features were implemented, but some of them are still in progress. Table 5 presents the finished and released features for EMTS.

Table 5 List of finished features for EMTS

Feature	Explanation
Set user-defined settings for test equipment	Allow users to set additional settings using EMTS which are not set by default.
Make measurements	EMTS makes measurements using a programmable measure device or programmable test equipment.
Record measurement results	Measurements results are stored in a .csv file so that they can be viewed and plotted in Excel
Pre-test	Check if minimum and maximum Load test values are suitable for the DUT and test equipment device can handle Source test values.
Specific measurements by user	Allow user to set which measurements should be taken by the current test. Also, allow a user to create a custom measurement sequence.
Store settings and configuration	Settings and configuration of the measurement setup are stored in a file so that test can be reloaded.
Support multiple loads(output) and sources(input)	It is implemented. However, It was not tested with more than five connected devices.
A user-defined output file name with timestamp	There is the ability to set the user-defined file name and timestamp for output data files.
A version of the program is always visible	It was an essential requirement from the employer
The EMTS supports to pause the test	In some cases, it is needed to pause the running test. All current commands would be finished before pausing the test. Then the user must press the button to continue the test.
The EMTS supports stopping the test.	All current commands would be finished as in the pause feature, but after that, EMTS will initiate a stop sequence. It is needed 1-2 seconds to stop output from instruments.

Pass-Fail	EMTS supports this. However, It is a needed GUI for convenient use of this feature.
UI	Offer a simple User interface as a console window
Calculate total energy efficiency by default	EMTS calculates total efficiency as in Equation 2 where P_{out} means load power and P_{in} means source power

Despite all requested features, the EMTS should always be remained to be allowed for improvement. What made an idea to create a simple program structure that can fetch and store data first. After that, the EMTS started to become overgrown with features.

Low-priority tasks were not implemented yet. Low priority tasks are tasks for advanced GUI or drawing efficiency graphs by EMTS. These features are low-priority because simple CLI is sufficient, and the efficiency graph can be drawn in Microsoft Excel.

4 Program implementation

Program implementation will be described in the last stable version of EMTS in this chapter. It is 0.4.3, which was delivered to the testing team on 6th February 2020. In this version, almost all the latest requirements are implemented and tested.

4.1 Program architecture

At the beginning of the Thesis, EMTS was started developing as a simple script with limited functionality. Further development added more possibilities for the user to control the process of the program's runtime. At the same time, it was forming the program's architecture.

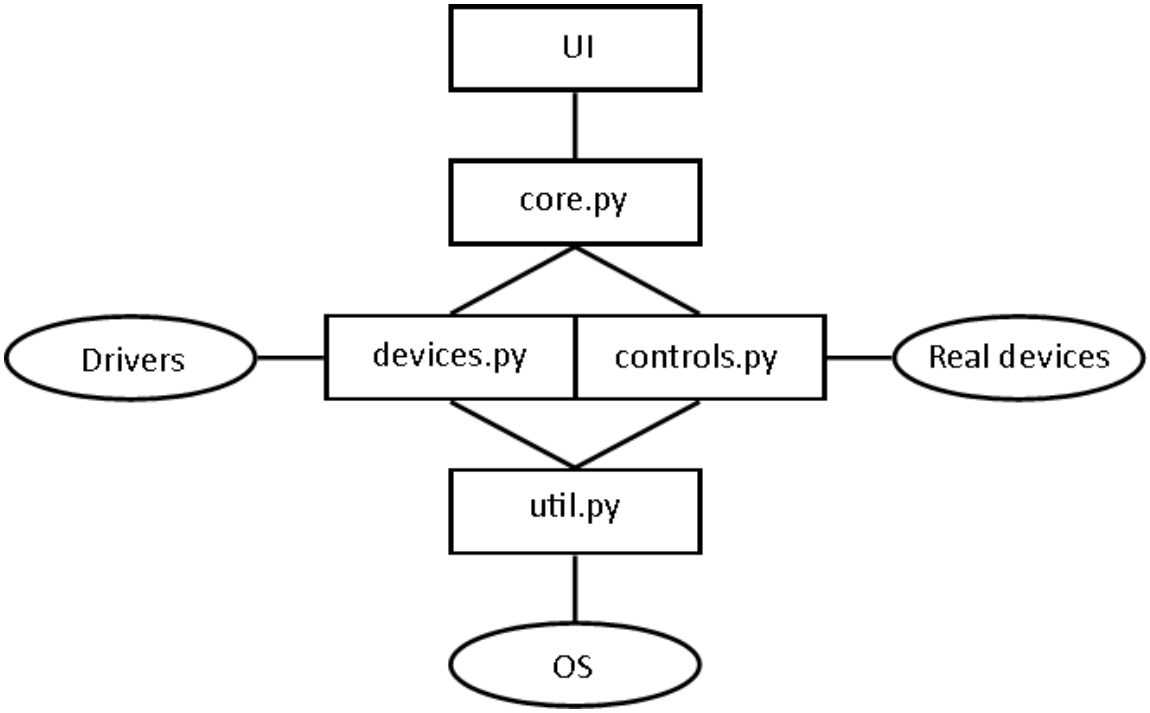

Figure 5 Program's logic of classes

The main program is based on four modules: core, devices, controls, and util. Each module has its responsibilities, and they will be described below. Figure 5 displays the current program's architecture. The current architecture is based on the principle that UI can initialize data and control states of the program's core. However, it can not interfere directly with workflow. It allows to cover most of the errors in the beginning and inform the user about them.

As mentioned before, the UI has the simple role of controller. In this case, UI is responsible for the user's input. It can be input files or the user's input itself during runtime. Moreover, UI can control states of the program's core, but these states are defined in the core and can not break the core's runtime. This approach allows changing user interfaces without changes to the core of the program. For example, the current version of UI is a console user interface, and in the future, there is a plan to switch to a more friendly graphical user interface.

The core is the part of the program responsible for everything that the program allows a user to do. It can run different actions such as running the main test sequence, finding and controlling programmable instruments, preparing output for a user, and checking the correctness of performed actions. However, this module cannot work as a standalone and needed a high-level controller as a user interface.

The controls module is responsible for communicating with real devices. In this case, real devices are programmable instruments. The current version of the program uses the PyVISA library to communicate with devices. In addition to that, this module is responsible for finding connected devices to the computer. However, it supports only connections over USB and COM ports.

The Devices module creates device objects that the program can use, and it works with controls to work with the real world. This module uses drivers to translate the program's tasks to SCPI commands for programmable instruments. In general, this module is responsible for having information about connected instruments and communicating with them.

Util module is a general part of the program, and right now, it is used by every part of the program. Mostly it has methods that operate with OS. However, it also has various classes and functions for repeatable code and actions. In the future, this module should be refactored.

4.2 Launching EMTS

This subchapter will explain program runtime from the user's point of view. The current workflow is presented in Figure 6. Despite that this workflow was created for the console version, most of the main options are also reuseable for other user interfaces.

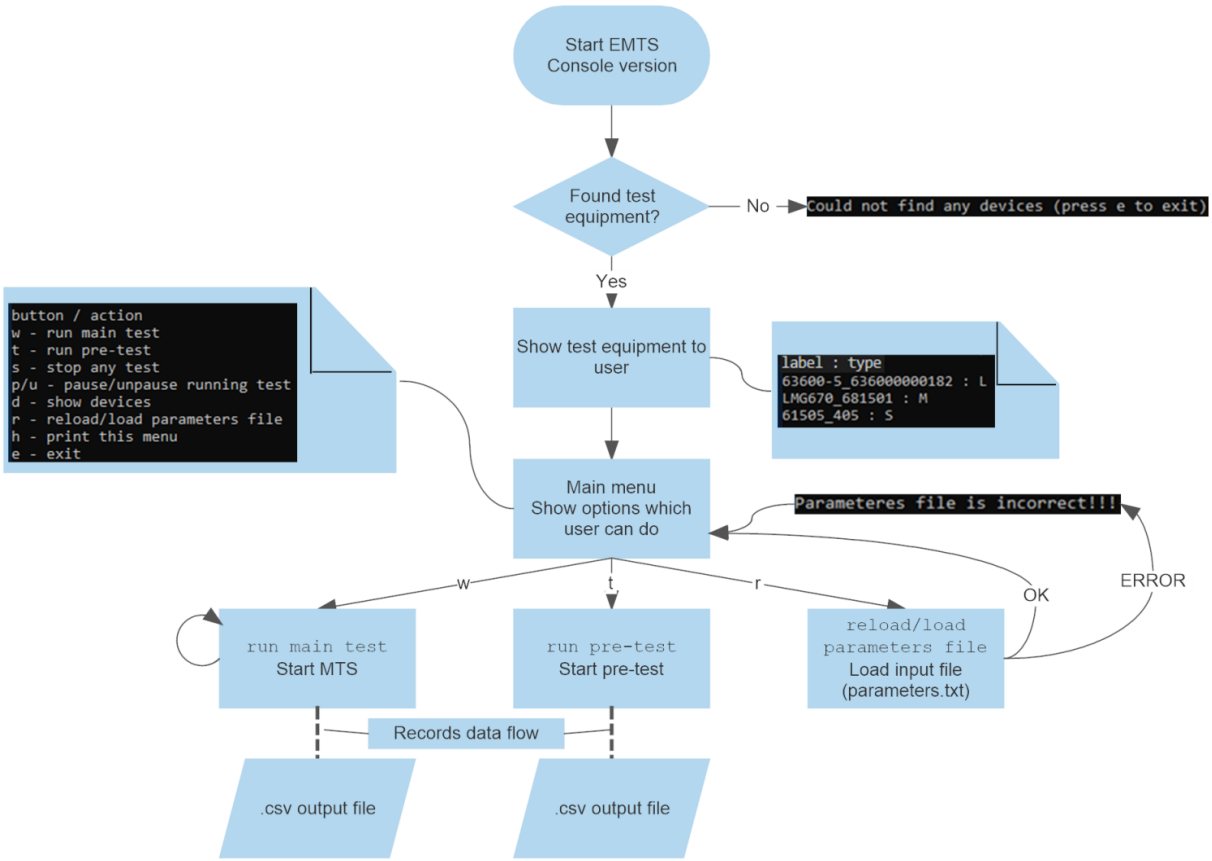

Figure 6 Console workflow of EMTS

The program tries to find test equipment on start. It uses the NI-VISA library and a file DeviceList.info, which will be described below. If EMTS did not find any instruments connected, then there will be the message on the screen "Could not find any devices (press e to exit)". In this case, a user should check the connection between instruments and computer with EMTS or check if the program supports used instruments.

When instruments were found, these instruments will be displayed on a user interface in a table with two columns: label and type, as in Figure 7. The label column stands for the instrument's name and the type column for the instrument type in this table. The instrument's label is a combined name from model name and serial number separated

with an underscore ("_"). At the beginning of development, a short serial number was used when the first extra zeros from the serial number were truncated but later was decided to use the whole serial number for the label. Unrecognized instruments are not displayed in this list, so the user must check by himself if all required instruments are present. Extra details could be found in the log files.

```
label : type
63600-5_636000000182 : L
LMG670_681501 : M
61505_405 : S
```

Figure 7 EMTS outputs list of instruments and their type

With displayed instruments also revealed the main menu, which has options that are displayed in Table 6. The problem with this menu is that the association about functionality for buttons is not finished. It means that letter "w" is not present in the phrase "Run the main test" at all, but this letter is used because before, there was a different phrase as "Work". There is still such an issue for the last version of the program because fixing this menu task has low priority. The main target is to create an advanced GUI where console output will not be used, so there is no need to fix it at that moment.

Table 6 Main menu options

Button	Action
w	Run the main test
t	Run pre-test
s	Stop running test
p	Pause running test
u	Un pause running test
d	Show devices
r	Reload/load parameters file
h	Print main menu
i	Instrument parameters download/upload
e	Exit

The last step before starting the test is to set up a parameters file which is called parameters.txt. The test will be able to be started after a user presses the button "r" to load parameters and the parameters file is valid. The program will notify the user if the user's settings in the parameters file are not valid.

4.3 DeviceList.info and how EMTS find instruments

In the config folder for EMTS, there is the file DeviceList.info. It has a description for communication information for supported devices. It has the following structure: response string which identifies instrument, response structure and driver, which is needed for this instrument.

Before describing how EMTS find instruments, there should be described how works command *IDN for programmable instruments. For instruments that were used for this thesis have similar logic. Command *IDN returns a string that has different values divided by comma.

For example, when sending command *IDN for instrument 62050P, it returns a message with the following content: *CHROMA,62050P-100-100,03.00,00604.* This line can be made out according to manual as next:

- CHROMA is the manufacturer.
- 62050P-100-100 is the model name.
- 03.00 is the Firmware version.
- 00604 is the serial number.

The first two values present information about the instrument used by EMTS to identify the instrument. For example, in DeviceList.info, it is written as:

CHROMA,62050P-100-100;MAN,MOD,SW,SN;Chroma62000P;

A semicolon separates this line. Each part has its meaning: the first element is the string to identify the instrument, the second element stores the structure of the returned string of *IDN command, and the third element stores the driver's name for this instrument. For the program, it works as follows:

1. The program collects connected instruments by sending the *IDN command to each device and awaiting a return. The device is added to the pool if devices answered on request.

2. From the pool of connected devices, each *IDN return is compared to each available first elements of the DeviceList.info to recognize the instrument.

3. When the device was recognized, its *IDN return is applied on the return template on the appropriate instrument from DeviceList.info. In the program is created dictionary which uses template's identifiers as keys and *IDN's values as values for these keys. It allows later to use *IDN's values independently. For example, the instrument's serial number is stored for the SN key.

4. The program creates an object instrument and connects it to a proper driver name.

All supported instruments by EMTS are stored in the file DeviceList.info. This file is the only option to get to know which instruments user can use. It should be noted that this file exists to simplify adding a new instrument for use. In this case, it simplifies work for the programmer.

4.4 Prepare parameters file

An essential part of EMTS is the parameter file. The user must prepare this file before using the program. This file stores settings for output files, sequence parameters, and pass/fail criteria. It was planned to create GUI which could simplify creating parameter files, but it is all the user's responsibility for the current version. Unfortunately, this file does not follow any modern standards what can make difficulties for new user.

The file structure is quite simple: Each line comprises two parts divided with the sign equal "=" to the left and right parts. The left part is the parameter which is not case sensitive and is used by the program to identify parameter. The right part is the value for the parameter, and for some parameters, it is case sensitive. For example, the user's input for LOGNAME is case-sensitive because the output file name and style for output files could be different depending on the project.

In table 7 is presented the list of available parameters for the parameters file. Parameters with an asterisk sign must present in any parameters file. Otherwise, the program will report an error in the paraments file. Also, for this table, some parameters set settings

for "criteria" and "devices". These parameters have compound value, and they depend on another user's parameters. It should be noted that parameter can be compound as well. The structure for compound parameters and compound values are static, and the user must follow it.

Table 7 Available parameters for parameters file

Parameters	Description	Example
*LOGNAME	This command defines the name of the file where records are stored with a timestamp.	`LOGNAME=DUT_test` Output file: `DUT_test (20191022_1313).csv`
DECIMALSYMBOL	This command defines a symbol between integer and fractional parts for output files.	`DECIMALSYMBOL=.`
LISTSEPARATOR	This command defines a symbol that separates columns in output CSV files.	`LISTSEPARATOR=;`
MEASUREDELAY	This command defines delay before making measurements in seconds	`MEASUREDELAY=3` `MEASUREDELAY=2.7`
*SOURCEWAIT	This command defines delay after the source's settings were applied in seconds	`SOURCEWAIT=10` `SOURCEWAIT=7.7`
SOURCEPAUSE	This parameter can be TRUE or FALSE TRUE means: the program is paused for unlimited delay after all sources outputs are ON. Until it is resumed by the user with the pressed button "u". FALSE means: Test program runs without additional delays	`SOURCEPAUSE=TRUE` `SOURCEPAUSE=FALSE`
ZIMMERSOFTRANGE	This parameter allows the user to enable the feature to use ranges for ZesZimmer.	`ZIMMERSOFTRANGE=TRUE` `ZIMMERSOFTRANGE=FALSE`
ZIMMER	This parameter commands the program to use ZesZimmer (TRUE) or Source and Load (FALSE) as a measurement instrument.	`ZIMMER=TRUE` `ZIMMER=FALSE`
*CHANNELS	This parameter has a compound value, and it bounds the actual device with a logical channel of the program. It is also used in CSV output file	`CHANNELS=S1 1, L2 2`
*MEASURE	This parameter has a compound value, and it defines a set of measurements for each channel Note: Error message produced if a measurement is not possible	`MEASURE=FETCHITRMS 1, FETCHUTRMS 2`
PASSFAIL	This parameter enables (TRUE) or disables (FALSE) pass-fail criteria.	`PASSFAIL=TRUE`

Parameters MEASURE and CHANNELS are parameters with compound values. Each of them has an array of values which is divided with comma sign. Each element of the array has a collection of two arguments. For CHANNELS, the first argument is a logical instrument label, and the second argument is a logical channel. These two arguments create a bounding instrument with the channel. Each argument must be unique. Also, this parameter defines which instruments are present in the test. This need in the channel come from some multi-channel devices. For example, in "`CHANNELS=S1 1, L1 2`", instrument S1 has channel one, and instrument L1 has channel two.

For MEASURE, the first argument is a measure command, and the second argument is a logical channel that creates. These two arguments create a bounding between measurement and instrument through the logical channel. A limited amount of measurements can be set for the first argument, but mainly it is limited by the instrument by itself.

Listing 1 presents an example of the correct Parameters file. There are present mandatory parameters and compound parameters. "`S1_MODEL`" is an example of a compound parameter for an instrument. It should be noted that all device parameters start with a logical label. Compound parameters use underline "_" to divide command into smaller parts. The number of parts can be two or greater. The program will display the error if the amount of parts is less than two. For this example, the first part is defined in parameter CHANNELS by the first argument. The second part defines the task of the parameter. For example, "`S1_MODEL=61505_605`" is the parameter where the task is to define a model name for device S1.

```
LOGNAME=DUT_name
MEASUREDELAY=3
SOURCEWAIT=10
ZIMMER=TRUE
MEASURE=FETCHITRMS 1, FETCHUTRMS 1, FETCHITRMS 2, FETCHUTRMS 2
CHANNELS=S1 1, L1 2

S1_MODEL=61505_605
S1_VOLT_MIN=50
S1_VOLT_MAX=60
S1_VOLT_DELTA=10
S1_COUP=DC

L1_MODEL=63201_548
L1_MODE=H
L1_AMP_MIN=0
L1_AMP_MAX=100
L1_AMP_DELTA=10
```

Listing 1 Example of correct parameters.txt

Some parameters for devices required the third part after the task part. In most cases, these parts are used to define which values must be applied for an instrument, and usually, these parameters work with other similar parameters. Table 8 present which the third parts can work together otherwise program does not proceed and throw message for user that there is conflict in user's input. Values which are set for such parameters are absolute.

Table 8 Basic options of user's input for values commands

	Option 1	**Option 2**	**Option 3**	**Option 4**
CONST	X			
MIN		X	X	
MAX		X	X	
DELTA		X		
STEPS			X	
CUSTOM				X

Table 8 presents four option for how a user can set up EMTS measurements sequence. Each instrument or channel, if such is present, can only use one option at one time. However, different instruments can have different options. For example, the parameter value for the source can be constant, and parameters values for the load can be set as in option two.

The simplest option is the first one when an instrument has a constant value during the test. For example, it is needed when the user wants to check the device under different loads with constant source value.

Option two and option three are similar, with a slight difference in the calculation of values. Both options have minimum and maximum value, which the program cannot exceed. These two options create consistent sequences for instrument values. However, option two calculates input values with delta, which means every next step will be different by delta value. When the next value exceeds the maximum, it will be truncated to max value. In option three is a defined number of steps. It should be noted that the STEPS parameter does not include the first value, which is MIN, from the parameters file. That means when the STEPS parameter equals four, then EMTS will do five values for the instrument.

The last fourth option allows the user to define an array of values for the instrument. This option was created for situations when all the rest options are not suitable for a specific

task. In addition to that, this option allows user to create inconsistent sequences. This feature opens new capabilities for EMTS.

4.5 Main Test Sequence (MTS)

During development, a solution which divides EMTS responsibilities into two independent features was created: creating test sequence and performing it. EMTS checks that the user's data is valid before creating a test sequence. EMTS validates data from the parameters file what includes the check which measurement input is in use from table 8.0.

There is the universal sequence which is the core of EMTS, but it should be changed in future. However, this solution covers current requirements. Current logic works in the following way: EMTS has values for instruments that are connected to DUT. There are values for Input and Output of DUT. Input represents as source instrument and output as load(s) instrument(s). MTS uses nested loops to create a one-dimensional array where outer loops are for source values and inner loops for load values.

Listing 2 presents an example where a source has values from 210 volts to 230 volts with delta 10 volts and constant frequency 50 Hertz and load from 10 Wats to 50 Wats with delta 10 Wats. Source loop values will look next: 210V 50Hz, 220V 50Hz, 230V 50Hz. Load loop values will look next: 10W, 20W, 30W, 40W, 50W. The program will apply each load value for each source value. Each set of load value with source value is called test values, and only one set can be in one period of time.

```
S: 210V 50Hz, L: 10W
S: 210V 50Hz, L: 20W
S: 210V 50Hz, L: 30W
S: 210V 50Hz, L: 40W
S: 210V 50Hz, L: 50W
S: 220V 50Hz, L: 10W
S: 220V 50Hz, L: 20W
S: 220V 50Hz, L: 30W
S: 220V 50Hz, L: 40W
S: 220V 50Hz, L: 50W
S: 230V 50Hz, L: 10W
S: 230V 50Hz, L: 20W
S: 230V 50Hz, L: 30W
S: 230V 50Hz, L: 40W
S: 230V 50Hz, L: 50W
```

Listing 2 Example of prepared values for test

Each element in the set takes time to be applied for an instrument. For this case, EMTS is optimized to change values that must be changed, and it does not apply value that is already set for the instrument. For Listing 2, the EMTS will change the value for source only three times and for load 15 times, instead of changing values independently for all instruments and getting 30 changes that must be applied.

MTS creates a loop for each instrument, and it contains values only for one instrument. Then program includes one loop in another where source values are the first loops, and load loops are the last. For the last version of the program, there is no ability to change this order.

Figure 8 presents each test's main logic, which is performed after the user started a main test or pre-test sequence. It should be noted that test's sequences are created before starting. Therefore, EMTS can estimate how many steps it is needed to complete the test. This estimation is needed to inform the user about progress on the progress bar. The progress bar is presented as a simple output of the percentage of completed work.

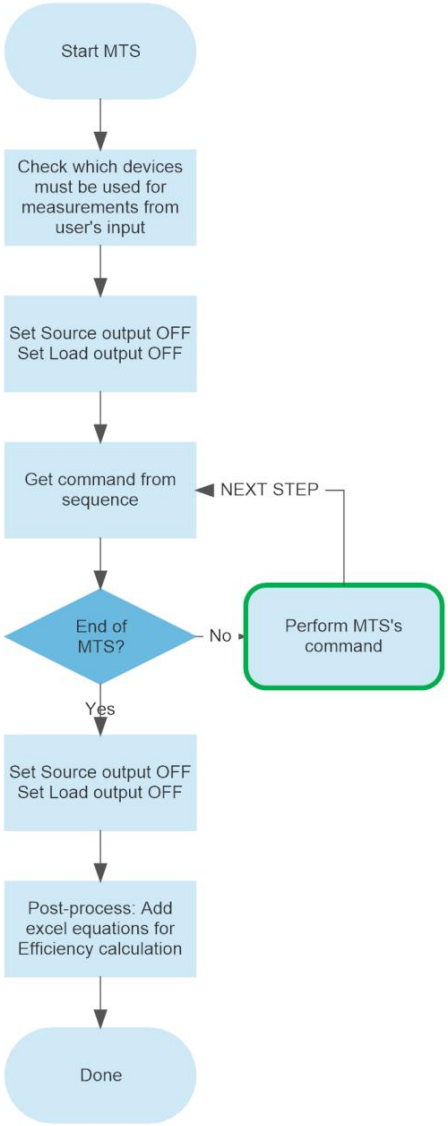

Figure 8 MTS logic

Figure 8 presents a flow chart of the execution of MTS. When MTS starts, it checks if all required devices for the test are present. However, EMTS checks only the exact device by serial number, and if the program cannot find it, it aborts sequence. From the program point of view, similar models but with a different serial number are not counted as the same instruments.

EMTS works with instruments consistently. That means that it does not work in parallel with instruments. Therefore, there are no unexpected cases from asynchronous work. However, EMTS is a multithread application, but a different thread is used for listening for user input. For example, the user can stop or pause the execution of MTS.

EMTS turns off all output for connected sources and loads before applying the first value. It should be noted that MTS's responsibility is to turn the output on for instruments at the correct time. When all output is off, then it is safe to start the test. The test is the loop that takes one command from a sequence and performs them. Figure 9 presents four main actions:

- Change output value for source instrument
- Change output value for load instrument
- Make measurement
- Wait for delay for source

These actions are depended on the user's parameters file. However, some actions allow a user to control the loop during runtime. These actions are pause and stop actions which offer the user more flexibility during the test run. For example, it is possible to pause the test if the user noticed something odd during a test. In addition to that, it is possible to stop the test. It should be noted that before pause or stop the test, EMTS must perform till the end a current action.

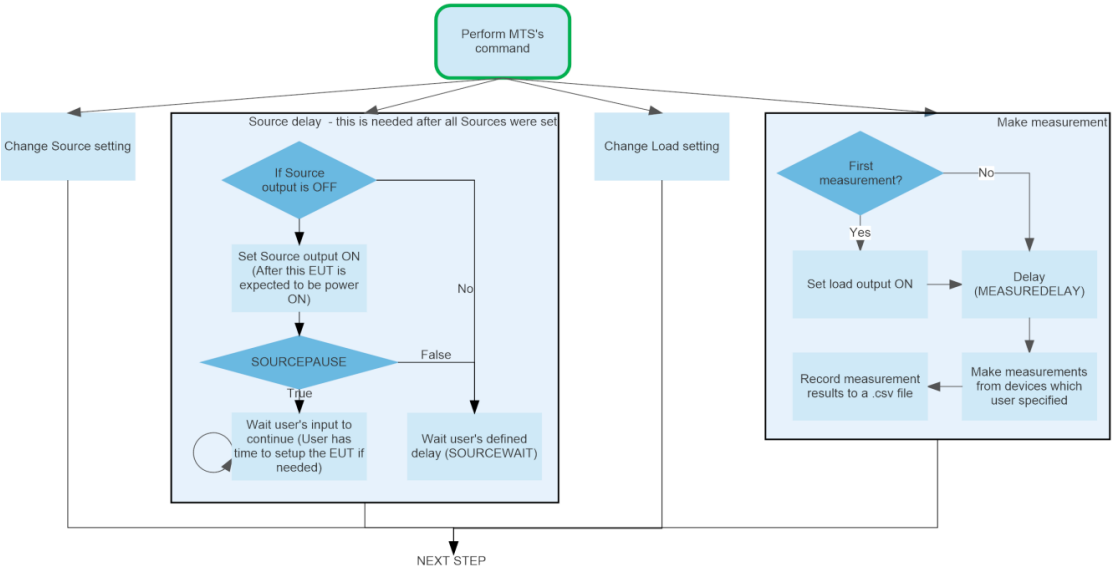

Figure 9 Actions for test performer

Commands in sequencer which are responsible for changing values for instruments are pretty simple. They contain instruments short name, the number of commands and commands themselves. Several commands can be different because one instrument can be applied with different values at the same time. For example, For AC source, it is

commonly required to set voltage and frequency. At the same time, the load can accept only one value at one time.

The source control command does not set values for the instrument. This command is responsible for making delay after the source value was set. When the source was set for the first time, the program allows the user to continue the test manually. There is an option in the parameters file that allows the user to force EMTS to wait until the user presses the button. It is needed for cases when DUT turns on and must be set up after the source output was on.

The measurement part contains two commands. There are delay and perform measurement commands. In addition to that, this part checks that output from all loads is present. If there is no output, then EMTS turns the output on for all loads. User can set delay, which should be performed before making a measurement. It is needed to wait until output for load would stabilize after its change. Then happen measurement itself, and this action can be done by source and load instruments or any specific programmable instrument, for example, Zes Zimmer. In the end, received data from measurement is stored in a CSV file.

4.6 User interface

For the latest version of EMTS, there was implemented CLI. This interface is easy to support, and it was suitable for the first time for people who were going to use it. It should be noted that there were attempts to create a graphical user interface. However, it was decided to use a CLI user interface because usually, it is time-consuming to introduce a new feature in a graphical user interface.

Figure 10 presents the latest version of the welcome message and main menu output. It was requested to have a welcome message and include the program's name, version, and connected devices to the computer where a program runs. The program's name is written at the beginning of the first line and has a more decorative meaning. On the second line is located the program's version and the date when the build was compiled. After that, the program lists devices to which were able to connect and recognize by the program.

```
Windows PowerShell
Welcome to Efore mini test system
Version: 0.4_3 (2020/02/06)
New device added: 63600-5_636000000182
New device added: 61505_000405
New device added: LMG670_00681501
New device added: 63201_00000548
New device added: 61505_000406
New device added: 63201_00000549

button / action
w - run main test
t - run pre-test
s - stop any test
p/u - pause/unpause running test
d - show devices
r - reload/load parameters file
h - print this menu
i - instrument parameters download/upload
e - exit
When parameteres file is ready press 'r'
Use next labels to setup input file:
label : type
63600-5_636000000182 : L
61505_000405 : S
LMG670_00681501 : M
63201_00000548 : L
61505_000406 : S
63201_00000549 : L
```

Figure 10 Welcome message and main menu in latest CLI

The main menu output includes descriptions for actions and buttons which is responsible for these actions. This menu can also be called when the user pressed the dedicated button (h). It should be noted that the user's input can be in upper and lower case. After main menu actions, there is a reminder that the parameter file should be set and reloaded before start. In the end, there is a list of instrument model names with serial numbers that the user should use in the parameters file.

5 Results and discussion

The development of EMTS is still ongoing. However, the program is already in use for actual tasks. Such an approach allows to gather feedback from the R&D team and make changes in the program before it becomes complicated and every change will be challenging to implement. This chapter contains the employer's feedback and thoughts about future improvements for EMTS.

5.1 Employer's feedback

In general, EMTS met positive reviews from co-workers as from management. The program can be improved unlimitedly. However, correct prioritization helped implement the most valued features of the program for the R&D team first. For example, in priority for the R&D team was to have more functionality and flexibility and the team was consonant to use simple CLI for the first time.

This project was made for Bachelor’s Thesis, so the R&D team considered that only one person should write the program's code. In addition to that, EMTS was created for internal use. For this reason, there were not strict deadlines. However, sometimes it was possible to implement new features faster than was planned. Consequently, it was appreciated by the employer.

5.2 Future improvements

There are plenty of features that could be implemented for EMTS, and they are in the list of requests. However, there are two main tasks which should be done

- Unit test
- GUI

Unit tests are needed to make future developing smother and safe. It was noted that they should be present from the begging of the project. Still, this project was started as experimental, and some main rules which apply to modern development were omitted.

The second task for improvement is GUI which must be more pleasant and convenient for the user. There were attempts to create a suitable GUI during development, and the first versions of EMTS had the most common GUI. However, later heaps of time to adjust GUI for any new feature was spent, so it was decided to simplify the user interface to CLI. For example, one of the problems was when GUI had to dynamically add new views or remove them depending on the instrument's amount.

In addition to these main tasks, some modules should be refactored for future development. One of them is which parses user input for EMTS.

6 Conclusion

This thesis aimed to create a PC program that could decrease the required amount of human interaction and save time for repeatable tests performed during development for Efore's products. Specifically, the program should be used to do the following: automatically connect to the programmable instruments, send commands to them, and store received data.

This objective was completed, and all mandatory requirements were completed as well. Also, the solution was tested by Efore's R&D team and got positive feedback. However, there are already requirements for future development and improvement of EMTS. In addition to that, all the most essential parts of realization were explained in this thesis.

Nevertheless, this was a large project, and the things we learned could be helpful in the future. It was exciting to see how the solution helps not only in theory but also in actual tasks. This gives motivation to continue the EMTS project in the future.

References

1 Bernard Homès and Bernard Homès. 2012. Fundamentals of Software Testing. First edition. [online] Available at: https://ebookcentral.proquest.com/lib/metropolia-ebooks/detail.action?docID=1120766 [Accessed 05 November 2020]

2 Python Software Foundation. The Python Tutorial [online]. Available at: https://docs.python.org/3.7/tutorial/index.html. [Accessed 30 December 2019]

3 David Halliday, Robert Resnick, Jearl Walke. 2013. Fundamentals of Physics. 10th edition USA: John Wiley & Sons.

4 Bill Gatheridge, Product Manager. Power Measuring Instruments, Yokogawa Corporation of America. 2014. Electric Motor Power Measurement and Analysis. [online] Available at: https://www.yokogawa.com/library/resources/media-publications/electric-motor-power-measurement-and-analysis/ [Accessed 30 December 2019]

5 National Instruments Corp, NI-VISA product details [online] Available at: http://www.ni.com/fi-fi/shop/select/ni-visa [Accessed 30 December 2019]

6 IVI Foundation. October 19, 2018. Systems Alliance, VPP-4.3: The VISA Library. Revision 7.0 [online] Available at: http://www.ivifoundation.org/downloads/Architecture%20Specifications/IVIspecstopost10-22-2018/vpp43_2018-10-19.pdf [Accessed 30 December 2019]

7 SCPI Consortium, "1999 SCPI Syntax & Style," IVI Foundation, 1999 [Online]. Available: http://www.ivifoundation.org/docs/scpi-99.pdf. [Accessed 6 January 2019].

8 Chroma. October 2015. Programmable AC Source 61505 User's Manual. Version 1.8 [online] Available at: https://www.chromausa.com/document-library/manual-61500-series/ [Accessed 30 December 2019]

9 Chroma. September 2015. High Power DC Electronics Load 63200 Series Operation & Programming Manual. Version 2.9. [online] Available at: https://www.chromausa.com/document-library/manuals-63200-series/ [Accessed 30 December 2019]

10 ZES ZIMMER Electronic Systems GmbH. December 2019. LMG641 Precision Power Analyzer Brochure. [online] Available at: https://www.zes.com/en/content/download/741/7292/file/ZES_LMG641_final_web_hi_EN.pdf [Accessed 30 December 2019]

11 SCPI Consortium. [online] Available at: https://www.ivifoundation.org/docs/scpi-99.pdf [Accessed 30 December 2019]

12 John V. Guttag: Introduction to Computation and Programming using Python. [online] Available at: https://doc.lagout.org/programmation/python/Introduction%20to%20Computation%20and%20Programming%20using%20Python%20%28rev.%20ed.%29%20%5BGuttag%202013-08-09%5D.pdf [Accessed 30 December 2019]

13 Python Software Foundation. About Python [online] Available at: https://www.python.org/about/apps/ [Accessed 30 December 2019]

14 wxPython. Overview of wxPython. [online] Available at: https://wxpython.org/pages/overview/ [Accessed 30 December 2019]