Potku Project

Jarkko Aalto
Timo Konu
Samuli Kärkkäinen
Samuli Rahkonen
Miika Raunio

Application Report

Public
Version 1.0.0
29.5.2013

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: 43

Abstract:
Potku project developed an user interface for a software used in analyzing data received from a recoil 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. The document describes the backgrounds of the application as well as presents the user interface and the general structure of the application. The programming and testing practices are also presented. The realization of the functional requirements and other agreed upon objectives is described, as well as advice for future development.
Keywords: Application structure, background, 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@jyu.fi	–

Instructors:

Itkonen Jonne	jonne.itkonen@jyu.fi	050-4432381
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. User interface and testing practices chapters were improved.	MR
0.1.2	13.5.2013	Small fixes across the document based on instructor Santanen's feedback were made.	MR
0.2.0	15.5.2013	Interface, application structure, programming practices and testing chapters were improved.	MR
0.2.1	16.5.2013	Small fixes across the document based on Kärkkäinen's feedback were made.	MR
0.2.2	17.5.2013	Proofreading and fixes across the document were made.	MR
0.2.3	20.5.2013	Fixes across the document were made according to Santanen's feedback.	MR
0.3.0	21.5.2013	Testing results were written.	MR
0.4.0	26.5.2013	Fixes across the document were made according to the feedback of Santanen and Julin.	MR
0.5.0	28.5.2013	Small fixes across the document were made according to the feedback of Santanen. Added a list of partly implemented requirements to Section 8.1.	MR
1.0.0	29.5.2013	Typos were corrected across the document.	MR

Contents

1 Introduction

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

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 calculations necessary for some tools of the application are 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. The user interface was programmed using Python 3.3 programming language. Potku contains external analyzation programs programmed in C, that were developed by the customer.

Documents were written during the project to describe the developed software and the project. The purpose of this document is to report the resulting software application, and the realization of plans and requirements set for the application. The requirements specification [1] contains the full list of requirements set for the application. How the project was carried out is described in the project report [8]. The class documentation is described in [10]. 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 and the goals of the project. In Chapter 4 the user interface is presented and the essential functionality of the application is demonstrated. The general structure of the application is described in Chapter 5. The used programming practices and a code sample are presented in Chapter 6. Chapter 7 reports the testing that was carried out and the results. The realization of objectives set for the application is described in Chapter 8. Finally, Chapter 9 contains recommendations 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 at 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 visualizes 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 measurements taken from a collection of samples. Each of the samples may have been manufactured differently from each other.

Recoil spectrometer is a research device used by the research team. It 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. For example, it could be 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 created in different circumstances, in different temperatures for example. 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 the 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.

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

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.

PyQt is a Python binding for the graphical user interface toolkit Qt. PyQt was used for developing the graphical user interface of the application.

Python is the primary 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 at 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. The beam 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 samples are usually provided by a customer, who wants to find out the composition of some material.

The setup of the recoil spectrometer is demonstrated in Figure 3.1. In the figure *Incident MeV heavy ion* is the ion beam launched from the accelerator. It will collide with *Sample* in angle α. Particles will recoil from the *Sample* in angle ϕ. The recoiled particles pass through carbon foil detectors T_1 and T_2, that will measure the time-of-flight of the particles. Finally, the *E detector* will measure the energy of the particles.

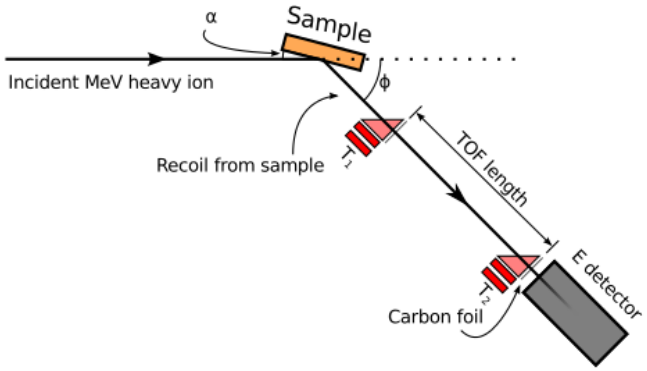

Figure 3.1: Basic setup of the recoil spectrometer.

There are several utilities for 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 utilities have their own set of tools which can be used in the analysis process.

The research team currently has an 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 known bugs and shortcomings. The research team decided that producing a new application from the start would be a better solution than continuing the development of Finlandia's source code. The research team contacted the Department of Mathematical Information Technology to develop an 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 is that the code is poorly documented and difficult to develop further. Basic functionalities however work and there aren't major drawbacks with the user interface either. So Finlandia could be used for reference of correct functionality, and inspiration could be drawn from it's GUI for the user interface of 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 of Potku 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 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.

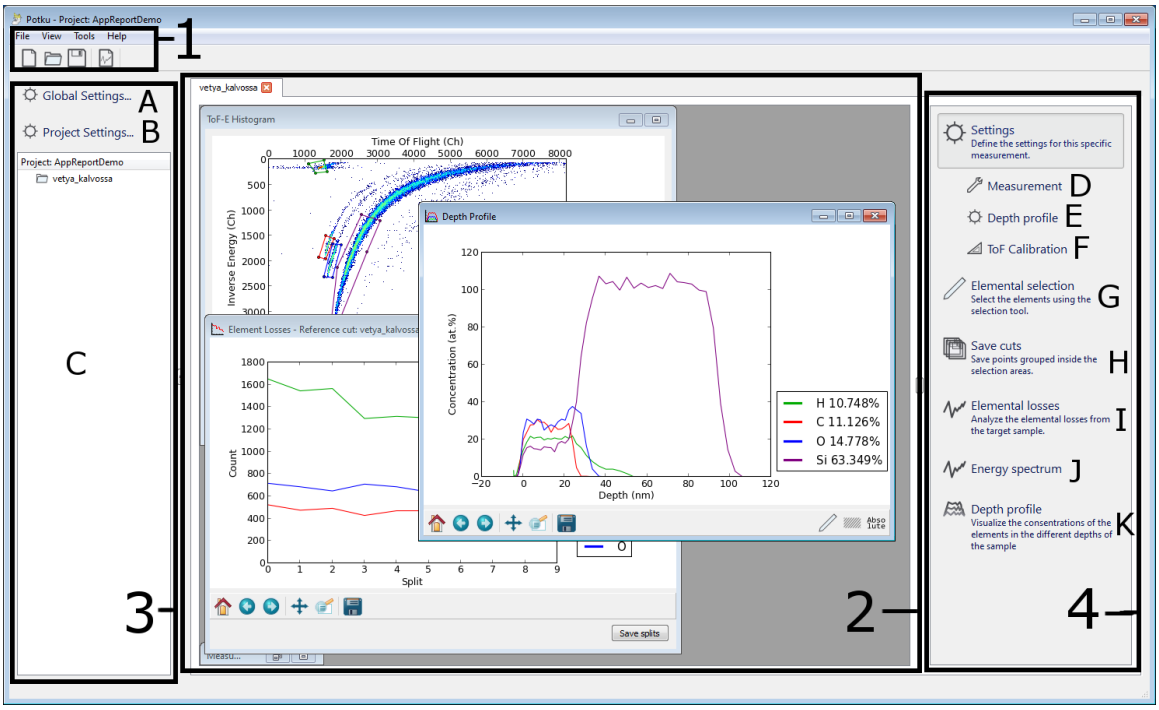

Figure 4.1: Potku main window.

There are four distinct components:

1. *Menus and top toolbar* contain all the functionalities 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. It can be hidden with the small button next to it.

4. *Right sidebar* contains buttons for all the major analysis tools. It is enabled only when a measurement has been loaded into the project. It 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 loaded in Potku.

B. *Project settings* contains settings that will only affect the current project. These settings are 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. The settings defined here can be used to override the corresponding parts in the project settings.

E. *Depth profile settings* contains settings regarding the depth profile. The settings defined here can be used to override the corresponding parts in the project settings.

F. *ToF calibration settings* contains settings regarding the time-of-flight calibration. The settings defined here can be used to override the corresponding parts in the 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 shown in Figure 4.2 is significantly less busy than in Figure 4.1. The user essentially has three options:

1. Defining global settings.
2. Loading an existing project.
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. Option 3 is demonstrated in Section 4.3.

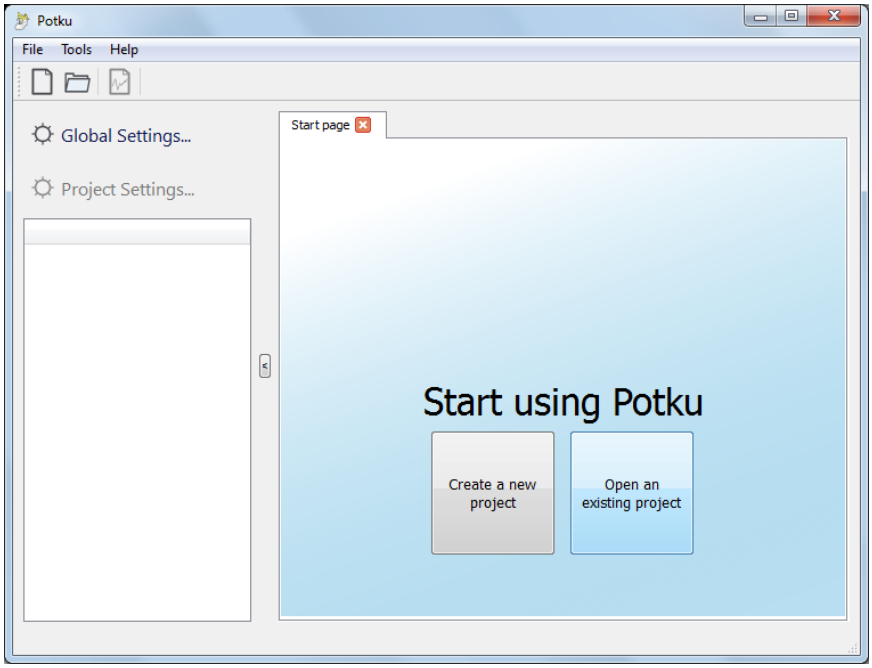

Figure 4.2: The main window with nothing loaded yet.

4.3 Creating a Project

To create a new project the user can click the big button in the workspace (see Figure 4.2) or from the toolbar menu *File* the command *New Project*. A dialog in Figure 4.3

opens 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 *Project Settings* button has been enabled, and the buttons in the workspace have been replaced by a *Create a new measurement* button.

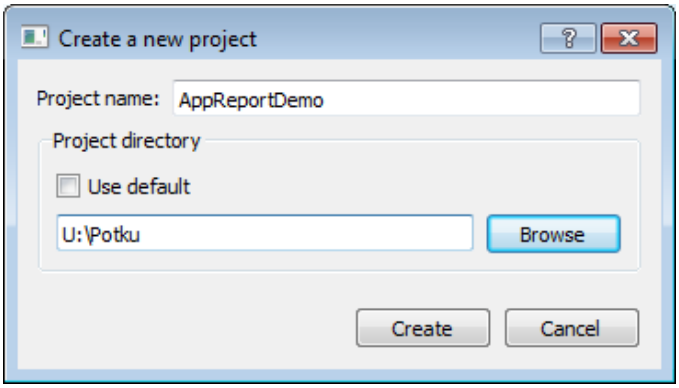

Figure 4.3: Creating a new project dialog.

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* the 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 as shown in Figure 4.4.

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

A. *Reset to 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* the graph.

D. *Zoom* in on the graph.

E. *Save* an image file of the graph.

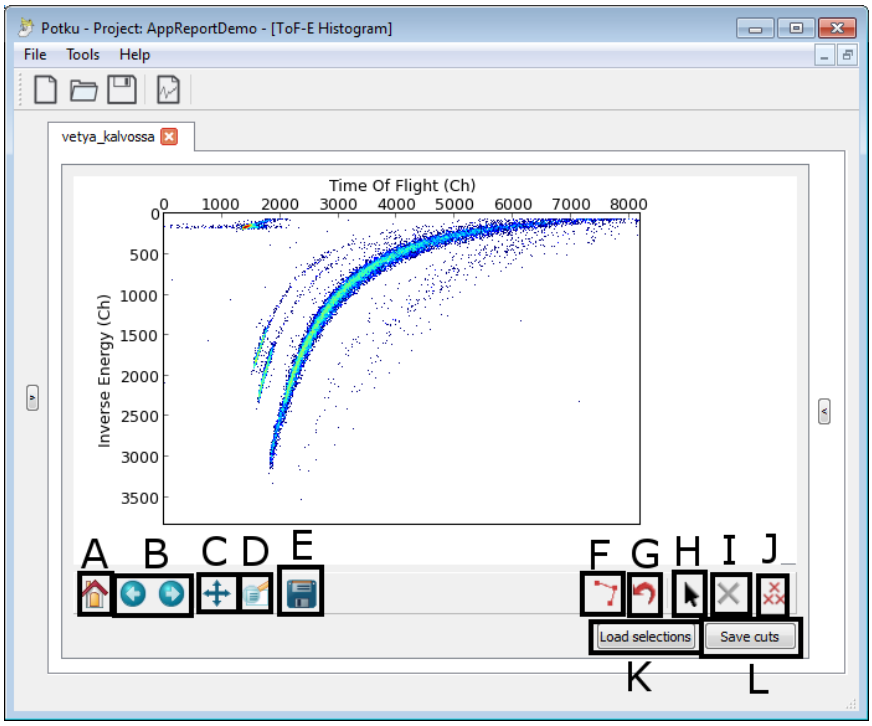

Figure 4.4: A loaded measurement with no selections.

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* selects an element selection done with tool F.

I. *Delete selected selection* deletes an selection selected with tool H.

J. *Delete all selections* deletes 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, the user has to equip the *Select element area* tool either from the right sidebar (see Figure 4.1) or from histogram window toolbar (F). With the tool the user will set selection nodes with the left mouse button. A

selection is completed 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 as shown in 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.

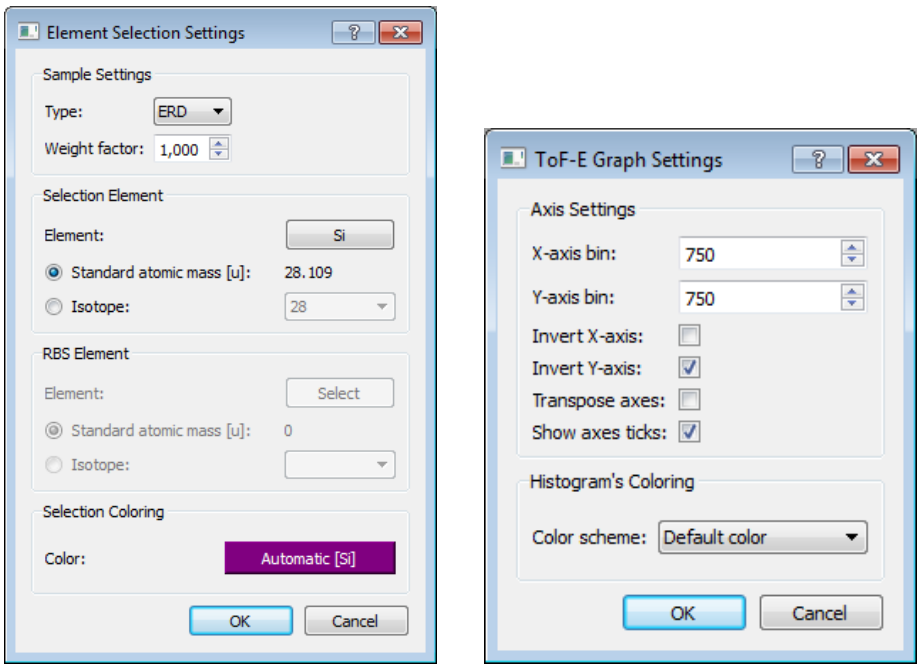

Figure 4.5: Selection settings and graph settings dialogs.

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

During the selection of a cut area, the user can **cancel previous nodes** by clicking *Undo last point* button (G). The user can select completed selections with the *Select element selection* tool (H) and re-edit their settings or delete them either with the *Delete selected selection* (I) and *Delete all selections* (J) buttons. Additionally, the user can load existing selections from a file via the *Load selections* button (K). Once the user is content with the selections, he/she can save the selections into cut files with the *Save cuts* button (L). Cut files are used in further analysis. A histogram with complete selections is shown in Figure 4.6.

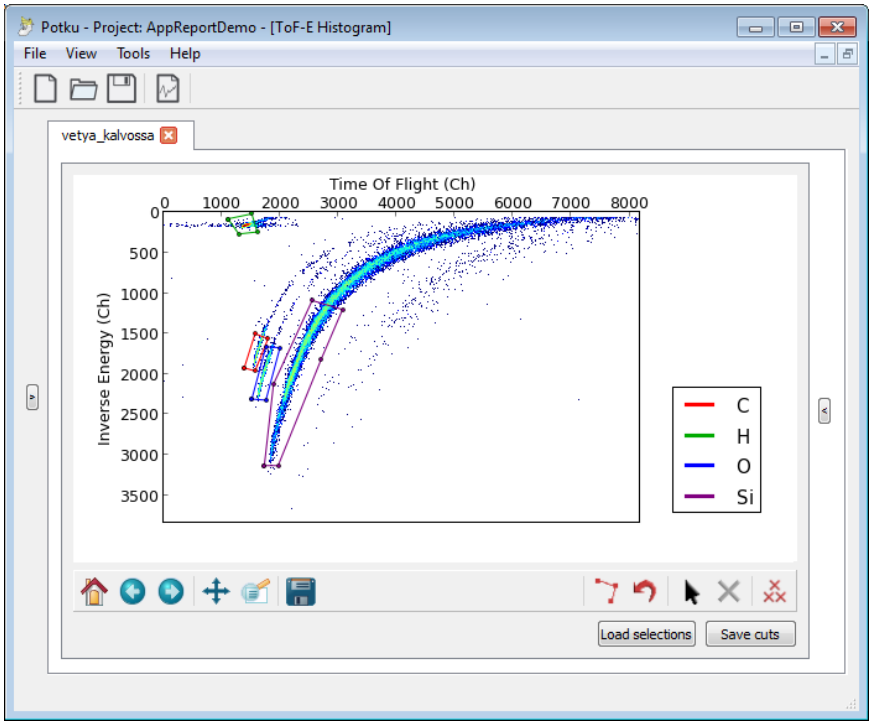

Figure 4.6: A loaded measurement with selection of four elements.

4.5 Defining Settings

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

Global settings contains the parameters for 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* in Figure 4.6. *Global settings* can be accessed from the left sidebar. The settings window is shown in Figure 4.7.

Project settings contains several parameters regarding the used equipment during the experiment. *Project settings* can be accessed from the left sidebar (see Figure 4.1). This settings window is shown in Figure 4.8.

Measurement settings essentially contains the same settings as project settings. The purpose of measurement settings is to override parts of the project settings in a single measurement, while leaving the project settings intact for the rest of the measurements. Measurement settings can be accessed from the right sidebar (see Figure 4.1).

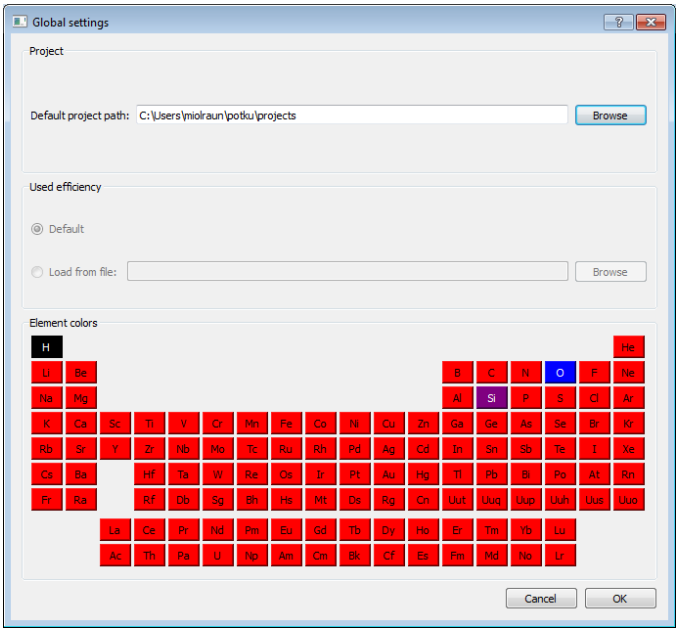

Figure 4.7: Global settings dialog.

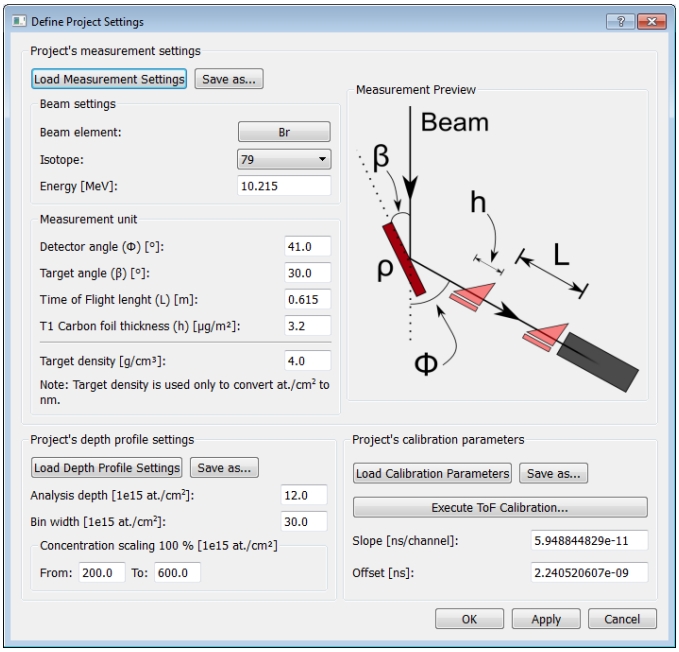

Figure 4.8: Project settings dialog.

4.6 Time-of-Flight Calibration

The user can perform **time-of-flight calibration** from the *Calculate ToF Calibration* button in the *Project settings* in Figure 4.8. This will open a dialog with two tabs: *Fitting* and *Calibration*.

First, the user has to perform **curve fitting** on the *Fitting* tab in Figure 4.9. Potku will attempt to automatically calculate the **front edge** of each cut, but the user can also manually set the edge with the *pen tool*. The curve fitting graph can also be replotted with the *bin width*, which can result in a more sensible automatic fit. The numeric values of the fit can also be set by the user in the *Channel number* and *Time of Flight* textboxes. When the user finds a satisfactory fit, he/she clicks *Accept point* button. Curve fitting is done for each cut individually. Other buttons of the dialog are described in Section 4.4.

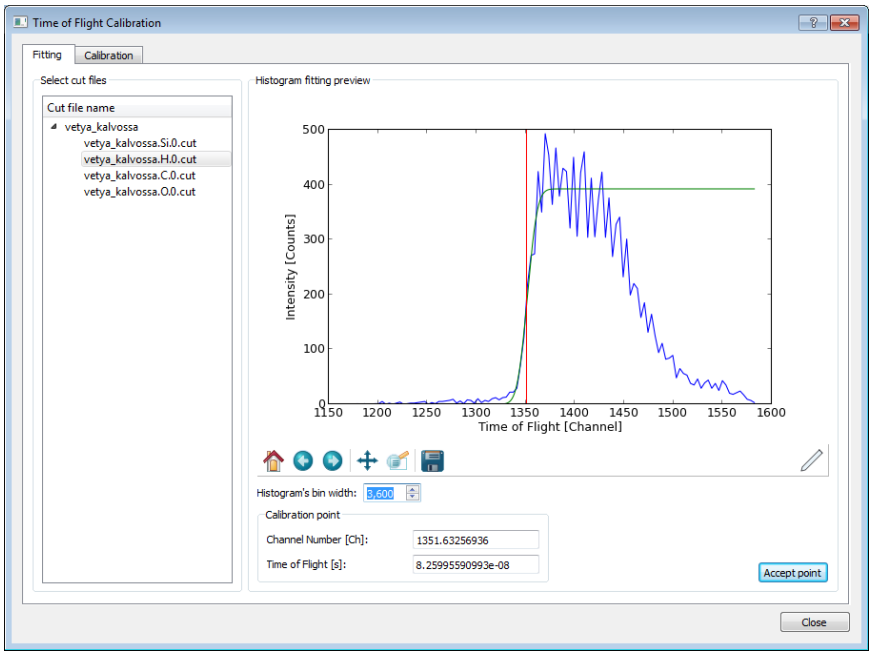

Figure 4.9: Fitting tab.

Once the user has accepted a point from each cut to be used in calibration, he/she can switch to the *calibration* tab in 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, so 'bad' points can be removed from the calibration in the element list on the left. In the demonstration, silicon did not quite fit. The numeric results of the linear fitting can be inserted directly in the *Slope* and *Offset* textboxes.

Once the user has found a satisfactory fit, he/she can click the *Accept calibration* button, which will return the user back to the project settings dialog in Figure 4.8. Other buttons of the dialog are described in Section 4.4.

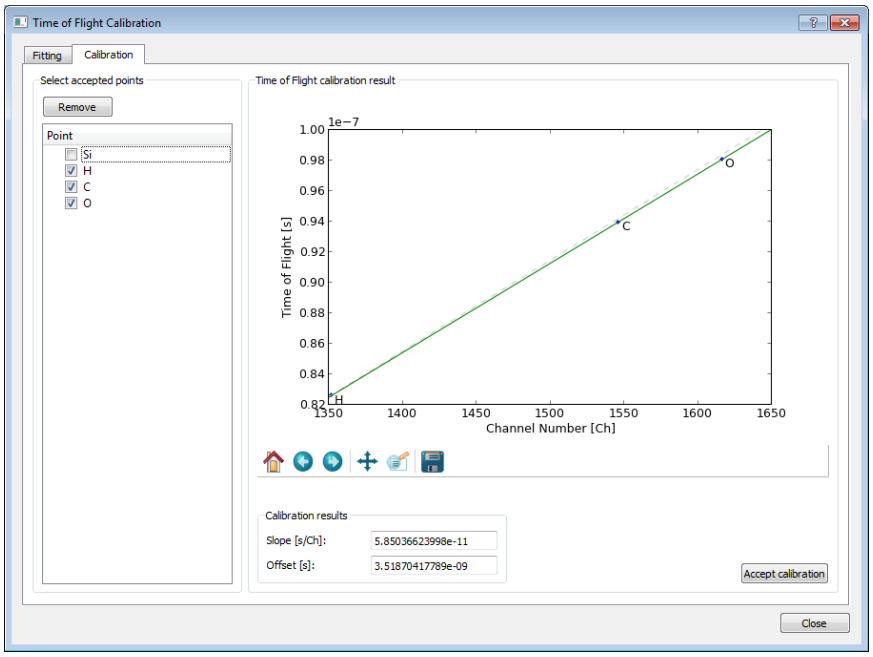

Figure 4.10: Calibration tab.

4.7 Generating an Elemental Losses Histogram

To generate a histogram to analyze **elemental losses**, the user can click the *Elemental losses* button on the right sidebar (see Figure 4.1), or from the toolbar menu *Tools* the command *Elemental losses*. Cut files have to exist to generate a elemental losses histogram. Split files can be used also, but in this demonstration, they are not generated yet.

The dialog in Figure 4.11 opens. The 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 the cut files are split into, and radio buttons for selecting the scale of the Y-axis. Once the user clicks *OK* button, an elemental losses histogram in Figure 4.12 is generated with the given parameter values.

To generate **split cut files**, the user can click *Save splits* button below the histogram.

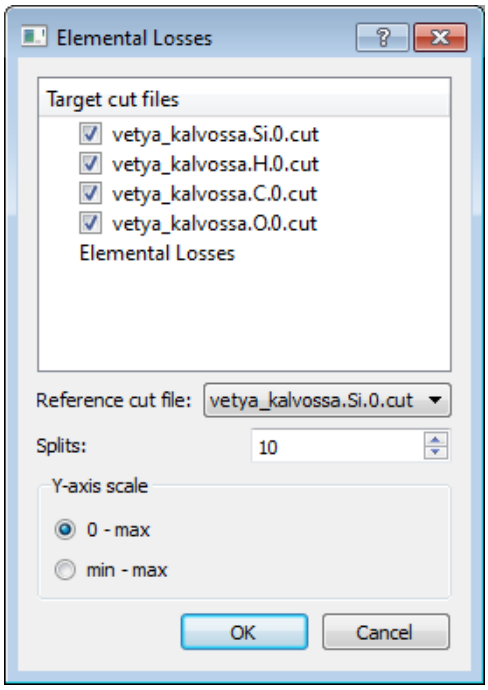

Figure 4.11: Elemental losses dialog.

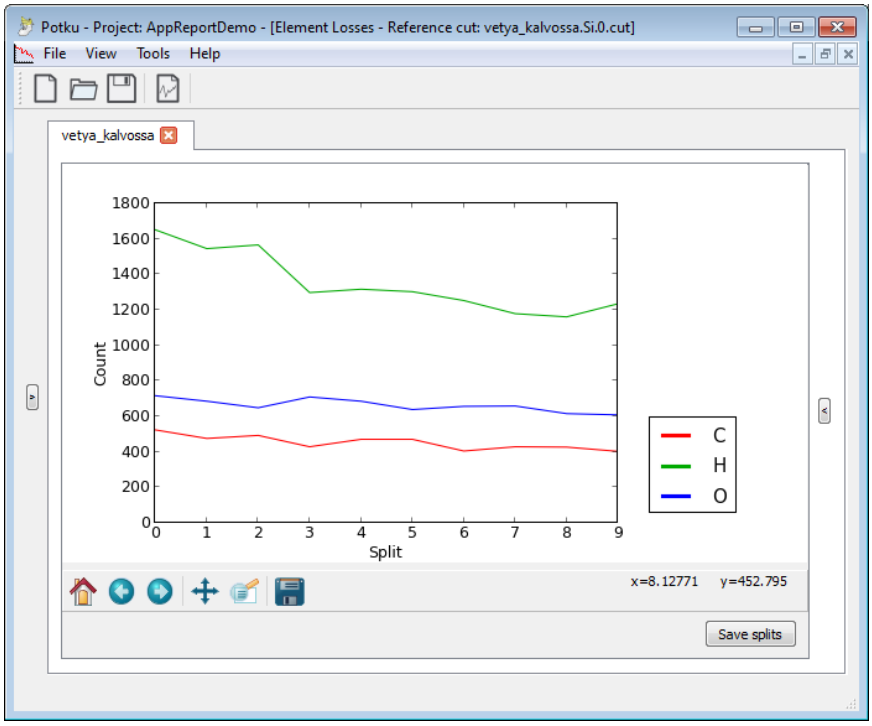

Figure 4.12: An elemental losses histogram.

Each split file contains the specified portion of the cut file from which it is derived. Split files can be used later like the cut files from which they are derived from. The other buttons are described in Section 4.4.

4.8 Generating an Energy Spectrum Histogram

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* (see Figure 4.1). This opens a dialog in Figure 4.13, in which all the available cut files and split files that can be used are listed in a check list, as well as the option to set the bin width of axes. Cut files have to exist to generate an energy spectrum. Clicking *OK* button will generate the histogram in Figure 4.14 with given parameters. The buttons of the energy spectrum histogram are described in Section 4.4.

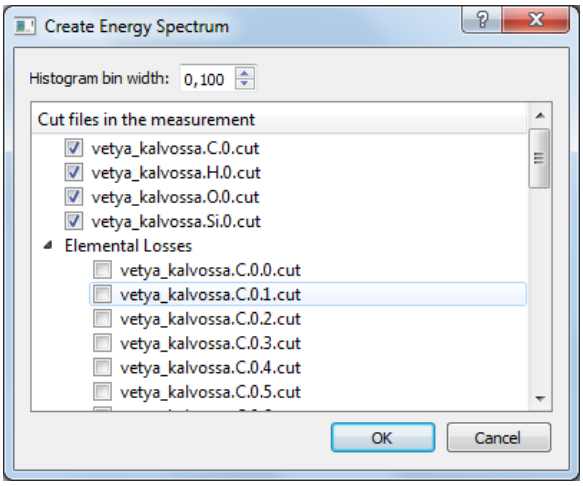

Figure 4.13: Energy spectrum dialog.

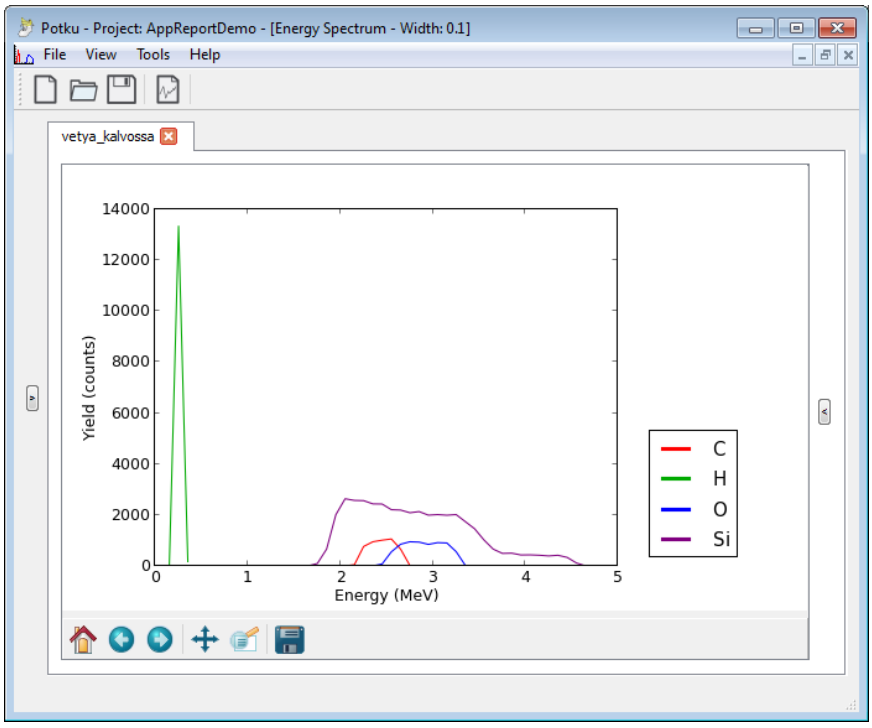

Figure 4.14: An energy spectrum histogram.

4.9 Generating and Analyzing a Depth Profile

To generate a **depth profile**, the user can click *Depth profile* button on the right sidebar (see Figure 4.1) or from the menu *Tools* the command *Create Depth Profile*. A dialog in Figure 4.15 opens, where the user can select the cut files and split files to be included in the depth profile as well as the units used in the X-axis. Cut files have to exist to generate a depth profile. Clicking *OK* button will generate and show the depth profile with the given parameters as seen in Figure 4.16.

The tools unique to the depth profile are as follows:

A. *Limit setting* sets the limits for calculating percentages of elements.

B. *Area selection* determines the areas affected by tool C.

C. *View* toggles between absolute and relative plotting of the depth profile data.

Clicking the *view* button (C) will toggle between an absolute view and relative view. In **absolute view**, data points are plotted according to their actual values. In **relative view**, data points are scaled to the total value of all the selected elements, which

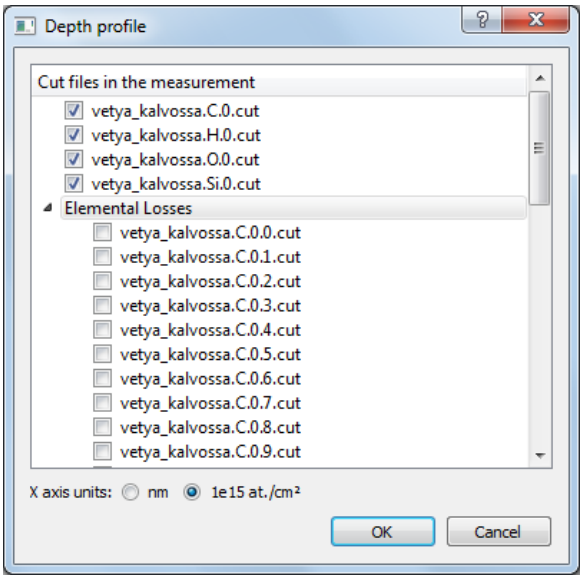

Figure 4.15: Depth profile dialog.

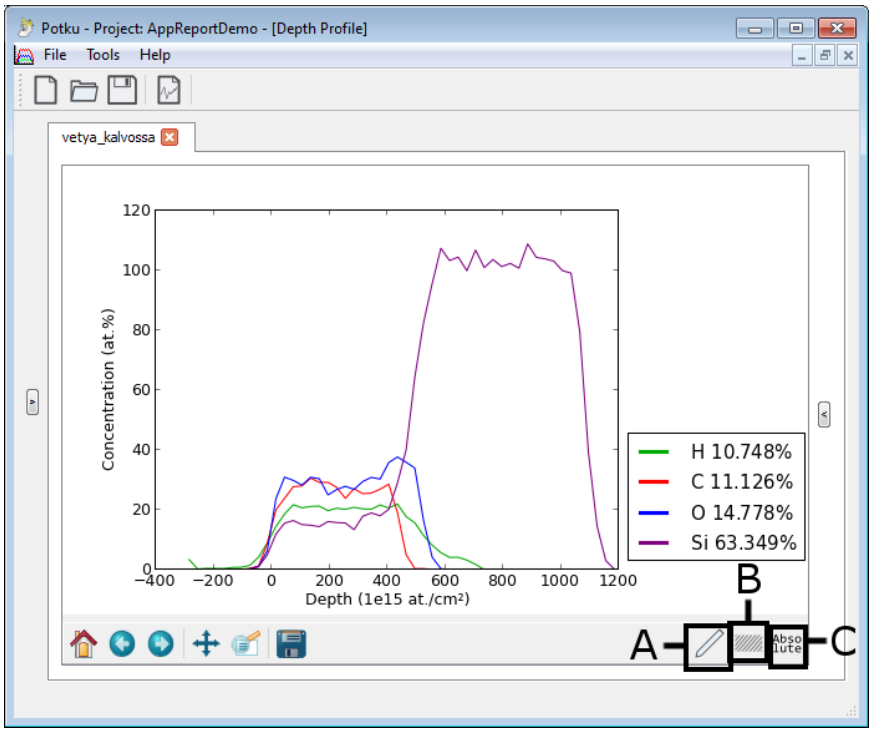

Figure 4.16: A depth profile in the absolute view.

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 and in absolute view in Figure 4.16.

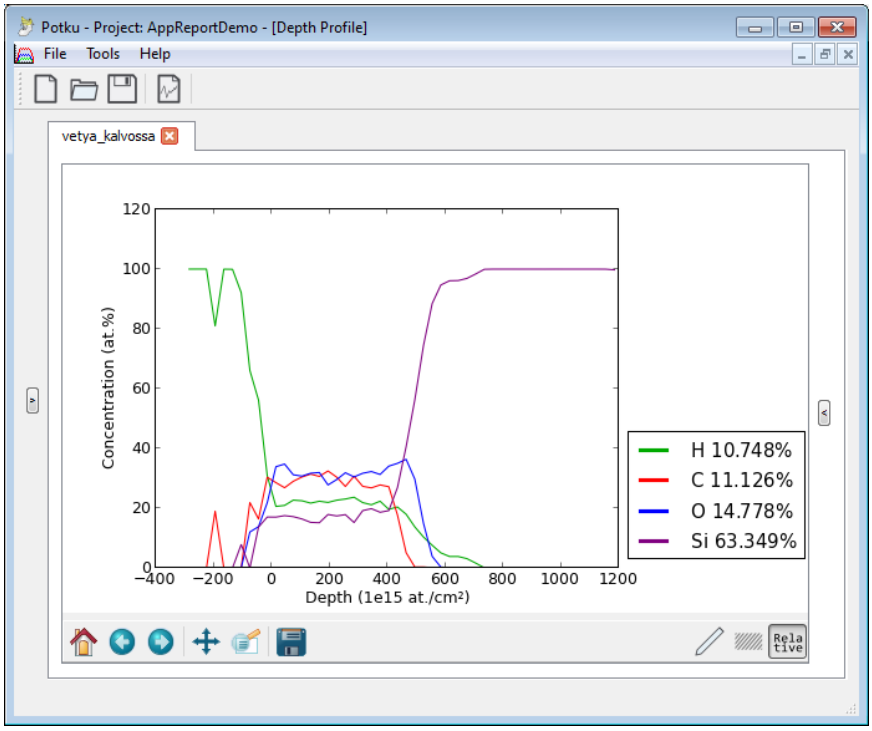

Figure 4.17: A depth profile in the relative view.

With *limit setting* tool (A), 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 (A) also enables the *area selection* button (B). The 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 (C) 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 clicks *view* button, while the area within the range will still be plotted with actual values. Usage of the button is demonstrated in Figure 4.19

The other buttons of the Depth Profile widget are described in Section 4.4.

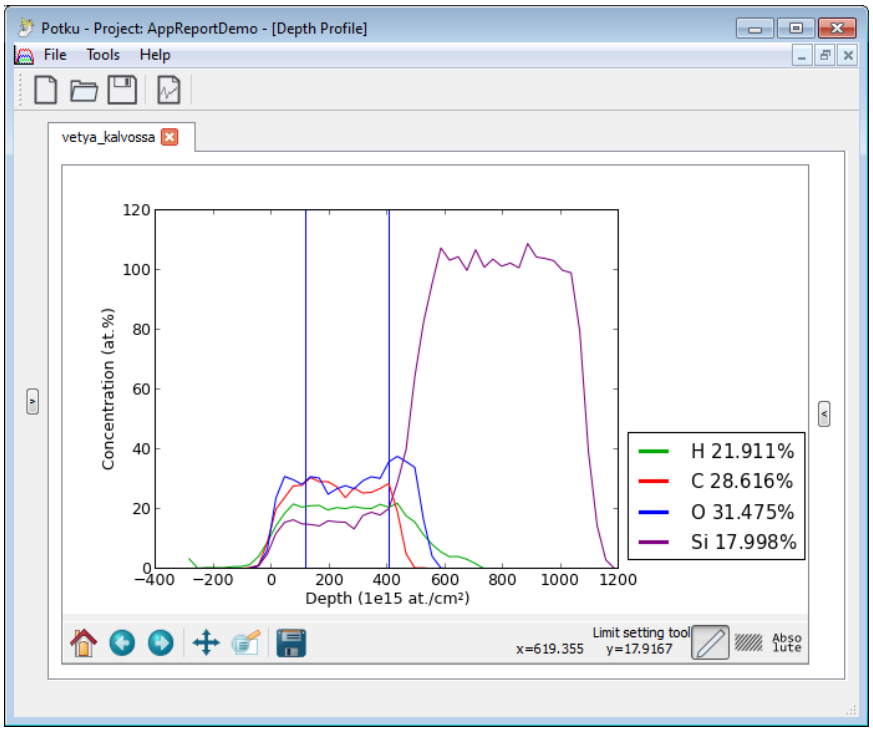

Figure 4.18: A depth profile with a custom range.

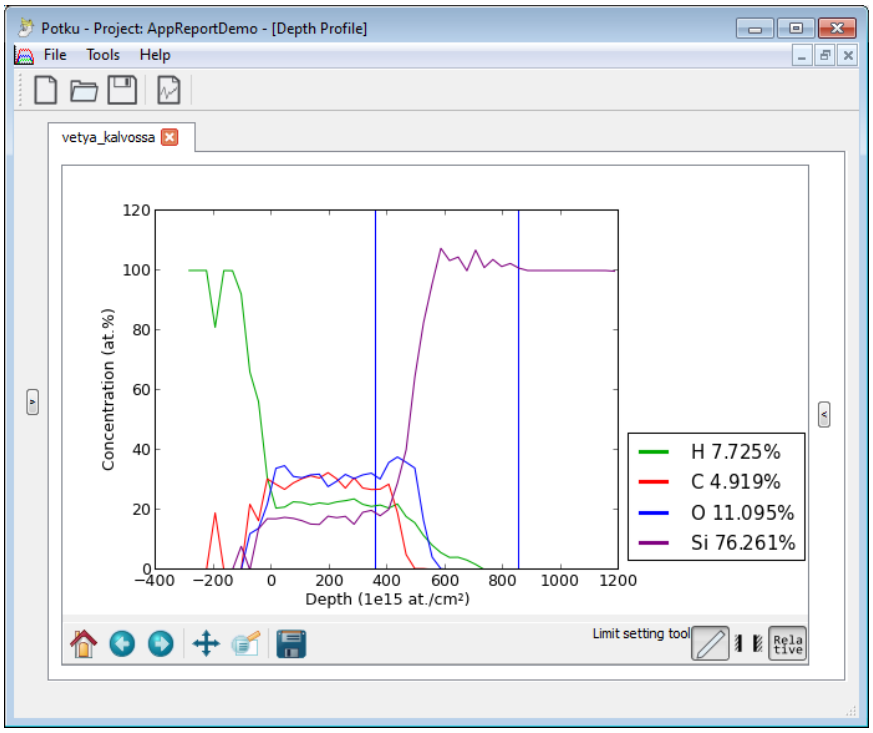

Figure 4.19: A depth profile with only areas outside of set range plotted relatively.

5 Application Structure

The chapter describes the different components in Potku application and their relations to each other. Potku is a 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`, `carbon_stopping` and `zbl96`.

5.1 Components and Software

Potku uses several existing Python libraries to achieve certain functionalities.

Matplotlib is a library for plotting different histograms. In Potku, the library was used for the plotting of ToF-E histograms, elemental losses histograms, energy spectrum histograms, as well as 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 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 C components, `platform` for determining what operating system the application is currently running on, and `csv` for importing data from file.

Reinhardt icons are a set of icons [6] used in Potku. The icons are licensed under LGPL.

In addition to the Python libraries, several C programs provided by the customer are also integrated into Potku.

carbon_stopping is a program that calculates the stopping of elements against carbon. The 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`. The component is necessary for generating depth profiles. The usage of `erd_depth` is further documented in [3].

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

zbl96 is a program that calculates stopping of particles, when they hit a material. The component is necessary for creating energy spectrums and depth profiles, and for performing ToF calibration.

5.2 General Structure

The structure of the the application is roughly presented in Figure 5.1. The structure is documented with more detail in the Potku class documentation [10].

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 File*. External programs are called by *Energy Spectrum*, *Depth Profile* and *ToF Calibration* utilities.

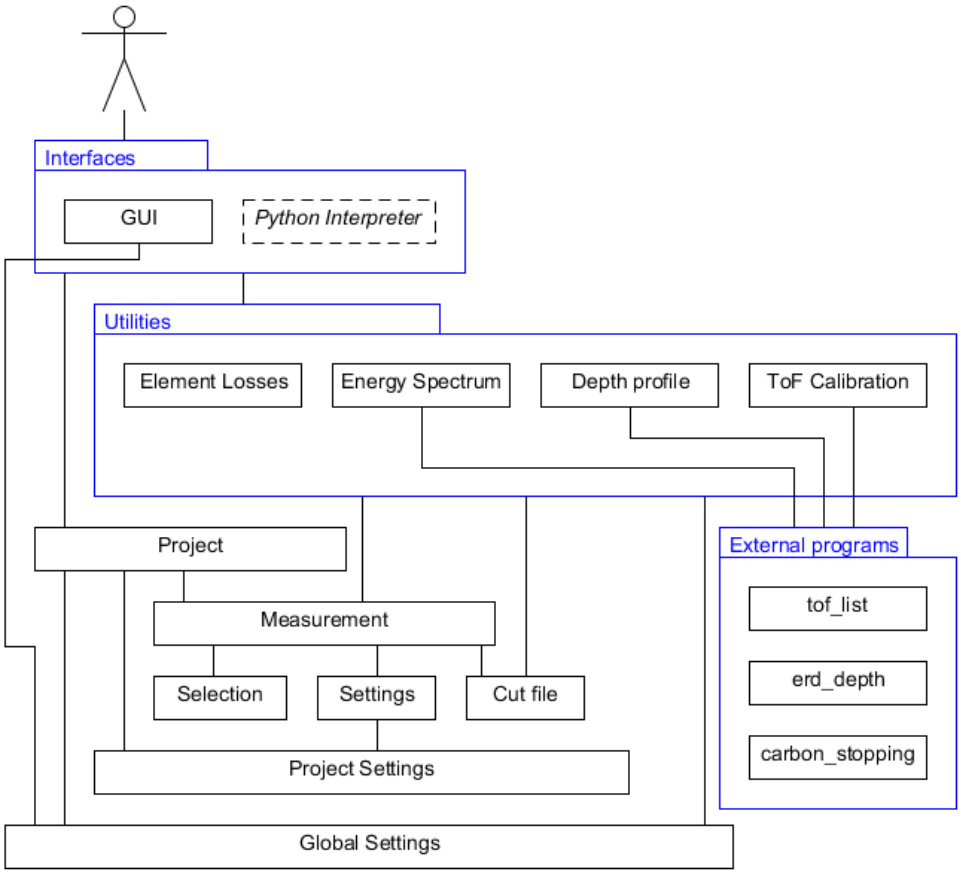

Figure 5.1: The structure of Potku.

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 (see Section 4.4), and it contains all the events of one of those selections. The naming convention of the file is `[Measurement].[Isotope][Element].[Id].cut`. For example, the first instance of a cut file containing the data points ^{12}C from a measurement named *vetya_kalvossa* would be `vetya_kalvossa.12C.0.cut`. The meanings of these fields are as follows:

`Measurement` is the name of the measurement file.

`Isotope` is the isotope of the selected element.

`Element` is the symbol of the selected element.

Id is the unique identifier of the element. It is used if there are multiple selections of the same element in the ToF-E histogram.

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 format:

```
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 utility in Section 4.7 can save **split files** that are very similar to cut files. In fact, a split file contains a $1/n$ portion of the data content in the source cut file, where n is the amount of split files created from the source cut file. The naming convention is similar to cut file, with the addition of one part: `[Measurement].[Isotope][Element].[Id].[SplitId].cut`. For example, the fourth split of

the the first instance of a cut file containing the data points ^{12}C from a measurement named *vetya_kalvossa* would be `vetya_kalvossa.12C.0.3.cut`. `SplitId` is a unique identifier given to split files. The rest of the fields serve the same purpose as in cut files.

5.4 Integration of External C Components

Most of the mathematical calculation in the application is carried out with `tof_list` and `erd_depth`. **`tof_list`** calculates energies for cut files based on the time-of-flight. The 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.

The external component **`erd_depth`** produces output files from which depth profiles are calculated. The naming convention of these files is `depth.[Element]`, where `[Element]` is the symbol of the element. For example, a depth file containing depth data of silicon would be named `depth.Si`. In addition, `erd_depth` always creates a `depth.total` file. The content of both of these files is explained in Table 5.3.

Depth (1e15 at./cm2)	Depth (μg/cm)	Depth (nm)	At. pct.	Concentration
-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, the external C component `carbon_stopping` was also in-

tegrated into Potku. The component calculates stopping data of elements against carbon, and is necessary for the ToF calibration.

For calculating stopping of particles when they hit a material, all three aforementioned C programs use `zbl96`. These programs call `zbl96` by themselves, and it is not called from the Python code. These programs are also, capable of getting the stopping data from SRIM, or other list data, but `zbl96` was used in this project.

6 Programming Practices

Preliminary programming 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. Most of his advice was taken into account, but not all. For instance, Potku does not use the objects as much as Itkonen recommended.

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 style guide 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 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 used in those codes were followed.

Each Python module contains function or method, and possibly line or block - specific commentary to help understand the purpose of the affected code region. Function or 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.

Most of the planned practices were followed. Naming conventions followed the Python style guide, or CamelCase when PyQt was used. Each method and function of each Python module was commented as planned. All the documentation was written in English and names were kept as self-explanatory as possible. The three-step version numbering of modules was not followed at all, and all the modules were always marked as version 1.0.

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

Grouping practices were not defined in the project plan [7]. The project team did however try to group python modules to packages that would contain modules serving similar purposes.

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 and other resources of Potku are spread to packages according to their purpose:

`Dialogs` contains the classes that act as the dialogs of the application.

`external` contains the C components from the customer.

`images` contains larger image files – not icons – used by the Potku interface.

`Modules` contains classes that form objects.

`ui_files` contains the ui files from which the user interface is generated from.

ui_icons contains the icons used by Potku. Some of the icons are from The Reinhardt Icon Set [6].

Widgets contains classes that form widgets to be used with PyQt.

Since so many tools of the application use Matplotlib, `Widgets/MatplotlibWidget` was split into several separate child classes to maintain a bearable line count.

6.4 Development Platform

For programming 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. Programming was mostly done on the Windows workstations and the Linux platform was mostly reserved for system testing purposes. The project team received a Mac workstation couple of weeks before the end of the project as well, but it was used mostly for system testing purposes as well.

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]. The 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 be integrated into the master. Unit testing was planned to the methods and functions, that return values. Though not all functions and methods that could be unit tested however were not unit tested as was planned. Unit tests that were carried out were written with Doctest. Since most of the calculations required by Potku are done in external C programs, 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.

Unit and integration testing of each module was done by the team member who had programmed the module.

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. Full system testing plan can be found in [9].

If a test case would fail during a system test, all the test cases would be carried out again in full at a later time, after the necessary fixes had been done. This was to ensure that a fix in one portion of the program would not break some other portion of the program.

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 instructor in charge, and the representatives of 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 releases of Potku for the organization to test. This however did not become reality. Instead, new or improved functional source code was uploaded to the master branch of the Potku repository 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 or bi-weekly project meetings. During the meetings, the project team would demonstrate the new implementations and modifications 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

System tests were carried out in Linux and Mac operating systems on about 20 occasions by Aalto. During each of these system tests, the result of each test case was noted. During each of these tests, the external C programs were also compiled. Each of these system tests revealed some bugs in the software, such as incorrect appearance of the user interface and problems in running external C programs.

There was a handful of faults that were commonly revealed in the system tests. Most esssential faults found and fixed were:

- User interface appeared "squashed" which severely hindered the usage of Potku, especially the more compact dialogs.
- External C programs did not compile directly in the form they were received from the customer. This resulted in failed test cases for energy spectrum, depth profile and ToF calibration.
- Few buttons did not have any functionality attached to them. This was often the case with toolbar buttons, and *OK* and *Cancel* buttons.

The last system test took place on 21.5.2013. **All platforms fully passed the system test.** During the last system test, there was a strange corruption error in `tof_list`. This was probably caused by compilation, since the source code or the makefile had not been modified in weeks, and the program has worked in previous runs.

Other bugs have been noted outside of system testing and are listed in the e-mail archive. Many of them were pointed out by instructor Santanen during his two test runs on 30.4.2013 and 8.5.2013.

8 Realization of Objectives

The 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. The chapter also describes certain implementations that were done, but the project team thinks the implementation is not done in best way possible.

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 *mandatory* (marked as 1), *important* (2), *possible* (3), *idea* (4), and *will not be implemented* (5). Mandatory 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 and tested**. Of the priority 1 requirements 43 of 45 were implemented or partly implemented. Of the 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.

The partly implemented mandatory requirements are the following:

1.4 An element uses an efficiency file if such exists. The external program `tof_list` currently has the capability of doing this. Currently though, it does not do this for any other element than ^{1}H.

1.11 GUI functions can be used through interpreter. Efforts were made to separate the functionality from the GUI for this requirement. Due to time constraints however, this functionality is only partly implemented.

The entirely unimplemented mandatory requirements are the following:

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 re-

placed with another. However, due to time constraints a replacement stopping model could not be integrated into Potku.

1.11 GUI produces (command log) process of sample collection in a python file was not implemented due to time constraints.

8.2 Unsatisfactory Solutions in the Implementation

While solutions selected by the project team 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 external C components should be done more gracefully than simply calling their executables with `subprocess.call`. Using the method is vulnerable to any system policies that are forced on running executables.

The class `Modules/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 challenges 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 challenge was the **output format of `erd_depth`**. It 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 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 the output of `erd_depth`.

Some **compromises** had to be made on the GUI that was developed **on Windows**.

When tested on Linux and Windows, several parts of the GUI that would appear as planned on Windows, appeared incorrectly on the other operating systems. To correct Potku GUI on the other operating systems, it had to be made less compact than planned.

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

Since no-one in the project team had extensive knowledge on 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 by the instructions of the technical instructor.

The subject (accelerator-based material physics) was also very alien to the project team. The challenge 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 customer has decided that Potku will go through further development after the project. In the chapter several tips are given 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 trouble it's usage. The following bugs should be prioritized when the application enters further development.

- Depth profiles are 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.
- 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.
- When measurements are deleted in a project, their data is not properly erased from memory. This memory leak could hog extensive amounts of memory in prolonged use of Potku. Some measures have been taken against this issue, but it remains unsolved.

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 separate compiled versions of the programs for all three operating systems. As is, a compiled Windows executable of each external C program is included, but compiled programs for Mac and Linux are not.

The external C components could be programmed again in Python to allow a smoother integration into rest of the software. The customer has plans of doing this in fur-

ther development. Although the calculation time differences between Python and C should be considered.

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.

The usage of `logger` 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 user interface for software used in the analyzation of data received from a recoil spectrometer. The customer of the project was the research team of accelerator-based material physics in Department of Physics at University of Jyväskylä. Of the mandatory requirements, 43 of 45 were implemented or partly implemented. All requirements not implemented were agreed with the customer to be left for further development outside the project.

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

Potku application was developed using PyQt GUI library, Matplotlib plotting library, as well as numpy and scipy mathematical libraries. 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 a usability expert and the source code was reviewed twice by the technical instructor of the project. The application was tested with ad hoc, unit, integration and system testing. Further development of the software is planned to start immediately 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, 20.5.2013.

[2] Iso-Ahola Pekka, Perttola Jussi and Tuovinen Tommi, "Kuvatus Project, Application Report", University of Jyväskylä, Department of Mathematical Information Technology, 26.4.2012.

[3] Sajavaara Timo, "Analysis with TOF-ERDA at IMEC", 20.12.2004.

[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 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, 18.4.2013.

[8] Aalto Jarkko, Konu Timo, Kärkkäinen Samuli, Rahkonen Samuli and Raunio Miika, "Potku-sovellusprojekti, Projektiraportti", University of Jyväskylä, Department of Mathematical Information Technology, 27.5.2013.

[9] Aalto Jarkko, Konu Timo, Kärkkäinen Samuli, Rahkonen Samuli and Raunio Miika, "Potku Project, System Testing Plan", University of Jyväskylä, Department of Mathematical Information Technology, 21.5.2013.

[10] Aalto Jarkko, Konu Timo, Kärkkäinen Samuli, Rahkonen Samuli and Raunio Miika, "Potku Project, Class Documentation", University of Jyväskylä, Department of Mathematical Information Technology, 20.5.2013.