Potku Project

Jarkko Aalto
Timo Konu
Samuli Kärkkäinen
Samuli Rahkonen
Miika Raunio

Application Report

Public
Version 0.2.0
15.5.2013

University of Jyväskylä

Department of Mathematical Information Technology

Jyväskylä

Approved by	Date	Signature	Clarification
Project manager	__.__.2013		
Customer	__.__.2013		
Instructor	__.__.2013		

Document Info

Authors:

- Jarkko Aalto (JA) jarkko.t.aalto@student.jyu.fi
- Timo Konu (TK) timo.j.konu@student.jyu.fi
- Samuli Kärkkäinen (SK) samuli.p.p.karkkainen@student.jyu.fi
- Samuli Rahkonen (SR) samuli.p.j.rahkonen@student.jyu.fi
- Miika Raunio (MR) miika.o.raunio@student.jyu.fi

Document name: Potku Project, Application Report
Page count: 39

Abstract:
Potku project developed an user interface software application for analyzing data received from a recoil spectrometer. The application receives ascii-format list data from the spectrometer. Using the data, the application can draw a time-of-flight over energy histogram (ToF-E histogram), and has further analysis tools based on selections done in the ToF-E histogram. This document describes the backgrounds of the application as well as presents the user interface and the structure of the application. The programming and testing practices are also presented. The realization of the functional requirements and other agreed upon objectives is given, as well as advice for future development.
Keywords: Application structure, future development, meeting the requirements, programming practices, Python, Qt, recoil spectrometer, testing, user interface.

Project Contact Information

Project group:

Aalto Jarkko	jarkko.t.aalto@student.jyu.fi	
Konu Timo	timo.j.konu@student.jyu.fi	
Kärkkäinen Samuli	samuli.p.p.karkkainen@student.jyu.fi	
Rahkonen Samuli	samuli.p.j.rahkonen@student.jyu.fi	
Raunio Miika	miika.o.raunio@student.jyu.fi	

Customers:

Sajavaara Timo	timo.sajavaara@jyu.fi	040-8054114
Laitinen Mikko	mikko.i.laitinen@jyu.fi	0400-994836
Julin Jaakko	jaakko.julin@jyu.fi	040-8054097
Arstila Kai	kai.arstila@iki.fi	–

Instructors:

Santanen Jukka-Pekka	santanen@mit.jyu.fi	040-8053299
Tuovinen Tero	tero.tuovinen@jyu.fi	050-4413685

Contact information:

Email lists	potku@korppi.jyu.fi potku_opetus@korppi.jyu.fi
Email archives	https://korppi.jyu.fi/kotka/servlet/list-archive/potku/ https://korppi.jyu.fi/kotka/servlet/list-archive/potku_opetus/

Version History

Version	Date	Modifications	Modifiers
0.0.1	25.4.2013	The report template was created.	MR
0.0.2	29.4.2013	First drafts of introduction and terminology chapters were written.	MR
0.0.3	30.4.2013	First draft of background chapter was written.	MR
0.0.4	3.5.2013	First draft of application structure chapter was written.	MR
0.0.5	5.5.2013	First draft of programming practices chapter was written.	MR
0.1.0	7.5.2013	First drafts of user interface and testing practices chapters were written.	MR
0.1.1	8.5.2013	First drafts of realization of objectives, guide for future developers and summary chapters were written. Improved user interface and testing practices chapters.	MR
0.1.2	13.5.2013	Small fixes across the document based on instructor Santanen's feedback.	MR
0.2.0	15.5.2013	Improved interface, application structure, programming practices and testing chapters.	MR

Contents

1 Introduction

The research team of accelerator-based material physics at Department of Physics of University of Jyväskylä use a accelerator to collide a projectile beam with a sample object, which causes particles from the sample to be ejected. The research team uses a recoil spectrometer to collect information on the particles that were ejected from the sample. The data contains the time-of-flight and energy of each particle the spectrometer detected. To analyze this data, they need a software application, which was developed by the Potku project in the spring of 2013.

Potku project designed and developed an application. The application can draw the data points received from the spectrometer into a histogram, allow the user of the application to select chemical elements from the histogram, and use the selected elements to produce elemental losses histograms, energy spectrum histograms and depth profile histograms. Most mathematical calculation necessary for some tools of the application is done by external C components provided by the customer.

The developed application was named Potku, after the name of the project itself. Potku software was developed to work under Windows, Linux and Mac operating systems, and it was programmed using the Python 3.3 programming language. Potku software uses a few external analyzation programs programmed in C, that were developed by the customers.

Documents were written during the project to describe the developed software and the project. The requirements specification [1] contains the full list of requirements set for the application. The realization of the goals and the practises are described in the project report. Kuvatus Project Application Report [2] was used in compiling this document.

In Chapter 2 the essential terminology used in this document is defined. Chapter 3 describes the backgrounds of the project. In Chapter 4 the user interface is presented and the essential functionality demonstrated. The structure of the application is described in Chapter 5. The used programming practices and a sample of the code are presented in Chapter 6. Chapter 7 reports the carried out testing and the results. The realization of objectives set for the application is described in Chapter 8. Finally, Chapter 9 contains recommendations and suggestions from the project team for future development of the application.

2 Terminology

The chapter explains the essential terminology used in the document.

2.1 Target Area and Application

Accelerator laboratory is a laboratory operating in the Department of Physics of the University of Jyväskylä. **The research team of material physics** works in the laboratory, and utilizes ion beams to research the composition of materials.

Chemical element is an atomic particle with a certain number of protons in it's nucleus. Elements are listed in the periodic table of elements, in which the elements are arranged according to the number of their protons.

Depth profile projects the amounts of elements in the thin film as a function of depth.

Elemental losses are events, in which the amount of certain elements in a sample are less after the experiment than before the experiment.

Finlandia is the software application, that the research team is currently using to analyze the data received from the recoil spectrometer. Finlandia utilizes components written by Kai Arstila.

Ion is an atom with an electric charge, because of an uneven amount of electrons and protons.

Isotope is a variation of an element with different amount of neutrons in it's nucleus.

Project is a collection of different measurements, in which there may be numerous experiments done to a sample with different parameters.

Recoil spectrometer is a research device used by the research team, that is used to collide ion beams from the Pelletron accelerator with a sample, which will eject ions from the sample. The **time-of-flight** and **energy** are measured.

Recoiled ion Is a particle that has been ejected from the sample, and is detected by the ToF-E telescope (ERD).

Sample is a thin film often cultivated over silicon, which could be for example aluminum oxide Al_2O_3.

Scattered ion is part of the ion beam launched from the accelerator, which has collided with the sample and scattered towards the Tof-E telescope.

Thin film is the material under research. Materials analyzed in a same project may have been for example created in different temperatures. The thickness of the film is typically between 10–300 μm.

Time-of-flight calibration is a procedure which transforms the time-of-flight received from the channels to seconds or nanoseconds.

ToF-E histogram is an abbreviation of Time of Flight - Energy histogram. In the histogram, recoiled and scattered ions are demonstrated as data points with the functions of time-of-flight and energy. In this histogram, concentrations spectrums of different elements are often called "bananas".

2.2 Software and Techniques

C is a programming language. The external analyzation components written by Kai Arstila and Jaakko Julin called in the application are written in C.

Eclipse is a integrated development environment for different programming languages. The project team used Eclipse for writing the source code of the application.

Egit is a Git extension for Eclipse, which enables using Git from within Eclipse.

Git is a distributed revision control software for managing source code and documents.

Matplotlib is a graphical library for plotting two dimensional graphs for Python.

NumPy is a Python extension, which enables the usage of large, multidimensional arrays.

PyDev is Eclipse extension, which enables Python programming in Eclipse.

PyDoc is a tool for the automatic generation of class documentation from Python classes.

PyQt is a Python binding for the graphical user interface toolkit Qt, which is used for creating the graphical user interface of the application.

Python is the programming language used in the development of the application.

SciPy is a Python extension, which enables the usage of various mathematical algorithms.

YouSource is a WWW-based source code release system. Yousource supports Git revision control.

3 Background and Goals

In Department of Physics of University of Jyväskylä operates an accelerator laboratory, which researches accelerator-based physics. In the accelerator laboratory operates a research team of accelerator-based material physics. The research team researches the composition of materials by accelerating an ion beam from the Pelletron accelerator, which collides with a sample of a material, and ejects particles from it. These particles are detected by a recoil spectrometer. The time-of-flight and energy of each particle is detected, which can be used for analysis. An individual measurement lasts for hours and can produce several million lines of data. Research subjects are usually provided by a customer, who wants to find out the composition of some object. For instance, a jewelry company could deliver to the research team samples taken from a silver trinket for them to analyze.

There are several tools of analysis. **ToF-E histogram** plots each data point received from the recoil spectrometer as a function of time-of-flight and energy. Each chemical element tends to form a mass in the histogram, which can be marked for further analysis. **Elemental losses** can be analyzed to find out how much the amount of each element in the analysis has reduced. An **energy spectrum** can be calculated to analyze distribution of elements by energy and yield of each element. With a **depth profile** the concentration of elements can be analyzed in different depths of the sample. Each of these tools have their set of sub-tools which aid the analysis process.

The research team currently has a application named Finlandia, which they can use to analyze the data received from the recoil spectrometer. It is functional and produces valid data for the most time, but there are some bugs and shortcomings. The research team decided that producing a new application from the start, would be a better option than continuing the development of Finlandia's source code. The research team contacted the Department of Mathematical Information Technology for a user interface application to replace Finlandia, and enable easier further development. The application came to be developed as a Student Software Project named as Potku.

The major shortcoming with Finlandia, which the research team currently uses, is that the code is poorly documented and difficult to develop further. Basic functionality however works and there aren't major drawbacks with user interface either. So Finlandia could be used for reference of correct functionality, and inspiration could

be drawn from it's GUI during the project for the Potku application. After further development the research team intends to publish the application to the scientific community.

4 User Interface of the Application

The user interface was developed using PyQt GUI libraries and MatPlotLib plotting libraries. In Section 4.1 the main window structure of Potku is introduced, while the rest of the sections of the chapter will introduce the functionalities of Potku with more detail.

4.1 Main Window Structure

The main window structure of Potku is presented in Figure 4.1. In the figure, multiple analyzation tools have been activated, so that the full main window interface is enabled.

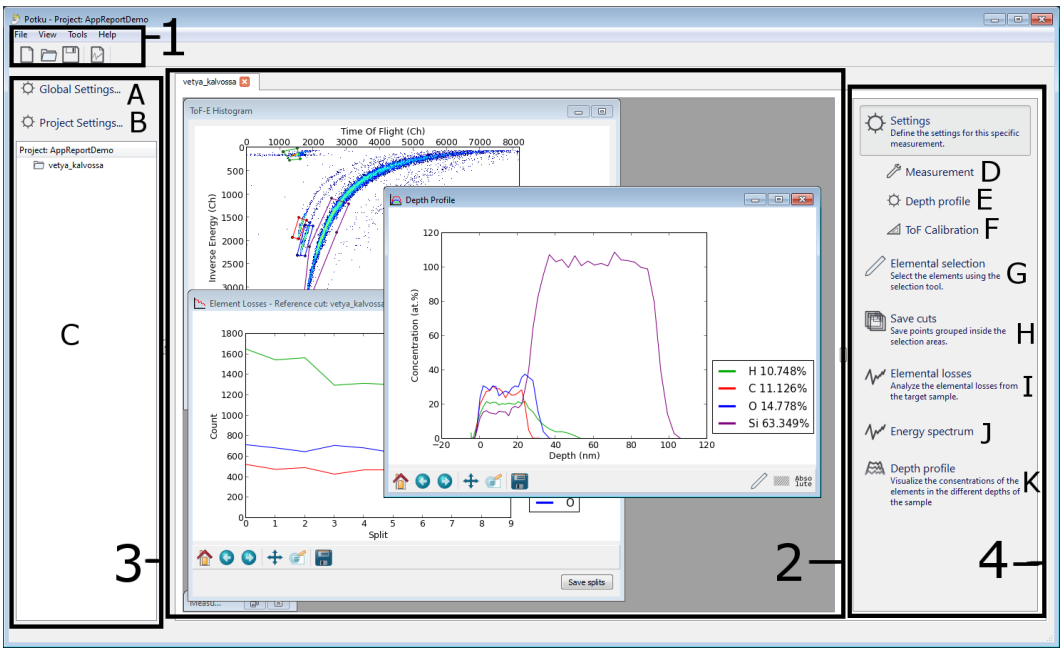

Figure 4.1: Potku main window

There are four distinct components:

1. Menus and top toolbar – Contains all the functionality of Potku.
2. Workspace – Contains all the histograms that Potku generates.
3. Left sidebar – Contains buttons to global and project settings, and the project manager. This can be hidden with the small button next to it.

4. Right sidebar – Contains buttons for all the major the major analysis tools. Is enabled only when a measurement has been loaded into the project. This can be hidden with the small button next to it.

Within these components there are several tools:

A. Global settings – Contains settings that will affect all the projects done in Potku.

B. Project settings – Contains settings that well only effect the current project. Is only enabled when a project is loaded.

C. Project manager – Lists all the measurements loaded into the project.

D. Measurement settings – Contains settings regarding the measurement. Settings defined here can be used to override corresponding parts in project settings.

E. Depth profile settings – Contains settings regarding the depth profile. Settings defined here can be used to override corresponding parts in project settings.

F. ToF calibration settings – Contains settings regarding the time-of-flight calibration. Settings defined here can be used to override corresponding parts in project settings.

G. Element selection – Enables the element selection tool in the ToF-E histogram.

H. Save cuts – Saves the cut files defined in the ToF-E histogram.

I. Elemental losses – Opens the elemental losses analyzation tool.

J. Energy spectrum – Opens the energy spectrum analyzation tool.

K. Depth profile – Opens the depth profile analyzation tool.

4.2 Starting Potku

When Potku opens the interface (Figure 4.2) is significantly less busy than in Figure 4.1. The user has three options: 1) defining global settings, 2) loading an existing project, and 3) creating a new project. Option 1 is demonstrated later in Section 4.5, option 2 is a quite self-explanatory file dialog for opening an existing saved project. For this demonstration we will choose option 3 as demonstrated in Section 4.3.

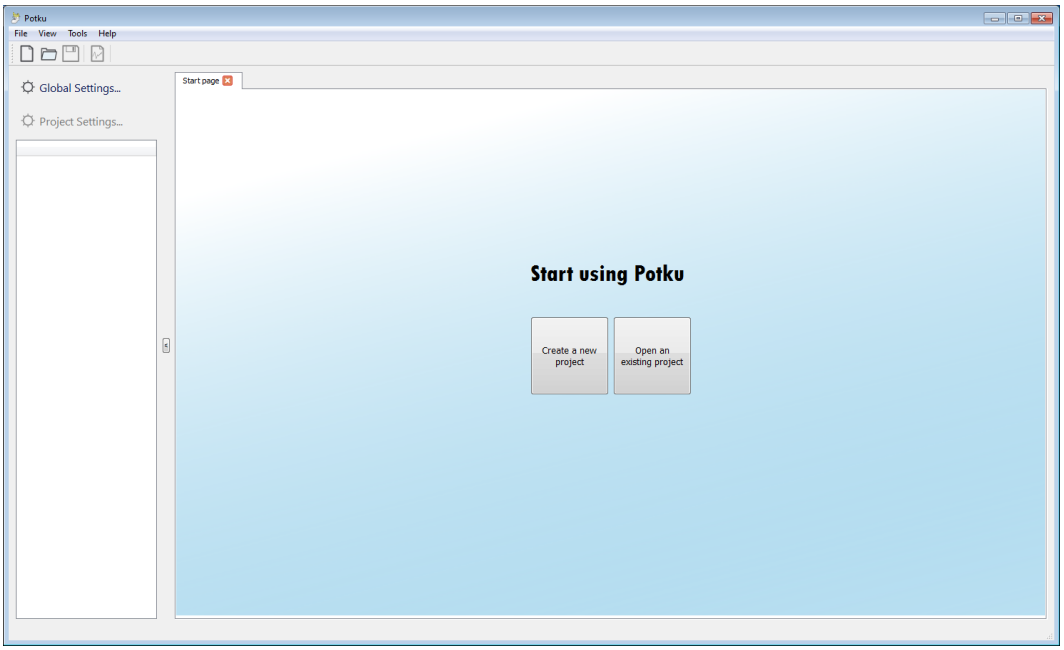

Figure 4.2: The main window with nothing loaded yet

4.3 Creating a project

To create a new project the user can click the big button in the workspace or from the toolbar menu *File* command *New Project*. A dialog opens (Figure 4.3) and asks for a name for the project as well as a directory to which the project is created. Once the user clicks the *Create* button, a project is created and the user is returned to the main window, where *Define Project Settings* button has been enabled, and the buttons in the workspace have been replaced by a *Create a new measurement* button.

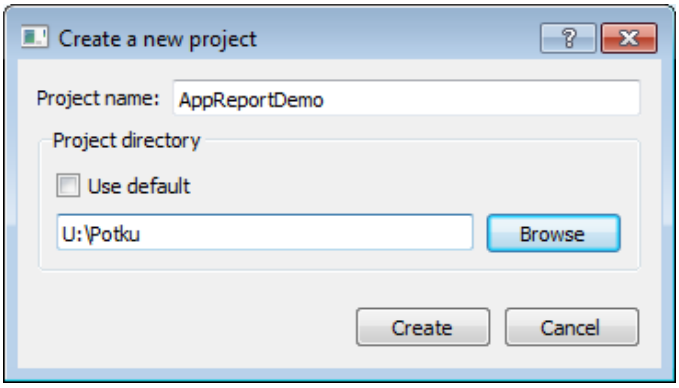

Figure 4.3: Create a new project dialog

4.4 Loading a Measurement and Making Cuts

To load a new measurement, the user can use the large button in the info window that is shown when a new project is created, or from menu *File* command *New Measurement*. This opens a file dialog that accepts an asc-file.

Once the file has been loaded, a new Tof-E Histogram is generated and shown in the workspace (Figure 4.4).

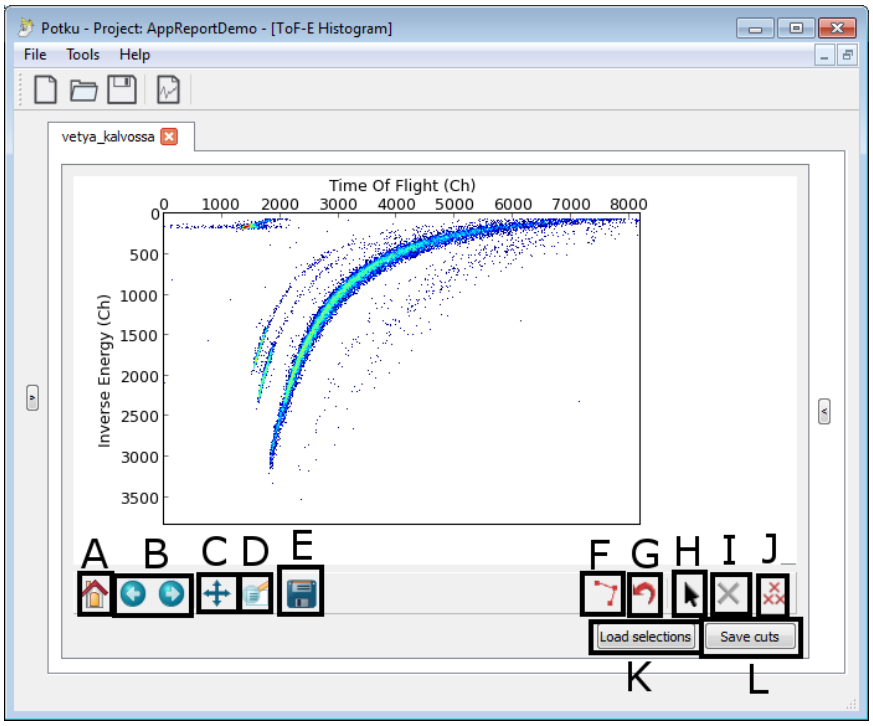

Figure 4.4: A loaded measurement with no selections.

The tools of *ToF-E Histogram* are as follows:

A. Reset original view – Returns the graph to the original zoom level and position.

B. Back/Forward to next view – If the user has zoomed multiple times on the graph, these buttons will allow switching between these zoom levels.

C. Drag – Drag the graph.

D. Zoom – Zoom in on the graph.

E. Save – Saves an image file of the graph.

F. Select element area – Allows the user to select elements from the histogram.

G. Undo last point – Removes the previous node selected with tool F.

H. Select element selection – User can select an element selection done with tool F.

I. Delete selected selection – Delete an selection selected with tool H.

J. Delete all selections – Delete all the selections in the histogram.

K. Load selections – Opens a file dialog to open a file containing existing set of selections to be used in the loaded ToF-E histogram.

L. Save cuts – Saves element selections to cut files.

Tools A–E are provided by MatPlotLib and were not developed by the project team. These buttons are common to all the graphs generated by Potku. Tools F–L were developed by project team and are unique to the ToF-E histogram.

To select elements from the histogram (Figure 4.4), the user has to equip the *Select element area* tool either from the right sidebar or from histogram window toolbar. With the tool the user will set selection nodes with the left mouse button, and complete a selection with the right mouse button or by clicking the first node again with the left mouse button. Once a selection is completed, ToF-E *Selection Settings* dialog opens (Figure 4.5). In the dialog, the user can select the element of the selection, the isotope of said element, a weight factor, whether the element is RBS or ERD and the color used for the selection.

During the selection of a cut, the user can cancel previous nodes by clicking *Undo last point* button. The user can select completed selections with the *Select element selection* tool and re-edit their settings or delete them either with the *Delete selected selection* and *Delete all selections* buttons. Additionally, the user can load existing selections from a file via the *Load selections* button. Once the user is content with the selections, she can save the selections into cut files with the *Save cuts* button. Cut files are used in further analysis.

Graph settings can be opened by right-clicking anywhere in the graph 4.5. This settings dialog contains several options to modify the axes as well the option to switch the histograms color scheme between the default colors and two version of grayscale.

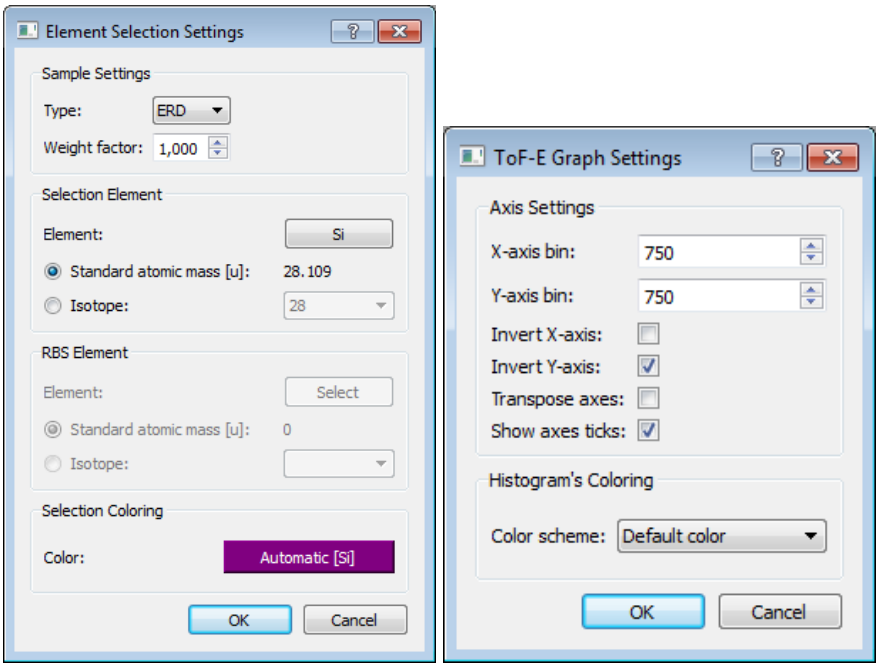

Figure 4.5: Selection settings and graph settings dialogs

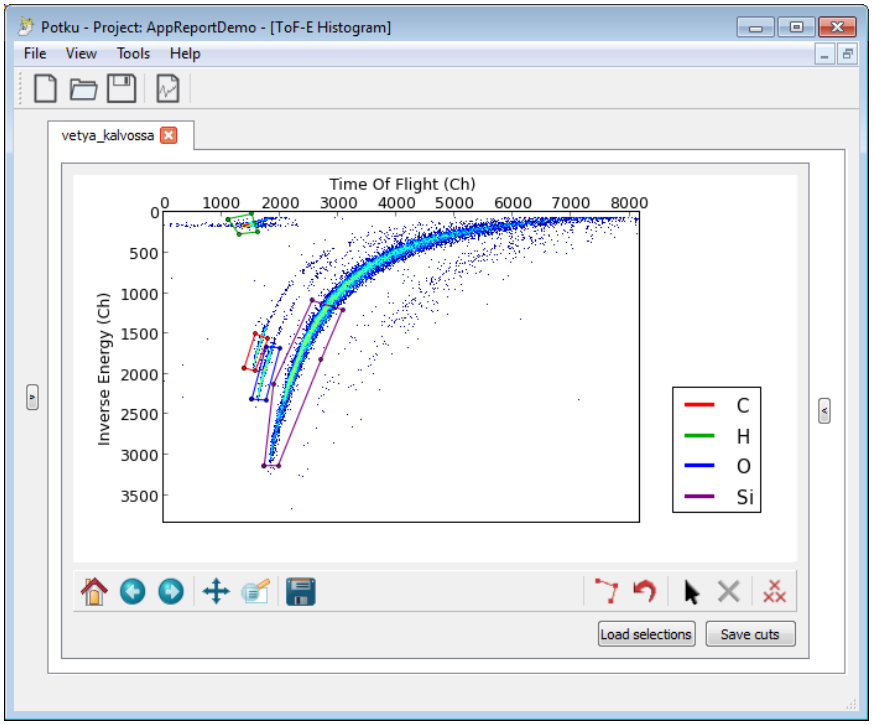

Figure 4.6: A loaded measurement with selection of four elements

4.5 Defining Settings

There are three types of settings regarding the measurement in Potku: Global settings, which affect all the projects. Project settings, which affect only a single project and all the measurements within that project. Measurement settings, which affect only a single measurement.

Project settings contains several parameters regarding the used equipment during the experiment. This settings window is shown in Image 4.7.

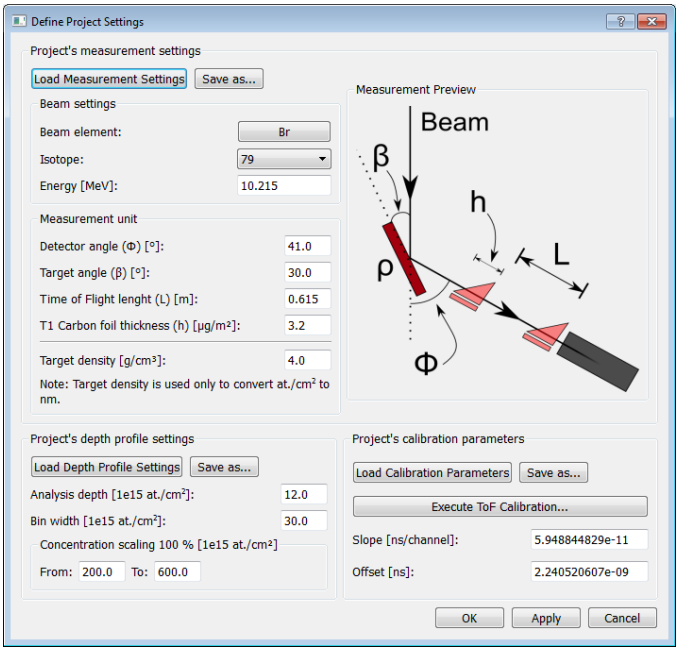

Figure 4.7: Project settings dialog

Global settings contains the parameters to set up the default directory for new projects and the default colors to be used for different elements when they are selected from a ToF-E histogram. The settings window is shown in Image 4.8.

4.6 Time-of-Flight Calibration

The user can perform time-of-flight calibration from the *Calculate ToF Calibration* button in the *project settings*. This will open a dialog with two tabs: fitting and calibration. First the user has to do the **fitting** for each cut file (Figure 4.9). Potku will attempt to automatically calculate the **front edge** of each cut, but the user can also

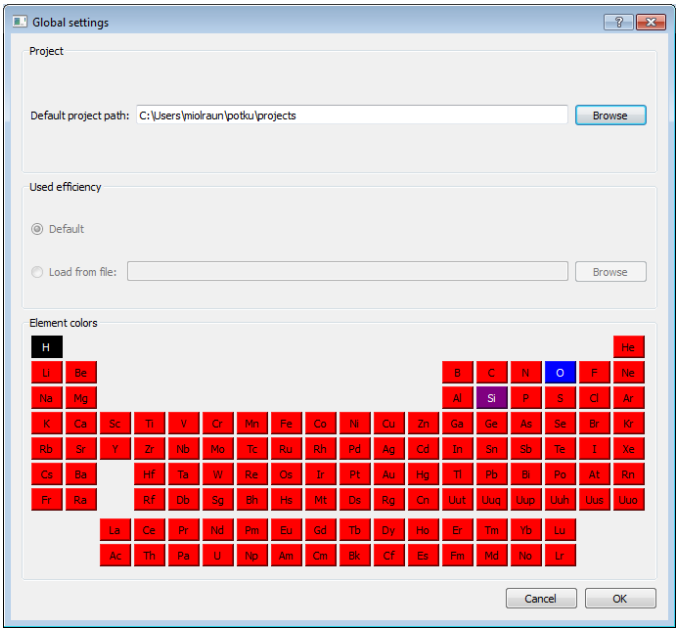

Figure 4.8: Global settings dialog

manually set the edge with the *pen tool*. The fitting graph can also be replotted with the *bin width*, which can result in a more sensible automatic fit. When the user finds a satisfactory fit, she clicks the *Accept point*. This is done for each cut individually.

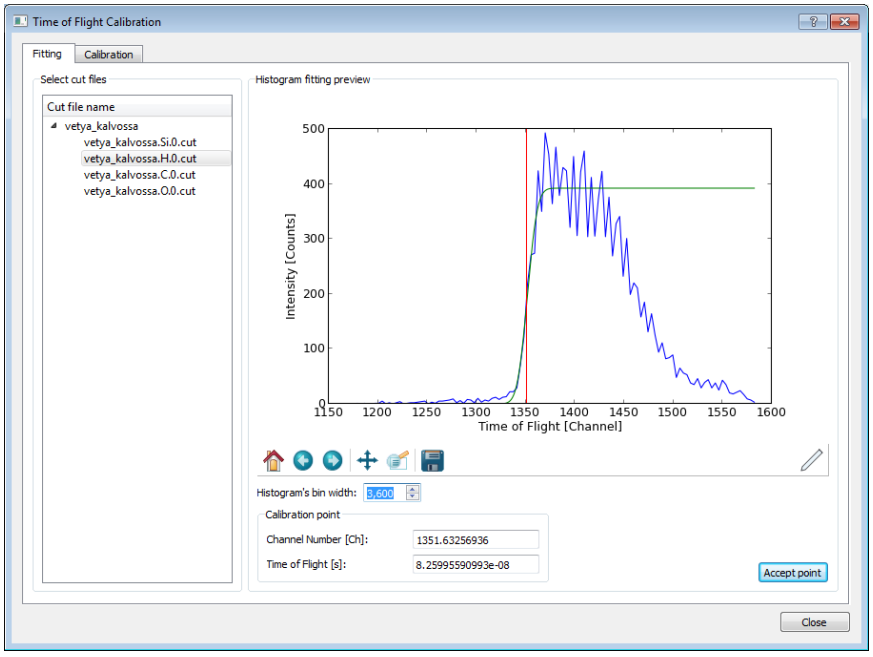

Figure 4.9: The *fitting* tab.

Once the user has accepted a point from each cut, she can switch to the **calibration**

tab (Figure 4.10). In this tab Potku will attempt to do a linear fit between the points accepted in the previous tab. Some of the points may not be satisfactory (in this demonstration, silicon did not quite fit), so 'bad' points can be removed from the calibration in the element list on the left. One the user has found a satisfactory fit, she can click the *Accept calibration* button, which will return the user back to the project settings dialog.

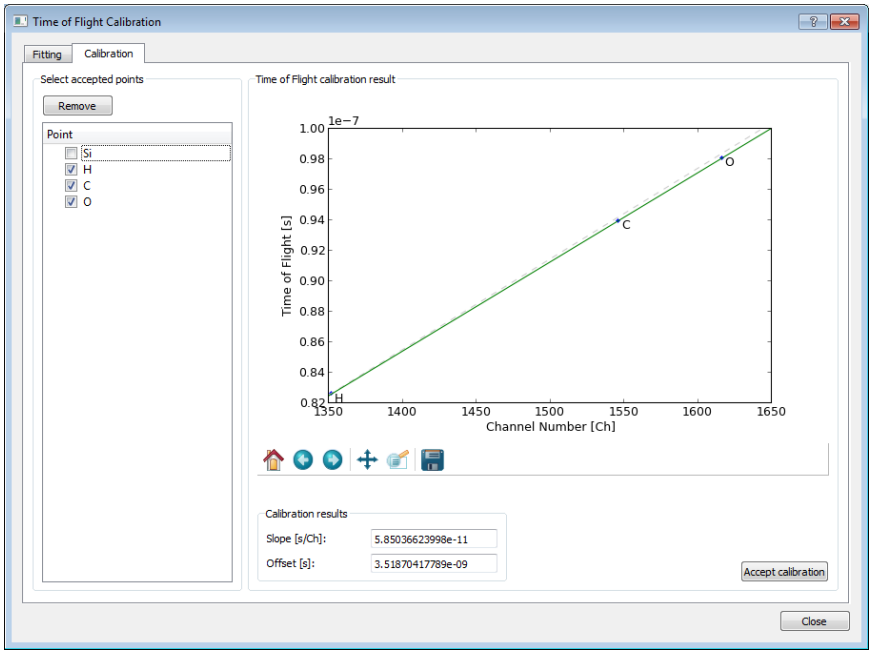

Figure 4.10: The *calibration* tab.

4.7 Generating Elemental Losses Histograms

To generate a histogram to analyze **elemental losses**, user can click the *Elemental losses* button on the right sidebar, or from toolbar menu *Tools* command *elemental losses*. Cut files have to exist to generate a elemental losses histogram.

First a dialog opens (Figure:4.11). This dialog contains the cut files that can be included in the histogram, a combobox where a reference cut file has to be chosen, a textbox where it must be defined how many splits cut files are split, and radio buttons for selecting the scale of the Y-axis. Once the user hits the *OK* button, a elemental losses histogram (Figure 4.12) is generated with the given parameter values.

To generate **split cut files**, the user can click the *Save splits* button below the his-

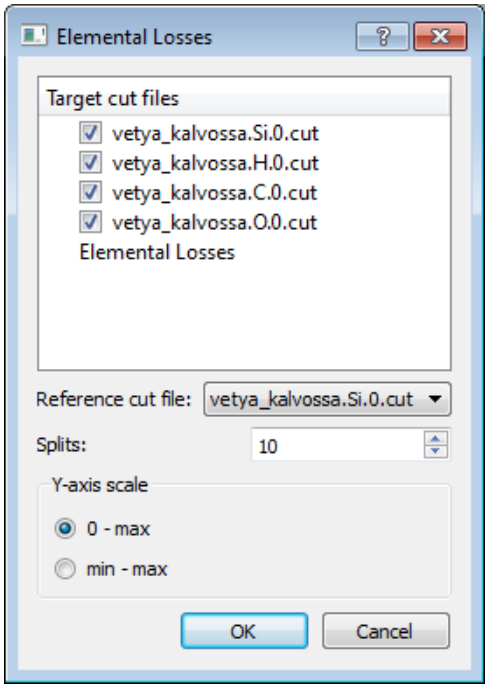

Figure 4.11: Elemental losses dialog.

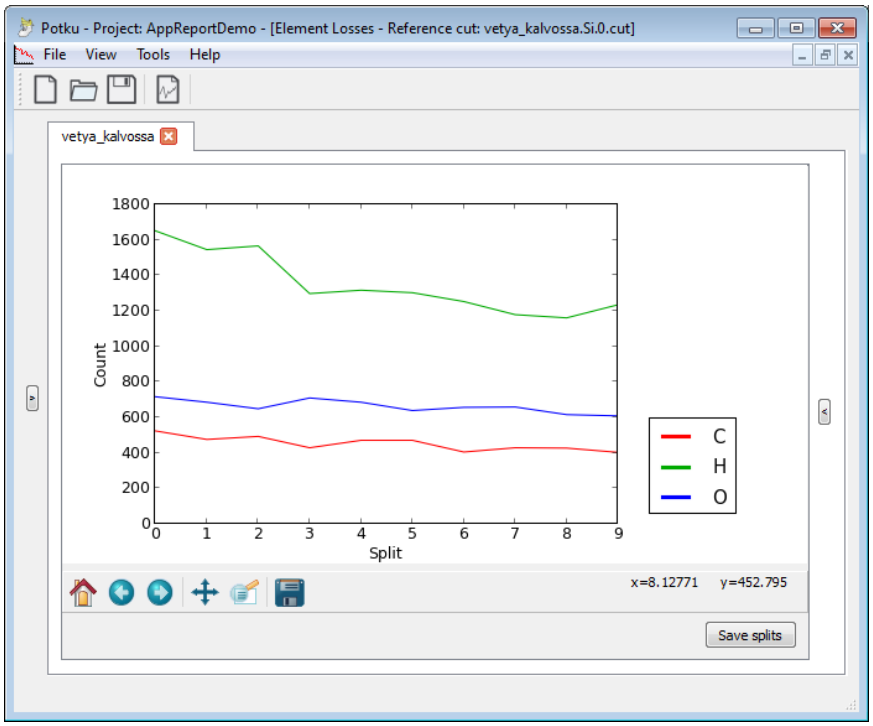

Figure 4.12: An elemental losses histogram

togram. A split file contains a certain portion of the cut file from which it is derived.

Split files can later be used like the cut files from which they are derived from.

4.8 Generating Energy Spectrum Histograms

To generate a histogram for analyzing **energy spectrums**, the user can click either the *Energy spectrum* button on the right sidebar or from menu *Tools* command *Create Energy Spectrum*. This opens a dialog (Figure 4.13) in which all the cut files that can be used are listed in a check list, as well as the option to set the bin width of axes. Clicking the *OK* button will generate the histogram with given settings.

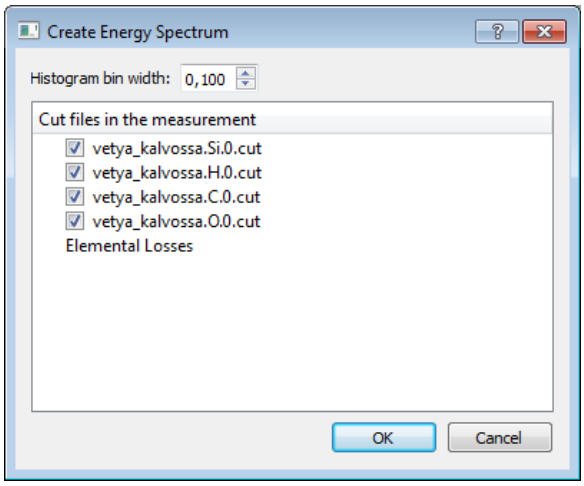

Figure 4.13: Energy spectrum dialog.

4.9 Generating and Analyzing Depth Profiles

To generate a **depth profile**, the user can click the *Depth profile* button on the right sidebar or from menu *Tools* command *Create Depth Profile*. A dialog opens (Figure 4.15) where the user can select the cut files to be included in the depth profile as well as the units used in the X-axis. Hitting *OK* will generate and show the depth profile with the given parameters (Figure 4.16).

The tools unique to the depth profile are as follows:

A. Limit setting – Set the limits for calculating percentages of elements.

B. Area selection – Determines the areas affected by tool C.

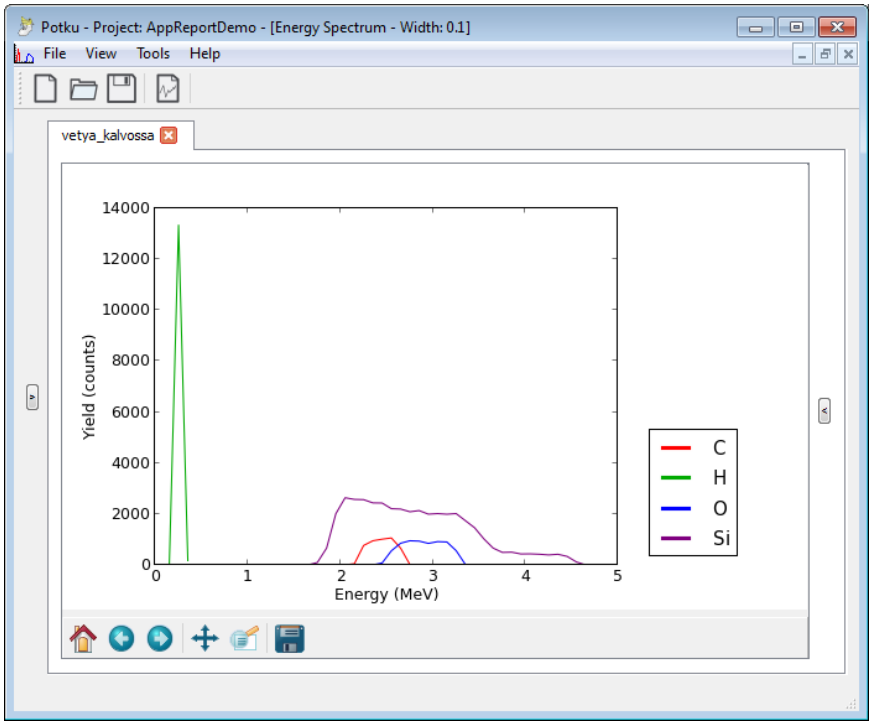

Figure 4.14: An energy spectrum histogram.

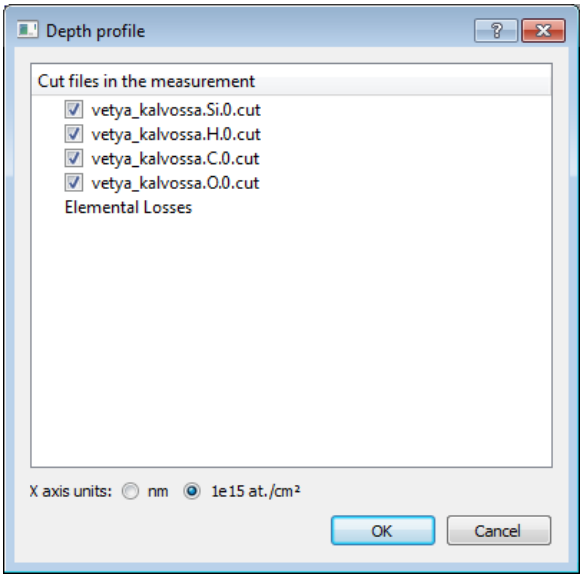

Figure 4.15: Depth profile dialog.

C. View – Toggles between absolute and relative plotting of the depth profile data.

Clicking the *view* button will toggle between an **absolute view** and **relative view**. In absolute view, data points are plotted according to their actual values. In relative

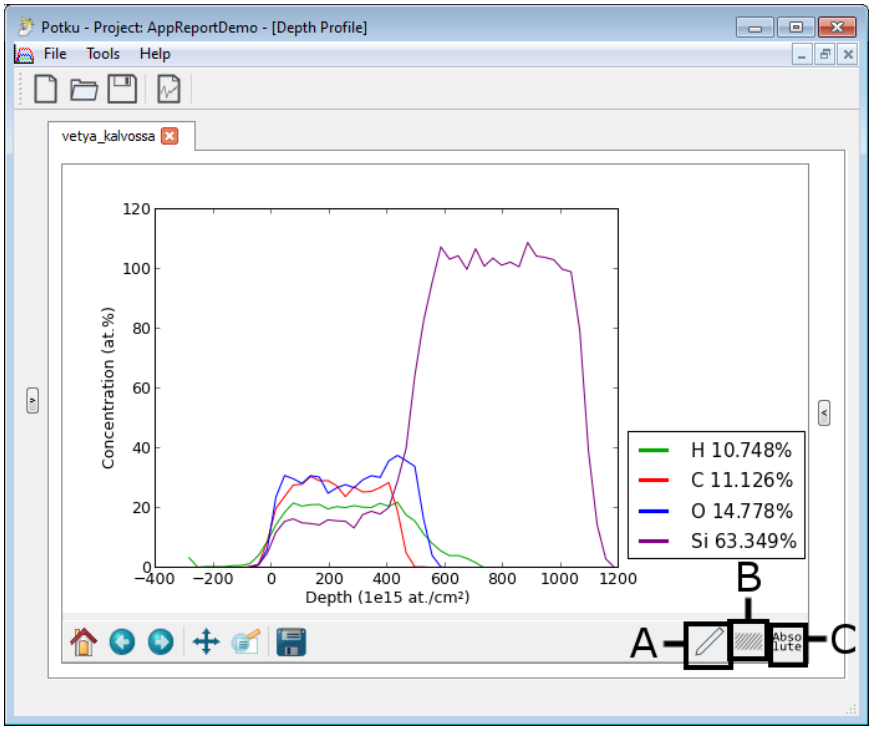

Figure 4.16: A depth profile

view, data points are scaled to the total value of all the selected elements, which ensures that the total amount of elements does not at any point exceed 100 percent. A depth profile in relative view is demonstrated in Figure 4.17.

With *limit setting* tool, the user can set the range in the histogram, from which the **percentages of elements** in the legend are calculated. The limits set will remain even after the tool is deselected. A depth profile with a custom range set is demonstrated in Figure 4.18.

The *limit setting* tool also enables the *area selection* button. This button has three modes: 1) The entire histogram is selected (default). 2) Only the area within the set limits is selected 3) Only the areas outside the set limits are selected. The view button will only affect the areas that are selected with this button. If for instance the button is set in mode 3, only the areas outside of the set range will be plotted relatively when the user hits the view button, while the area within the range will still be plotted with actual values. Usage of this button is demonstrated in image 4.19

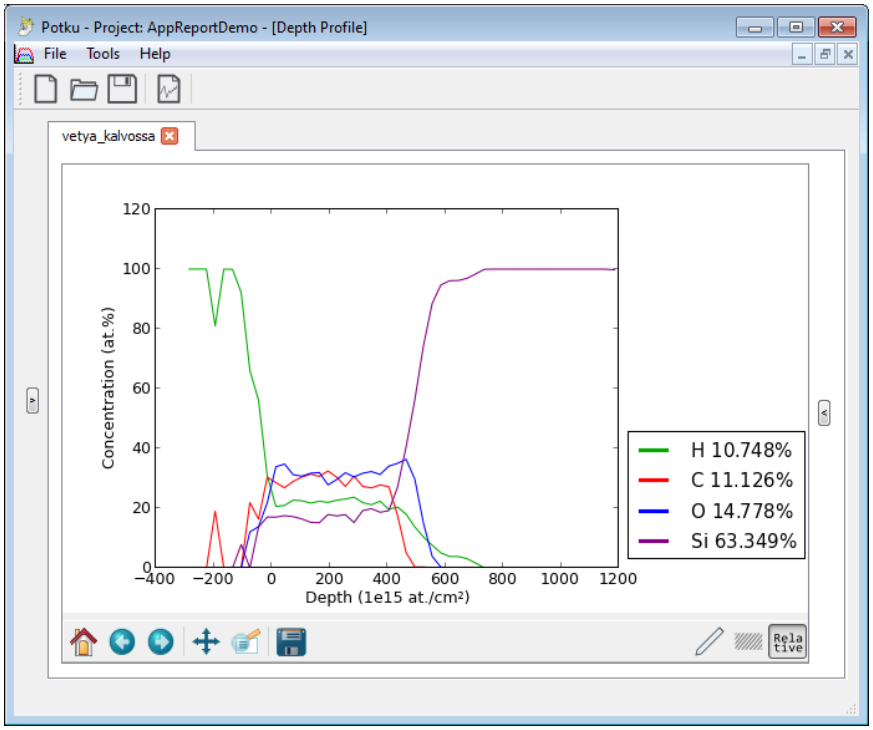

Figure 4.17: A depth profile in relative view

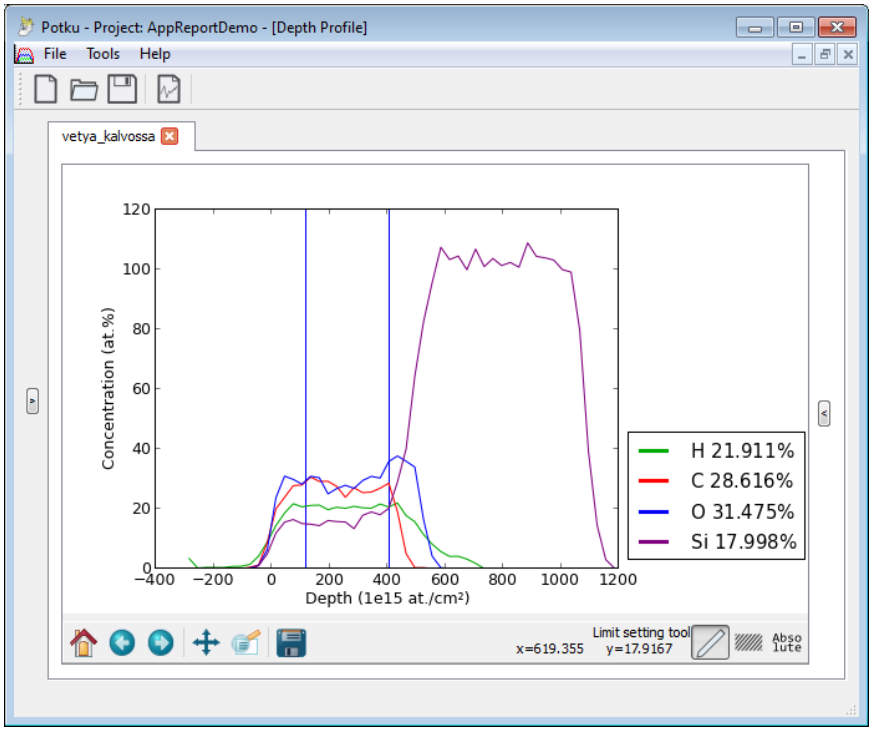

Figure 4.18: A depth profile with a custom range

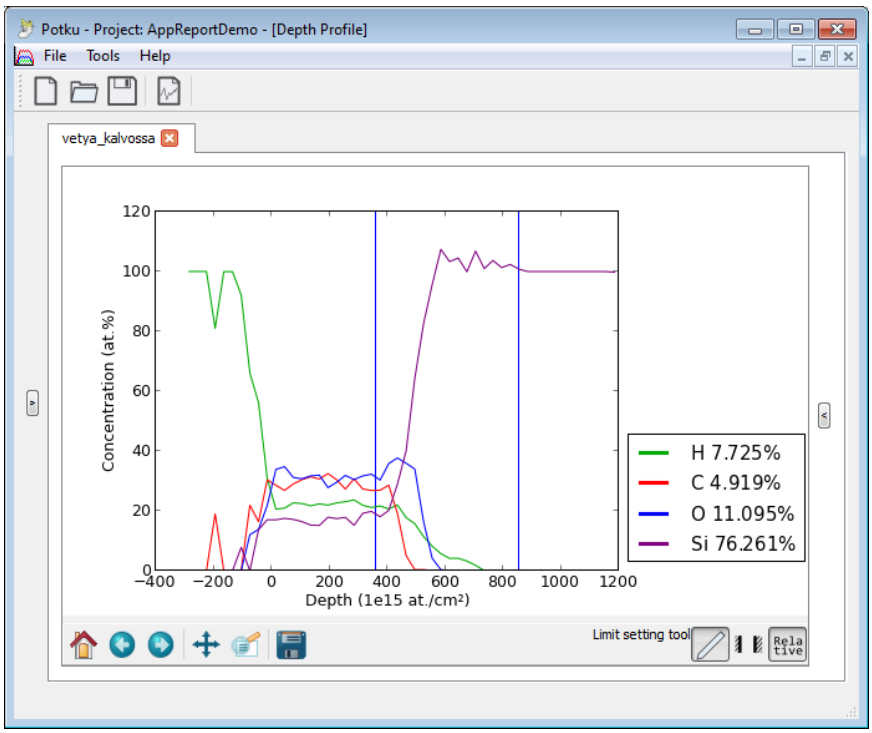

Figure 4.19: A depth profile with only areas outside of set range plotted relatively

5 Application Structure

The chapter descibes the different components in Potku application and their relations to each other. Potku is a installable workstation application. The application uses Python's own libraries, the GUI libraries of PyQt, the plotting libraries of MatPlotLib, the mathematical libraries of NumPy and SciPy, and external C components `tof_list`, `erd_depth` and `carbon_stopping`.

5.1 Components and Software

carbon_stopping is a program that calculates the stopping of elements against carbon. This component is necessary for performing ToF calibration.

erd_depth is a program that calculates the amount of elements at certain depths from the data generated by tof_list. This component is necessary for creating depth profiles. The usage of erd_depth is further documented in document [3].

MatPlotLib is a library for plotting different histograms. In Potku, the library was used for the plotting of the ToF-E histogram, elemental losses histogram, energy spectrum histogram, and ToF Fitting and Calibration histograms.

Numpy and **SciPy** mathematical libraries were used for some mathematical operations.

PyQT is a Python binding for the GUI library **Qt**. The entire Potku GUI was developed with PyQt, safe for histograms, which were plotted with MatPlotLib and placed inside Qt widgets.

Python libraries are the standard libraries that come with a Python installation, and were used for many purposes throughout the Potku source code. These libraries include `re` for regular expressions, `os.path` for managing directories, `logging` for logging, `math` for mathematical functions, `configparser` for

managing configurations, `subprocess` for invoking the external components, `platform` for determining what operating system the application is currently running on, and `csv` for importing data from file.

tof_list is a program that calculates the energy of a cut-file from time-of-flight. This component is necessary for creating energy spectrums and depth profiles. The usage of tof_ list is further documented in document [3].

5.2 Structure

The structure of the the application is roughly presented in Figure 5.1.

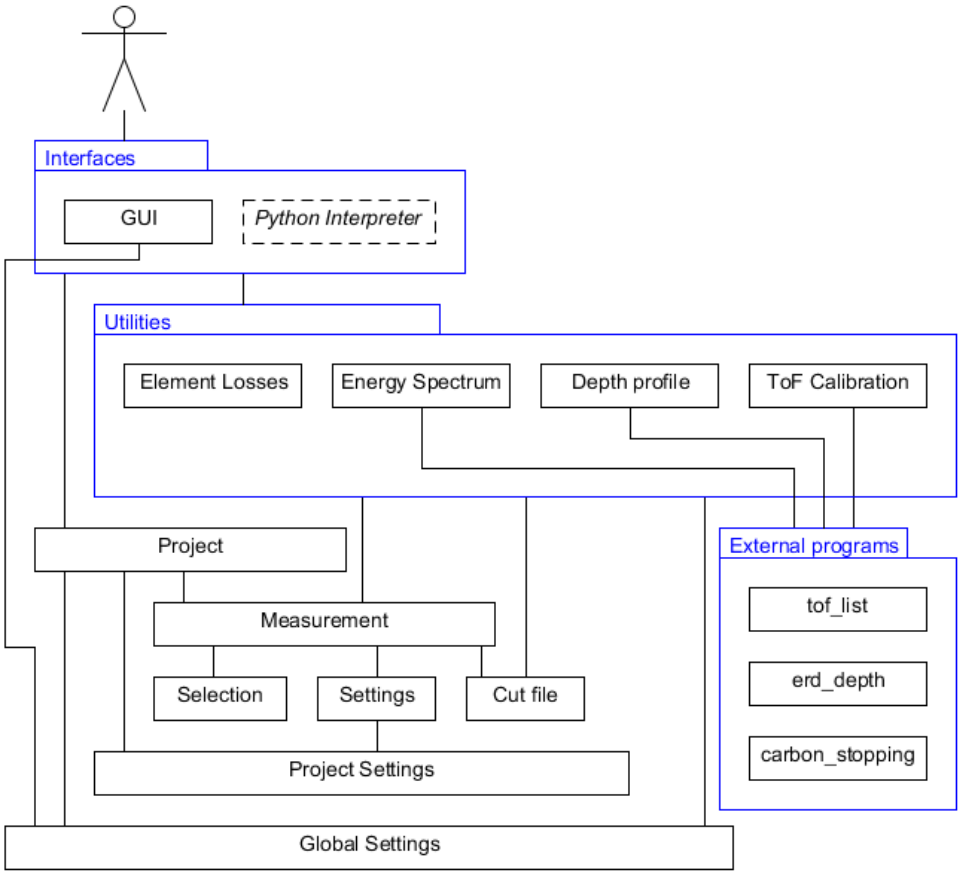

Figure 5.1: The structure of Potku.

Two interfaces were planned for Potku: graphical and Python interpreter -based. The interpreter interface was however not entirely implemented. All the utilities require information from *Measurement* and *cut files*. External programs are called by Energy spectrum, Depth profile and ToF calibration utilities.

More detailed documentatiton of the structure of Potku is documented in the Potku class documentation [**?**].

5.3 File and Data Formats

Potku uses proprietary data formats with the exception of the screenshots produced by MatPlotLib widgets, which produce picture files in standard formats such as png, pdf and svg.

A **cut file** is produced for each selected element from the ToF-E histogram, and it contains all the events of one of those selections. The naming convention of the file is `[1].[2][3].[4].cut`, where `[1]` is the name of the measurement file, `[2]` is the isotope of the selected element, `[3]` is the element symbol, and `[4]` is the count, if there are multiple selections of same element. For example, `vetya_kalvossa.12C.0 .cut`.

The content of a cut file consists of two parts: metadata and a two dimensional array. The length of the metadata is 10 lines and it contains parameters necessary for some functionalities. The metadata is of the following forma t:

```
Count: 4866
Type: ERD
Weight Factor: 1.0
Energy: 0
Detector Angle: 0
Scatter Element: None
Element losses: False
Split count: 1

ToF, Energy, Event number
```

The size of the content array is $3 \times n$, where n is the count of data points within the selection of the cut file. The content of the array is explained in table 5.1.

ToF	Energy	Event number
1663	1614	9
1620	1854	27
⋮	⋱	⋱
1580	1799	126514

Table 5.1: Example data content of a cut file.

Elemental Losses tool can save **split files** that very similar to cut files. In fact, a split file contains a $1/n$ portion of that files data content, where n is the amount of split files created from the cut file. The naming convention is a bit different: `[1].[2][3].[4].[5].cut`, where `[1]` is the name of the measurement, `[2]` is the isotope, `[3]` is the element, `[4]` is the count if there are multiple selections of the same element, and `[5]` is the count of the split. For example, `vetya_kalvossa.12C.0.3.cut`.

5.4 Integration of External C Components

Most of the physical calculation in the application has been left for `tof_list` and `erd_depth`. `tof_list` calculates energies for cut files based on the time-of-light. This data serves as input for `erd_depth` and energy spectrum. The format of the data generated by `tof_list` has been demonstrated in table 5.2

Det. A	Det. B	Energy	Elem.	Mass	Event type	Weight	Event num.
0.0	0.0	2.34176	6	12.0000	ERD	3.000	9
0.0	0.0	2.46176	6	12.0000	ERD	3.000	27
⋮	⋱	⋱	⋱	⋱	⋱	⋱	⋱
0.0	0.0	3.24498	14	27.9769	ERD	1.000	126568

Table 5.2: Example content of tof_list output.

External component `erd_depth` produces files from which **depth profiles** are calculated. The naming convention of these files is `depth.[1]`, where `[1]` is the symbol of the element. For example, `depth.Si`. In addition, erd_depth always creates a `depth.total` file. The content of these files is explained in table 5.3.

Depth (1e15 at./cm2)	Mystery	Depth (nm)	Atomic percentage	Mystery
-285.000	-1.914	-4.785	0.00000	0.00000e+00
-255.000	-1.864	-4.659	0.00000	0.00000e+00
⋮	⋱	⋱	⋱	⋱
1185.000	37.722	94.304	0.65255	2.45400e+28

Table 5.3: Example content of a depth file.

Late into the project, external C component carbon_stopping was also integrated into Potku. This component calculates stopping data of elements against carbon, and is necessary for the ToF calibration.

6 Programming Practices

Preliminary coding began soon after the project team had been introduced to the research team's current software, Finlandia. Source code was from the start written with Python 3.3. Any test prototypes that the project team wrote throughout the project were also written in Python 3.3, which made the integration of those codes into the master code easy.

One of the basic goals of the project was to keep the source code as readable as possible to enable further development of the application after the project. Through the project, there were two source code review events, in which the technical advisor Jonne Itkonen shared his notices and advice about the source code.

Internet resources were used in solving programming problems throughout the project, including the official Python 3.3 documentation [5].

6.1 Formatting, Naming and Commenting Practices

Potku followed the standard formatting and guidestyle of Python 3 [4]. Names of variables were written entirely lowercase with underscores symbolising spaces. Names of methods followed the same naming convention. An exception to these conventions were the variables automatically generated by the PyQt, since PyQt follows the C style CamelCase. All classes were named with CamelCase as well. Names of variables and objects were kept as self-explanatory as possible. For any modifications on external C codes, the style standards of C were followed.

Each module contained function or method specific commentary, and possibly line or block -specific explanations to help understand the purpose of the said code region. Function/method specific commentaries include the purpose of the function or method, explanation of the arguments it takes, and a explanation of what it returns.

It was agreed that an indentation would always be four whitespaces. Uniform indentation is especially important in Python, since grouping in Python is often determined by indentation. An uneven indentation would cause problems when trying to run the code. Any function calls that had overly long parameter lists were spread across multiple lines to spare horizontal space and ease readability.

All the variable names, function names and class names were written in English, as was all the documentation. Like all the documentation of the project, the author of each source code file was credited as the entire project team in alphabetical order.

Each source code file was to use a three step version numbering practice similar to the other documentation of the project. This however did not happen, and instead each file is version 1.0, which was automatically generated for each new source file by Eclipse.

6.2 Source Code Example

The following source code example is taken from file `Modules/Element.py`. The example demonstrates the naming, commenting and formatting practices followed, as well as unit testing.

```
# coding=utf-8
'''
Created on 10.4.2013

'''
__author__ = "Jarkko Aalto \n Timo Konu \n Samuli Kärkkäinen
             \n Samuli Rahkonen \n Miika Raunio"
__versio__ = "1.0"

import re

class Element:
    def __init__(self, element, isotope=None):
        '''Inits element class.

        >>> test_a = Element("1H")
        >>> test_b = Element("H")
        >>> test_c = Element("H", 1)
        >>> test_d = Element("Ca", 40)
        >>> test_e = Element("")
        >>> test_f = Element("H1")
```

```
        >>> test_a.to_string()
        '1H'
        >>> test_b.to_string()
        'H'
        >>> test_c.to_string()
        '1H'
        >>> test_d.to_string()
        '40Ca'
        >>> test_f.to_string() # Suppose we ignore numbers or
                                 whatever after element.
        'H'
        '''
        if element:
            m = re.match("(?P<isotope>[0-9]{0,2})
                         (?P<element>[a-zA-Z]{1,2})",
                         element.strip())
            if m:
                self.name = m.group("element")
                if isotope:
                    self.isotope = Isotope(isotope)
                else:
                    self.isotope = Isotope(m.group("isotope"))
            else:
                raise ValueError("Incorrect string given.")
        else:
            self.name = element
            self.isotope = Isotope(isotope)


    def to_string(self):
        '''Transform element into string.

        Return:
            Returns element and its isotope in string format.
        '''
        return "{0}{1}".format(self.isotope.to_string(),
                               self.name)
```

```
def get_element_and_isotope(self):
    '''Get Element's name and isotope.

    Return:
        Returns element's name (string) and its isotope
        (class object).
    '''
    return self.name, self.isotope
```

6.3 Grouping Practices

Since it was intended that the application could be used from a command line, it was intended that functional logic and UI of the application would be in separate classes. The classes are spread to packages according to their purpose: `Dialogs` contains the classes that act as the dialogs of the application. `Modules` contains classes that form objects. `Widgets` contains classes that form widgets to be used with PyQt.

Since so many tools of the application use MatPlotLib, the MatPlotLibWidget was split into several separate child classes to maintain a bearable line count.

6.4 Development Platform

For developing the Python source code, the project team used Eclipse with the PyDev extension. Modification of external C source codes was done with each project member's text editor of choice (usually Notepad++ on Windows, Nano on Linux and Mac) and compiling was done with GCC.

For development, the project team had four Windows workstations and one Linux workstation. Development was mostly done on the Windows workstations and the Linux platform was mostly for system testing. The project team received late into the project a Mac workstation as well, but this too was used mostly for system testing purposes.

Source code is encoded in UTF-8. Version control was handled with YouSource, which uses Git.

7 Testing Practices and Results

The testing practices were planned originally in the project plan [7]. This chapter describes the realization of these practices and the results of testing.

7.1 Unit and Integration Testing Practices

The project team members programmed a simple prototype of a new feature, before writing a class to integrate in the master. Unit testing was done to functions and methods that were sensible to unit test. Not all functions and methods that could be unit tested however were not unit tested in the end. Since most of the calculations required by Potku are done in external C programs, so these aren't unit tested by the project team either. Each new class added into the project was integration tested as they were being written. Once the integration was satisfactory, the new module could was pushed into the Potku repository.

7.2 System Testing Practices

One of the requirements of Potku was that it would work on Windows, Linux, and Mac operating systems. Thus it had to be system tested on all these platforms.

For most of the programming, the project team used Windows, and Windows testing could be done as source code was written. Once Potku worked satisfactorily on Windows, it could be tested on Linux and Mac, platforms which the application was not programmed in. Mac system testing came very late in the project, since there was trouble in acquiring a Mac computer for testing purposes and installing the necessary software on it. A suitable Mac was prepared eventually, and the project team could carry out the Mac system testing

7.3 Usability Testing Practices

While usability testing wasn't formally planned in the project plan, it inevitably happened as Potku was developed. The usability of Potku was tested by the project

team, the instructors, and the customer. In addition, usability expert Meeri Mäntylä commented on the usability during the usability day that was held on April 10th.

It was intended in the project plan [7] that the project team would produce two two public versions of Potku for the organization to test. This however did not become reality. Instead, new or improved functional source code was uploaded to Potku's repository's master branch nearly on daily basis, from where anyone in the project organization could download the latest version of the application.

The usability and features were discussed in the weekly/bi-weekly project meetings, during which the project team would demonstrate the new implementations developed since the previous meeting on a laptop in which the latest version was installed. In addition to these demonstrations, the customer did test the software on their own too.

7.4 Testing Results

8 Realization of Objectives

This chapter describes the objectives set to Potku, and how well they were met. Major functionalities that were not implemented at all were the command prompt user interface and the reporting tools.

8.1 Realization of Requirements

In the Potku requirements specification [1], 83 requirements were specified. Each requirement was given a priority depending on how essential it is to the customer. These priorities were 1 = mandatory, 2 = important, 3 = possible, 4 = idea, and 5 = will not be implemented. Priority 1 requirements were essential to the program and took priority on the project, while priority 4 and 5 were never intended to be implemented during the project.

Of the 83 requirements defined in the requirements specification, 50 have been been implemented. Of the priority 1 requirements 42 of 45 were met. Of priority 2 requirements 4 of 16 were met. Of priority 3 requirements 4 of 17 were met. As intended, none of priority 4 or 5 requirements have been implemented. Unimplemented priority 1 requirements are the following:

1.4 An element uses an efficiency file if such exists: Currently the handling of efficiency files is handled by tof_list. It however currently only accepts efficiency file for 1H, if it in the same directory as tof_list. Modifying the C program would have been too time consuming for the project team, so this was not implemented.

1.7 Depth profile's stopping model can be chosen: It was intended that zbl96 stopping model would be only temporarily used and it would later be replaced with another. However, due to time constraints a replacement stopping model could not be integrated into Potku.

1.11 Python interpreter interface: Not fully implemented due to time constraints.

8.2 Unsatisfactory Solutions in the Implementation

While selected solutions are functional, some of them aren't done in the best possible way. During further development, refactoring these solutions should be considered.

Getting output from the C programs should be done more gracefully than simply calling them with subprocess.call. This way is vulnerable to any system policies on running executables.

DepthFiles.py should be rewritten as a proper object or only as a collection of functions necessary for depth profiles. As is DepthFiles.py is a strange mixture of a very small object used for calling the external C programs to generate depth files, and a collection of functions related to operating depth profile data.

8.3 Challenges During the Implementation

Many of the issues faced during the project were related to **integrating the external C programs into Potku**. These programs were originally written in Linux, and compiling them on Windows was not entirely without problems. Some modifications had to be made to enable the same C source codes to be compiled with the same Makefiles on Windows, Linux and Mac.

Another issue was the output format of `erd_depth`. `erd_depth` outputs a file for each element it analyzes, but it does not save the information about the isotope of the file in any way. This is not an issue, as long as long as only one isotope of an element is being analyzed at a time, but multiple isotopes of the same element would cause problems when generating depth profiles, that do require `erd_depth`'s output.

Another issue came late in the project, when ToF calibration was being implemented. ToF calibration required data from an external C program `carbon_stopping`. Compiling this program was again not entirely without problems on Windows and Linux.

Since no-one in the project team had extensive knowledge of Python, or the selected libraries, there were occasional minor problems in understanding them. Although these were often easily remedied by the extensive documentation of the language, and the instructions of the technical instructor.

The subject (accelerator-based material physics) was also very alien to the project team. This issue was however well-remedied with close contact to the customer as well as the possibility to draw inspiration from an existing application.

9 Guide for Future Developers

The customers have decided that Potku will go through further development after the project. In this chapter are several tips for the future developers of the software, so that the most critical issues may be attended to as soon as possible. For more detailed list of open issues, please see the Requirements Specification [1].

9.1 Essential Bugs

There are some known bugs in Potku that hinder it's usage. The following bugs should be prioritized when the application enters further development.

1. Depth profiles is prone to failing, if the project settings are not properly configured. It fails because erd_depth will crash, if it receives invalid values as parameters.
2. Depth profiles will fail, if there are multiple selections of the same element. This is because the output files of erd_depth lose the information of what cut file they were generated from.
3. When measurements are deleted in a project, their data is not properly erased from memory. This is a memory leak that could hog extensive amounts of memory in prolonged use of Potku. Some measures have been taken against this issue, but it remains unsolved.
4. When loading a project, Potku does not remember what previous graphs were generated, as it does not save information about any energy spectrums, elemental losses or depth profiles that were generated in a project. All the graphs have to be regenerated each time.

Known bugs of lesser priority are listed in section 7.4.

9.2 Improvements of Existing Features

Although the source codes of the external C programs should compile on any Mac, Linux and Windows with MinGW, it might be better to include compiled versions of the programs for all three operating systems. As is, a compiled executable of each external C program is included, but compiled programs for Mac and Linux are not.

9.3 Further Development Ideas

The usage of logger is is not widely implemented in Potku. In further development the usage could be extended to modules where it is not yet properly implemented.

10 Summary

Potku project developed a software for the analyzation of data received from a recoil spectrometer. The customer of the project was the research team of accelerator-based material physics under University of Jyväskylä's Department of physics. Not all the mandatory requirements were met due to time constraints, and all requirements not implemented were left to for further development outside the project.

The user interface consists of several tools used for the analysis of data received from the recoil spectrometer. The major tools are ToF-E histogram, ToF calibration, elemental losses and depth profile. The software is suitable for limited use, but does require further development.

Potku application was developed using the PyQt GUI library, MatPlotLib plotting library, and numpy and scipy mathematical libraries. Some of the application's icons are from from The Reinhardt Icon Set [6] licensed under LGPL.

Potku application relies on the functionality of the C programs delivered by the customers. For correct operation of Potku, it is recommended that these programs are compiled when the application is installed to a new platform.

The application was once reviewed by an usability expert and the source code was reviewed twice by the technical instructor of the project. System was tested with ad hoc, unit, integration and system testing. Further development of the software is planned to start after the project.

11 References

[1] Aalto Jarkko, Konu Timo, Kärkkäinen Samuli, Rahkonen Samuli and Raunio Miika, "Potku Project Software Requirement Specification", University of Jyväskylä, Department of Mathematical Information Technology, 2013

[2] Iso-Ahola Pekka, Perttola Jussi and Tuovinen Tommi, "Kuvatus Project Application Report" University of Jyväskylä, Department of Mathematical Information Technology, 2012

[3] Sajavaara Timo, "Analysis with TOF-ERDA at IMEC", 2004

[4] van Rossum Guido and Warsaw Barry, "PEP 8 – Style Guide for Python Code", available at `http://www.python.org/dev/peps/pep-0008/`, cited at 3.5.2013

[5] Python Software Foundation, "Python 3.3.1 documentation", available at `http://docs.python.org/3/`, cited at 3.5.2013

[6] Jensen Dan Leinir Tuthra, "The Reinhardt Icon Set", available at `http://leinir.dk/leinir/content/en/Reinhardt+Icon+Set`, cited at 8.5.2013

[7] Aalto Jarkko, Konu Timo, Kärkkäinen Samuli, Rahkonen Samuli and Raunio Miika, "Potku-sovellusprojekti Projektisuunnitelma", University of Jyväskylä, Department of Mathematical Information Technology, 2013

[8] Aalto Jarkko, Konu Timo, Kärkkäinen Samuli, Rahkonen Samuli and Raunio Miika, "Potku-sovellusprojekti Projektiraportti", University of Jyväskylä, Department of Mathematical Information Technology, 2013

[9] Aalto Jarkko, Konu Timo, Kärkkäinen Samuli, Rahkonen Samuli and Raunio Miika, "Potku Testing Report", University of Jyväskylä, Department of Mathematical Information Technology, 2013

[10] Aalto Jarkko, Konu Timo, Kärkkäinen Samuli, Rahkonen Samuli and Raunio Miika, "Potku Class Documentation", University of Jyväskylä, Department of Mathematical Information Technology, 2013