Evaluating and comparing tools for generating PDF files

Jesper Niemi
Abstract

Evaluating and comparing tools for generating PDF files

Jesper Niemi

There exists many software tools for generating PDF files, some of which have advantages over others. To determine what tool is better than others is of interest when one has a choice to make between different tools; one obviously wants to pick the tool that suits the user the best. In this thesis, four different software tools for generating PDFs, LaTeX, Groff, HTMLDOC and ReportLab, are evaluated under a set of requirements and compared against each other to determine which of these tools fulfills the requirements the best. The evaluation was done by researching literature, manuals and user guides, along with practical testing to determine whether a tool fulfilled a requirement or not. The results from the evaluation and comparison are that LaTeX has clear advantages over the other tools. Thus, the conclusion of this thesis is that LaTeX should be chosen as the tool for generating PDF over Groff, HTMLDOC and ReportLab.
Contents

1 Introduction 6
  1.1 Problem ................................................................. 6
  1.2 Requested Solution .................................................. 6
  1.3 Prior Work .............................................................. 7
  1.4 My Contributions .................................................... 8

2 Methods 8

3 Background 9
  3.1 LaTeX ................................................................. 9
  3.2 Groff ................................................................. 10
  3.3 HTML ................................................................. 12
  3.4 Python ............................................................... 14

4 Requirements 18
  4.1 Hard requirements .................................................. 18
  4.2 Soft requirements ................................................... 19

5 Evaluation and Comparison 21
  5.1 Dynamic and non-dependent on templates ......................... 21
  5.2 Outcome must be PDF ................................................ 21
  5.3 Table ................................................................. 21
  5.4 Embedded images .................................................... 21
    5.4.1 LaTeX .......................................................... 21
    5.4.2 Groff ............................................................ 22
    5.4.3 HTMLDOC ....................................................... 22
    5.4.4 ReportLab ....................................................... 22
  5.5 Footnotes ............................................................ 22
    5.5.1 LaTeX .......................................................... 22
    5.5.2 Groff ............................................................ 22
    5.5.3 HTMLDOC ....................................................... 22
5.5.4 ReportLab ........................................... 23
5.6 Footers .................................................. 23
  5.6.1 LATEX ............................................ 23
  5.6.2 Groff ............................................. 23
  5.6.3 HTMLDOC ........................................... 23
  5.6.4 ReportLab ......................................... 24
5.7 Table headings ........................................ 24
  5.7.1 LATEX ............................................ 24
  5.7.2 Groff ............................................. 24
  5.7.3 HTMLDOC ........................................... 25
  5.7.4 ReportLab ......................................... 25
5.8 Page numbering ........................................ 26
  5.8.1 LATEX ............................................ 26
  5.8.2 Groff ............................................. 26
  5.8.3 HTMLDOC ........................................... 26
  5.8.4 ReportLab ......................................... 27
5.9 Free of charge ........................................ 27
5.10 Works without the Internet ............................. 27
5.11 Works on Windows ...................................... 28
5.12 Works on Linux ........................................ 28
5.13 Graphs ................................................ 29
  5.13.1 LATEX ............................................ 29
  5.13.2 Groff ............................................. 29
  5.13.3 HTMLDOC ........................................... 30
  5.13.4 ReportLab ......................................... 30
5.14 Mixed page orientation (portrait/landscape) in same document 33
  5.14.1 LATEX ............................................ 33
  5.14.2 Groff ............................................. 33
  5.14.3 HTMLDOC ........................................... 34
  5.14.4 ReportLab ......................................... 34
5.15 Disk usage ............................................ 35
5.15.1 \LaTeX{} .......................... 35
5.15.2 Groff .......................... 36
5.15.3 HTMLDOC .......................... 36
5.15.4 ReportLab .......................... 36

5.16 Software maturity .......................... 36
5.16.1 \LaTeX{} .......................... 36
5.16.2 Groff .......................... 37
5.16.3 HTMLDOC .......................... 37
5.16.4 ReportLab .......................... 38

5.17 License .......................... 38
5.17.1 \LaTeX{} .......................... 38
5.17.2 Groff .......................... 38
5.17.3 HTMLDOC .......................... 40
5.17.4 ReportLab .......................... 40

5.18 Performance .......................... 40
5.18.1 \LaTeX{} .......................... 41
5.18.2 Groff .......................... 42
5.18.3 HTMLDOC .......................... 42
5.18.4 ReportLab .......................... 42

5.19 Standalone installation .......................... 42
5.19.1 \LaTeX{} .......................... 42
5.19.2 Groff .......................... 43
5.19.3 HTMLDOC .......................... 43
5.19.4 ReportLab .......................... 43

5.20 Summary .......................... 44

6 Related Work 45

7 Discussion of the Results 46
7.1 \LaTeX{} .......................... 47
7.1.1 Pros .......................... 47
7.1.2 Cons .......................... 47
7.2 Groff ............................................. 47
  7.2.1 Pros ..................................... 47
  7.2.2 Cons .................................... 48
7.3 HTMLDOC .................................... 48
  7.3.1 Pros ..................................... 48
  7.3.2 Cons .................................... 49
7.4 ReportLab .................................... 49
  7.4.1 Pros ..................................... 49
  7.4.2 Cons .................................... 50
7.5 Conclusion .................................. 50

8 Reflection and Future Work 51
  8.1 Reflection ................................. 51
  8.2 Future Work ............................... 53

Appendices 58
A Today’s reports 58
B HTML footnote example 59
C ReportLab footer 60
D \LaTeX{} landscape orientation 61
E ReportLab page orientation 62
F \LaTeX{} installation guide 64
  F.1 Windows 10 .............................. 64
  F.2 Debian 8.8 ............................... 65
G Groff installation guide 66
  G.1 Windows 10 .............................. 66
  G.2 Debian 8.8 ............................... 66
H HTMLDOC installation guide
   H.1 Windows 10 .................................................. 67
   H.2 Debian 8.8 .................................................. 67

I ReportLab installation guide
   I.1 Windows 10 .................................................. 68
   I.2 Debian 8.8 .................................................. 68
1 Introduction

Biotage is a company headquartered in Uppsala that offers solutions, knowledge and expertise in the areas of analytical chemistry, medicinal chemistry, peptide synthesis, separation and purification[1]. A vital function of the instruments made at Biotage is the ability to generate reports, showing information such as the method run configuration, parameters and outcome from the run. Every instrument made at Biotage have its own way of generating reports, with its own strengths and weaknesses. This thesis will focus on one of Biotage’s instruments, namely ”Extrahera”.

1.1 Problem

The current implementation of the report generating program on Extrahera is not flexible enough to guarantee further development. The implementation consists of static templates which requires a lot of collaborators when information needs to be added to them. To mention some obstacles, the program generating the reports:

- are dependent on the marketing designer and Adobe Indesign when changes needs to be done to the templates, even if the changes are small.
- has to include all thinkable fields in the reports, even if only a small subset of data is wished to be published.
- sometimes need to provide different templates on the same page because different configurations of the same method differ too much.
- cannot easily make information ”spill over” to the next page if it does not fit on one page.

1.2 Requested Solution

Biotage wants to move away from their current solution into a dynamic and flexible way of producing reports. In order to do this, Biotage has requested that available open-source tools should be examined and evaluated in a highly constrained environment, similar to that of the environment of Extrahera. These are the constraints on the environment:
PDF generation must be totally dynamic and not dependent on predefined template files

The outcome must be a PDF (Portable Document Format)

Must be able to output the corresponding data as the reports that exist today does

Must be free of charge

Must work without the use of the Internet

Must be able to run on both Windows and Debian

The open-source tools that Biotage wishes to be examined are at least the following:

- \LaTeX
- Groff
- HTML
- Python

The outcome of this thesis work should be a comprehensive overview of at least these open-source tools; their pros and cons and how they fulfill the requirements of the environment. Finally a conclusion should be presented on which tool, if any, that best suits Biotage.

1.3 Prior Work

No prior work on this exact subject could be found during the search for relevant work, i.e. the author of this thesis could not find any work where open-source tools are evaluated, under a set of requirements, and compared for their ability to produce PDF files.
1.4 My Contributions

This thesis will hopefully serve as a guide not only for Biotage, but for anyone who is looking for a PDF generating open-source tool. With information about every tool presented, their pros and cons and what requirements they fulfill the reader will be able to make an educated choice on which tool that suits their needs the best, and the short code snippets presented in this thesis will hopefully serve as an introduction and a quick-start on how to use the tools. This thesis also opens up possible future works, where to most interesting work would be to develop a complete general solution usable on all of Biotage’s instruments.

2 Methods

This study will be a literature research along with practical testing, meaning that literature, manuals and user guides will be examined to determine whether a tool fulfills a requirements or not, and tests will be performed to confirm claims. These tests will be included, where the author finds it necessary, in the thesis in such a manner that the reader can recreate them, should they want to.

Although no prior work on this exact subject could be found, similar work has been done, where software tools have been evaluated and compared under a set of requirements. In one study, made by Yang and Wagner [3], 12 commonly acknowledged features characterizing the next generation catalog were presented. Seven open source and ten proprietary discovery tools were then evaluated and compared against these features, to see whether or not these features were present in the tools. Their results were then presented in tables. In another study, made by Shahid and Ibrahim [4], 19 different software tools for test coverage were presented and described. These tools were then evaluated and compared against five evaluation criteria, namely language support, instrumentation, coverage measurement, GUI and reporting. Their results were then presented in a summary table. The section ”Related Works” describes these studies in greater detail.

These studies, their structure and ways-of-working, will inspire the study described in this paper. Just like Shahid and Ibrahim worked in their study, this thesis will present every tool to be examined, explaining informa-
tion about the tool such as when and why it was created, where it is usually used and licenses and maintenance. The thesis will then continue, just like Shahid and Ibrahim, and Yang and Wagner did in their studies, by presenting the evaluation criterias (or the ”Requirements”, as called in this thesis), explaining them in detail and motivating their necessity. The evaluation and comparison will take the same form as in both studies, where each tool is checked against a requirement in a table. Whether or not a tool fulfills a requirement will be justified by claims in literature, manuals and user guides, and tests will be done to confirm these claims. Lastly a conclusion will be presented, where a discussion on the results from the evaluation will be held. Each tool will be discussed, explaining what their pros and cons are, and which of the tools, if any, that fulfills the requirements the best.

3 Background

This section contains background information about the open-source tools examined in this project. Since Biotage’s reports contains mainly of tables, a small piece of code is presented at the end of every tool description, showcasing how the most simple table can be created with that tool.

3.1 \LaTeX

\LaTeX is a document preparation system, created by Leslie Lamport in the early 1980s, and is widely used in scientific and academic communities as well as in industry [2]. The program uses \TeX’s typesetting engine and macro system, which was created by Donald Knuth some years earlier. The kernel of \LaTeX is very stable to this day, and will probably remain stable in the future, as no unnecessary ”enhancement” will be added to it. The reason for this is because in the early 1990s \LaTeX underwent a ”too much development” period, which led the unfortunate result in different \LaTeX ”dialects”; incompatible \LaTeX formats came into use at different sites. In order to put an end to this, the development of a new standard for \LaTeX was started by the \LaTeX Project Team. The release of this new standard, announced in 1994, had every extension brought under a single format and furthermore resulted in a more powerful system with both a robust mechanism for extension (via \LaTeX’s package system) and a solid technical support and maintenance
system. The package system together with the development guidelines and the legal framework of the LATEX Project Public License (LPPL) have enabled LATEX to remain almost entirely stable while supporting a wide range of extensions.

When processing a document, LATEX reads and writes to many files, though most of this happens behind the scene, and most users does not know, nor do they need to know, how it all works together [2]. However, some of the most common and important files and their functions are listed below [2]:

- **plain text file**: This is the input file to LATEX, containing formatting commands and the contents of a document in question, as well as information about what subsidiary files should be used. They often have the file extension `.tex` and can be written with a basic text editor.

- **class files**: Class files have the file extension `.cls` and contain structure and layout definitions. LATEX is distributed with with five standard classes: **article**, **report**, **book**, **slides** and **letter**.

- **package files**: These files have the file extension `.sty` and are used to customize the document classes.

- **formatted output file**: This file is the formatted result of the LATEX program given a plain text tile. The file typically has the file extension `.dvi` or `.pdf`

Table 1 shows the code for creating a simple table in LATEX, as well as the generated output when executing the code. This output was achieved by executing the following command in the Windows 10 command prompt (the command is the same for Debian 8.8):

```bash
pdflatex InputFile.txt
```

### 3.2 Groff

*Groff* (GNU troff) is, similar to LATEX, a document preparation system. Just like LATEX the program takes as input a text file containing formatting commands and outputs a typeset document on a variety of devices [5].
Table 1: LATEX table example and generated output

<table>
<thead>
<tr>
<th>InputFile.txt</th>
<th>OutputFile.pdf</th>
</tr>
</thead>
<tbody>
<tr>
<td>\documentclass{article}</td>
<td>This is a</td>
</tr>
<tr>
<td>\usepackage[english]{babel}</td>
<td>LATEX table example</td>
</tr>
<tr>
<td>\begin{document}</td>
<td>This &amp; is &amp; a&amp;</td>
</tr>
<tr>
<td>\begin{center}</td>
<td>\hline</td>
</tr>
<tr>
<td>\begin{tabular}{ l</td>
<td>l</td>
</tr>
<tr>
<td>\hline</td>
<td></td>
</tr>
<tr>
<td>This &amp; is &amp; a&amp;</td>
<td></td>
</tr>
<tr>
<td>\hline</td>
<td></td>
</tr>
<tr>
<td>LATEX &amp; table &amp; example&amp;</td>
<td></td>
</tr>
<tr>
<td>\hline</td>
<td></td>
</tr>
<tr>
<td>\end{tabular}</td>
<td>This &amp; is &amp; a&amp;</td>
</tr>
<tr>
<td>\end{center}</td>
<td>\hline</td>
</tr>
<tr>
<td>\end{document}</td>
<td>\hline</td>
</tr>
</tbody>
</table>

Groff saw its beginning in the around 1965 when the formatting program RUNOFF was written by Jerry Saltzer. This unusual name for a document formatting program came from the phrase "run off a document", meaning to print it out [5]. It ran on the CTSS (Compatible Time Sharing System), which was a MIT project, and was later ported to other architectures by Bob Morris. Its name had now become roff (an abbreviation of runoff).

During 1971, developers of the UNIX operating system wanted to get a PDP-11 (a minicomputer) and proposed the development of a document formatting system for the AT&T patents division in order to justify the cost [5]. This program, written by J. F. Ossanna, was a reimplementation of earlier version of roff. The version of roff that would provide the basis for all future versions, called nroff (for "Newer roff"), was written by the same development team as a response to the need of a more flexible language. Ossanna wrote a version of nroff that would drive the Graphic Systems CAT Phototypesetter and it was called troff, for "typesetter roff".

Work on preprocessors for troff began some time later, since there were things that could not be done easily by troff. The preprocessors includes e.g. the eqn preprocessor, which allows mathematical formulae to be specified in a much simpler manner [5].
Since troff thus far was written specifically for the PDP-11 machine, with output specifically for the CAT phototypesetter, it was very much device dependent [5]. The task of writing a device independent version of troff was taken by Brian Kernighan. This new version, named ditroff for "device independent troff" produced device independent code, that was easy for postprocessors to read and translate, and had several extensions, including drawing functions.

In the early 1989 a GNU implementation of ditroff was being worked on by James Clark, and the first version, Groff 0.3.1, was released June 1990 [5]. Groff is now released under the GNU General Public License (GPL) v3 and the current version, version 1.22.3, was released in November of 2014 [6].

Groff provides a number of macro packages, that specify how certain routine operations, such as starting paragraphs, printing headers etc., should be done. The most common macro packages are man, mdoc, me, ms and mm [5]. The most relevant to this work is probably ms, which is suitable for reports, letters, books and user manuals among others. A short example on how to create tables with Groff, and the result of running the example, is presented in table 2. The following command was executed in the Debian 8.8 shell to achieve the result seen in table 2 (the command is the same for Windows 10):

```
groff -t -ms -Tpdf InputFile.txt > Outputfile.pdf
```

### 3.3 HTML

HTML (HyperText Mark-up Language), invented by Tim Berners-Lee, is a language for making documents, viewable in Web browsers [9]. During its
life-span it has undergone dramatic changes from being a simple language with a small number of tags, to a complex system with a lot of features such as animated images and sound. Its journey towards standardization has been particularly long and bumpy.

Tim Berners-Lee, working in a computing service section of CERN, the European Laboratory for Particle Physics in Geneva, Switzerland, in the late 1980s had an idea of enabling researchers from remote sites around the world to organize and pool together information [9]. Not only making research documents available for downloading to individual computers as files, Tim wanted to link the text in files themselves and cross-reference to other files, meaning that whilst reading one research paper, one could quickly display parts of another paper holding relevant information simply by following a link. He thought this could be done with some form of hypertext, with which you could already follow links and do hypertext jumps, although limited only to files on the same computer. Tim had previous experience with document production and text processing, in 1980 he had created his own hypertext system ”Enquire”, and now decided to create a new hypertext language that made global hypertext links possible. The language he created was called HTML and came with its own simple protocol, HTTP (Hyper-Text Transfer Protocol), for retrieving other documents’ text via hypertext links. HTML was strongly based on SGML (Standard Generalized Mark-up Language) and used a lot of its feature and structure, such as pairs of tags, e.g. `<TITLE>` and `</TITLE>`. In 1990, Tim’s prototype Web browser on the NeXT computer came out.

As HTML began its development journey as a programming language, a lot of people and companies, groups and forums were included respectively created [9]. Some mentionable names and events are Dave Raggett, who early exchanged ideas with Tim on what features HTML would need to mature. The World Wide Web conference was a meeting first held in 1994, where Dan Connolly talked about the importance of a proper HTML specification, and Raggett presented his idea for HTML+. The same year the World Wide Web Consortium (W3 Consortium) was created by Tim, including members such as Raggett and Connolly. Their purpose was to fulfill the potential of the Web through the development of open standards and HTML. The HTML Editorial Review Board (HTML ERB) was formed in 1996 to help with the standardization process of HTML. The board consisted of representatives from companies such as IBM, Microsoft and Netscape, as well as
members from the W3 Consortium and they began to work on "Cougar", the
code-name for the next HTML standard. The project was fully formalized
as HTML 4.0 in 1998. In 2014 and 2016 HTML 5 respectively HTML 5.1
was published as a W3C Recommendation [10].

Notice that HTML differs from the other software tools examined in
this thesis, in that it is mainly designed for the creation of web pages. Only
one method for converting HTML pages into PDF files was found during the
search for existing methods. It is shortly presented below:

**HTMLDOC**: HTMLDOC was created by Michael R Sweet as a response
to the need for generating high-quality documentation in printed and elec-
tronic forms [15]. It can take as input a HTML file (or a plain text file, i.e.
a .txt file) and convert it to a PDF file, and it can be called from the shell/-
command prompt [15]. The software’s current version is 1.9 and it runs on
Linux, macOS, Windows and most Unix operating systems [15]. It is avail-
able under the GNU GPL version 2 license and is maintained to this day by
its creator during spare time [15]. The software is limited to the versions of
HTML it can work with, e.g. the software supports only HTML 3.2 tables
[16]. The code for creating a simple HTML 3.2 table can be seen in table 3.

![HTMLDOC Table](image)

Given this code, contained in a file named **input.html**, the follow-
ing command was used in Windows 10 command prompt (the command is the
same for Debian 8.8) to retrieve the converted PDF file, named **output.pdf**:

```
htmldoc --webpage -f output.pdf input.html
```

The content of the **output.pdf** file can be seen in figure 1.

### 3.4 Python

In 1989, Guido van Rossum created a new programming language which
he named **Python** (taking inspiration from the television program *Monty*
Table 3: HTML table example

```
<!DOCTYPE HTML PUBLIC "−//W3C//DTD HTML 3.2 Final//EN">
<html>
<head></head>
<body>
<table border="1"
     cellspacing="0"
     cellpadding="3"
     width="15%">
<tr>
    <td>This</td>
    <td>is</td>
    <td>a</td>
</tr>
<tr>
    <td>HTMLDOC</td>
    <td>table</td>
    <td>example</td>
</tr>
</table>
</body>
</html>
```
Python’s Flying Circus) [7]. It was created as a response to the need of utility tools for the operating system Amoeba.

During the late 1980’s, Guido was working at Centrum Wiskunde & Informatica (CWI), a research center in Amsterdam, developing a distributed operating system called Amoeba [7]. During its development, the development team realized that they would need a large amount of user-level tools like an editor, a mail program, a login utility etc. A small team of people was assigned to work on these tools, but the progress was very slow since they wrote everything in the programming language C. Guido had done earlier work on a programming language called $ABC$ and wondered if, given the large amount of time they had for developing Amoeba, he would be able to develop a new programming language, similar to $ABC$, that would be suitable for quickly developing the OS utilities Amoeba needed. Therefore, during a long holiday break in December of 1989, Guido started working on this new $ABC$-like language, which he named Python, and after three months of work he could show his friends and colleagues what he had built.

Python quickly gained popularity within CWI and people started helping out with the development of Python [7]. After a year of use and development, Guido and his colleagues decided to release Python as a free software (the term open-source had not been coined yet), since there was a possibility that Python might have a broader use than for just developing utility tools for Amoeba and thus, after CWI gave permission, the first version of Python was released in February of 1991 and was distributed using the alt.sources newsgroup on Usenet. After the release Guido immediately started receiving useful and positive feedback from people who picked up Python and a routine of doing new Python releases was quickly adapted. The Python community grew and numerous organizations started depending on the language, some of which were US government agencies. These agencies wanted to help grow and stabilize the Python community and Guido got an invitation from $NIST$ (National Institute of Standards and Technology) to come to the United States for a couple of months. There he organized and hosted the first Python workshop in November of 1994, during which he met people from $CNRI$ (Corporation for National Research Initiatives). They offered him a job working on Python and evolving its community, and from 1995 to 2000 Guido worked in the US in the northern Virginia at CNRI.

The development of Python continue to this day, with the latest releases 2.7.13 and 3.6.1 [8]. Extensions to Python, in the form of modules,
packages and libraries, have been developed by the community, some of which were made for the intent of PDF generation. The most promising extension found is shortly presented below:

**ReportLab**: ReportLab is a software library for Python used to create PDF documents [11]. It exists both as an open-source version, as well as a commercial version called ReportLab Plus. The main difference between the two is that ReportLab Plus lets developers use Report Markup Language (RML), an XML dialect, for creating templates rather than working at a low level with Python coding [12]. ReportLab Plus also includes more tools than the open-source version, as well as mentoring and support from the ReportLab team. One of the contexts where ReportLab claims to be useful is when you need a ’’build system’’ for complex documents with charts, tables and text [11]. The open-source version is available under the BSD license and the latest version, version 3.4, was released in March of 2017 [13] [14]. It gets 50,000 downloads per month and is the print engine for Wikipedia [12]. Table 4 showcases a program for creating a simple table with ReportLab. When the program in Table 4 is executed with the command `python fileName.py` in the command prompt (same command on both Windows 10 and Debian
4 Requirements

The requirements posed by Biotage are highly constraining, but necessary and well motivated. In this section the requirements will be split up into two categories, namely hard and soft requirements, where the hard requirements are mandatory for the tools to fulfill and the soft requirements are additional features wanted in the tools but not necessarily needed. Moreover will the requirements be discussed in greater detail.

4.1 Hard requirements

- Dynamic and non-dependent on templates: Biotage’s current solution is based on predefined templates as a foundation. These templates are then simply filled with the data generated during a run. The downsides of using templates are that a lot of work with collaborators has to be made when information needs to be added to the report. Furthermore, should some piece of data not be necessary during a run, and thus not generated, its field in the template will still be printed out in the report, but empty. This wastes a lot of space in the reports. The static nature of templates also makes it difficult for information to ”spill over” to the next page if it does not fit on one page. Thus a dynamic solution, that generates the layout of the report given the data, is desired as it will remove these problems.

- Outcome must be PDF: The reason why the outcome must be a PDF is simply because of the universal nature of that format. The reports should be viewable on most devices and operating systems.
• Must output data as today’s reports does: Biotage’s reports today consists not only of tables with data in them, but of numerous other objects (see appendix A). The following list contains all of these objects:
  – Tables
  – Embedded images
  – Footnotes
  – Footers
  – Table headings
  – Page numbering

If no predefined templates are to be used, the tools will need to have these features in order to produce reports that are similar-looking to today’s reports.

• Must be free of charge: In order to avoid paying for a license for every instrument made by Biotage, the tools have to be free of charge.

• Must work without the Internet: Without any way of connecting to the Internet, Biotage’s instruments can only use tools that works without the use of the Internet.

• Must be able to run on both Windows and Linux: The instrument Extrahera runs a version of Linux, Debian Jessie 8.0, but demo runs shown to costumers are run on laptops using Windows. It is therefore important that the tools works on both operating systems.

4.2 Soft requirements

• Graphs: Extrahera does not use graphs in its reports, but other instruments made at Biotage does. A tool used to make a general solution, usable on all instruments, should therefore be able to draw graphs.

• Mixed page orientation (portrait/landscape) in same document: Some tables in today’s reports are to wide to fit on a portrait oriented page, and thus have to be put on a landscape oriented page. This problem might get solved by dynamically generating tables, but this feature is still desired in cases where tables gets to wide.
Disk usage: The memory capacity on the instruments is somewhat limited. It is therefore interesting to know how much memory a tool uses after installation.

Software maturity: The maturity of a software tool should be taken into consideration when evaluating the tools. If one is to incorporate a software as a third party product in a company, then one obviously wants a reliable and mature software rather than a software that might be "dead" within a few years. The level of maturity that a software tool has can be based on factors such as

- **Maintenance**: A software tool that is being maintained frequently will most likely be more reliable and mature, in the long run, than a software tool that has not seen maintenance in year

- **Community**: A software tool’s community can tell something about the maturity of the tool. A larger and more active community indicates a well-used software tool, i.e. a reliable tool

- **Documentation**: A software tool with good and educational documentation such as user guides, tutorials and manuals will most likely be a more mature tool than one that lacks this kind of documentation

- **Communication between developer and user**: Users will be those who have the best feedback when it comes to improvements to the tool and bug-reporting. A development team of a tool that offers communication with its users, discussing matters such as improvements and bugs, will most likely be a more mature tool than one that lacks this communication

Licenses: What type of license the tool is released under could be of legal interest.

Performance: How long time it takes for the tool to generate a report is of interest, if the time is long enough to affect user experience.

Standalone installation: The software tool for the laptops using Windows preferably has to be portable, in order for Biotage to provide the complete software installation as a bundle. This means that one should be able to copy the software tool from one storage device to another
without causing to software to malfunction. Otherwise, Biotage will have to install the software tool on every laptop they use.

5 Evaluation and Comparison

In this section the tools presented in the ”Background” section will be evaluated against the requirements. The results from the evaluation will be presented in a summary table at the end of this section.

5.1 Dynamic and non-dependent on templates

None of the tools examined in this thesis requires any predefined template in order to produce a PDF. As demonstrated in the background section, ÒTEX, Groff and HTMLDOC can create PDFs from simple text files (.txt), created by e.g. the main process running on an instrument, whereas ReportLab can read (with Python's `open` and `read` function) from text files in order to retrieve data and create PDFs with that data.

5.2 Outcome must be PDF

All the tools are able to produce PDFs as their formatted output, as demonstrated in the background section.

5.3 Table

As demonstrated in the ”Background” section, all tools are able to create tables.

5.4 Embedded images

5.4.1 ÒTEX

With ÒTEX, images can be embedded into the output PDF with the standard graphics package `graphics` or the extended package `graphicx`, where the format type of the images can be PDF, PNG, JPEG and MetaPost [17]. The command used to insert files is `\includegraphics{fileName}`.
5.4.2 Groff

Groff has the functionality to embed PS and PDF files into its output, with the escape sequences [5]
\texttt{\textbackslash X\textasciitilde ps: import file llx lly urx ury width [height]}’
respectively
\texttt{\textbackslash X\textasciitilde pdf: pdfpic file alignment width [height] [linelength]}’

5.4.3 HTMLDOC

HTMLDOC is limited to recognize only the \texttt{<IMG>} tag element version 1.0, and supports the formats BMP, GIF, JPEG and PNG image files [15].

5.4.4 ReportLab

ReportLab can embed images of type JPEG by default, and by installing the PIL extension to Python nearly 30 different file formats can be handled, some of which are BMP, GIF, PNG and IM (by looking in the PIL folder, one can see every format supported) [11]. By importing \texttt{Image} from \texttt{reportlab.platypus} the class \texttt{Image} becomes available, and an instance of this class can hold an image.

5.5 Footnotes

5.5.1 \LaTeX

The \texttt{\textbackslash footnote\{\}} command can be used in \LaTeX in order to create footnotes [17].

5.5.2 Groff

Groff uses the macros \texttt{.FS} and \texttt{.FE} in order to specify footnote text, and \texttt{\*[*]} to specify the location of the numbered footnote marker in the text [5].

5.5.3 HTMLDOC

HTML does not have a dedicated way to create footnotes, but footnotes can be created manually with the \texttt{<sup>} </sup> tags, which HTMLDOC
supports [15]. By writing <sup>[1]</sup> in the text where you want a footnote number mark, creating a horizontal line with the <hr> tags and writing another <sup>1. Footnote text</sup> tag pairs containing the footnote text, a visually correct footnote can be created [18]. However this footnote will not ”stick” to the bottom of the page (see appendix B for a visualization). Moreover, should the page be to full to fit the footnotes they will be ”spilled over” to the next page.

5.5.4 ReportLab

ReportLab does not seem to have a dedicated method for creating footnotes [19].

5.6 Footers

5.6.1 \LaTeX

The package \texttt{fancyhdr} can be used in \LaTeX to create both headers and footers [20]. Before the command \texttt{\begin{document}} one can specify the pagestyle with the command \texttt{\pagestyle{fancy}} and after that specify e.g. a right footer with the command \texttt{\rfoot{Footer text}}.

5.6.2 Groff

With Groff, footers can be created with the macros \texttt{.OF 'left’center’right’} and \texttt{.EF ‘left’center’right’}, defining footers for odd page numbers respectively even page numbers [5]. The arguments determine what should be in the footer and how it should be oriented.

5.6.3 HTMLDOC

HTMLDOC can create footers for the generated PDF through HTML comments in the input file [15]. Adding the comment <!—— FOOTER RIGHT ”Footer test” ——> at the top of the input file will add the text ”Footer test” in the bottom right side of every generated page.
5.6.4 ReportLab

ReportLab does not have a dedicated method for creating footers, however following the example (see appendix C) of user Juan Fco. Roco on stackoverflow.com, a ”footer” canvas class can be made and used in ReportLab’s multiBuild method to draw a footer on every page [21].

5.7 Table headings

5.7.1 \LaTeX

Table headings can be created with the package caption in \LaTeX[17]. Table 5 consists of a small example that generates a table with the table heading “Table heading”, using the \caption command.

5.7.2 Groff

In Groff, table headings can be created by giving the optional H argument to the macro .TS [5]. Text up till the macro .TH will be used as heading. Table 6 showcases an example that creates a table with the table heading
Table 6: Groff table heading example

```
.TH
.T &
| l | l | l | .
- Table heading test
- This is a Groff table example
- .TE
```

"Table heading test”.

5.7.3 HTMLDOC

The HTML tag pair `<caption>` and `</caption>` can be used inside the table tags when creating a table in order to create a header or caption for the table [22]. These tags are supported by HTMLDOC [15].

5.7.4 ReportLab

ReportLab does not seem to have a dedicated way of creating table heading, but a table heading text can be defined as a piece of table data. This data can then be spanned over a number of columns (and rows) using the style specification `SPAN, (sc, sr), (ec, er)` [11]. Table 7 consists of a small code example, showcasing a table with the text "Table heading test" as a centered, bold font table heading.
5.8 Page numbering

5.8.1 \LaTeX

Page numbering seems to be a default behavior in \LaTeX. In every test file created with \LaTeX during the evaluation, the page number has been included at the center bottom of the page.

5.8.2 Groff

In Groff’s ms macro package (the package used in every Groff example in this thesis), the default page numbering method is to print the page number, starting from the second page, at the center top of the page [23]. This method can be changed, if so desired. By writing `.ds CF \n[PN]` and `.ds CH` at the beginning of the input file, the page numbers are instead displayed at the center bottom of the page, starting from the first page.

5.8.3 HTMLDOC

HTMLDOC has the command option `−−footer lcr`, that specify the contents of the footer on every page created [15]. The lcr parameter is a three-
character string representing the left, center and right footer fields. Each of these characters can be a number of things (full description can be seen by following the reference), but the parameter combination .1. results in the page number being printed in the center bottom on every page.

5.8.4 ReportLab

Following the example in appendix C, page numbers can be created on every page with ReportLab. By making the following changes to the draw_canvas function (see table 8), the page number is displayed in the ”traditional way”, at the center bottom on every page.

5.9 Free of charge

\LaTeX, Groff, HTMLDOC and ReportLab are free of charge and can be downloaded from their respectively home page.

5.10 Works without the Internet

All tools, after downloading and installation of the software and required packages/libraries, work without the Internet. The small test files presented in the background section were used to test this property, and all tools generated the predicted outcome, i.e. PDF’s containing tables, whilst being offline.
5.11 Works on Windows

Every tool have been tested on Windows 10, with different input files varying in content. The tools have worked fine, with the exception of Groff, that was not able to generate outcome of the type PDF. After specifying PDF as the output device in the command (i.e. -Tpdf), errors emerged when executing the command. The first error was "’perl’ is not recognized as an internal or external command, operable program or batch file.", after which an empty and damaged PDF file was created. To resolve this error, Perl was downloaded and installed, and after executing the command again the following error emerged:
"Failed to open ’download’
troff: fatal error: error writing output file”, after which an empty and damaged PDF file was created once again. The only way the author of this thesis managed to create PDF files on Windows 10 with Groff was to either install GhostScript and create a PS file (.ps) with Groff, which could then be converted to a PDF with GhostScript’s ps2pdf function, or to install \LaTeX and create a DVI file (.dvi) with Groff, which could then be converted to a PDF with \LaTeX’s dvipdfm function. The following two commands demonstrate the complete series of Windows 10 commands that was used when creating PDFs with ps2pdf respectively dvipdfm:

groff -ms -t -Tps inputFile.txt > outputFile.ps & ps2pdf outputFile.ps

groff -ms -t -Tdvi inputFile.txt > outputFile.dvi & dvipdfm outputFile.dvi

5.12 Works on Linux

The tests that was performed on every tool in Windows 10 were also performed in Debian 8.8, and every tool generated the same output on both operating systems. Groff was able to produce PDFs in Debian 8.8.
5.13 Graphs

5.13.1 \LaTeX

The package \texttt{pgfplots} is a \LaTeX package used for creating plots and labeled axes [17]. It is capable of drawing normal, log and semi-log plots in both two and three dimensions. It also has the functionality to read coordinates from separate data files, and draw plots from these.

Table 9 and figure 3 shows a small example of a plot created with the package.

5.13.2 Groff

The preprocessor \texttt{grap} can be used in Groff in order to plot graphs [24]. By writing the command option \texttt{−G} when executing a Groff command, Groff is told to use \texttt{grap} as a preprocessor. Table 10 contains a small example of a
Grap was not included in the Windows 10 Groff installation used in this thesis, but could be downloaded and installed separately (see appendix G.1 for an installation guide). After installation of grap, using the example in table 10 along with the command `groff −ms −G −Tps input.txt > output.ps` creates a PS output file, containing the graph seen in figure 4.

5.13.3 HTMLDOC

The only way to create graphs in HTML seems to be by using some kind of JavaScript tool, such as JavaScript Stock Chart [25]. However, HTMLDOC does not support scripting [15].

5.13.4 ReportLab

ReportLab has the ability to create many different kinds of graphs, including bar charts, line charts, line plots and pie charts [11]. Table 11 contains code for a small program that creates a line plot graph, which can be seen in figure 5.
Table 10: Grap graph program

```plaintext
.G1
label top "Test graph"
label left "y val"
label bot "x val"
ticks left in from 1 to 5 by 1
ticks bot in from 1 to 5 by 1
draw solid
0.2 0.5
0.5 0.7
1.2 0.9
1.8 1.1
2.1 1.4
2.7 3.5
.G2
```

Figure 4: Graph drawn with grap
Table 11: ReportLab graph program

```python
from reportlab.graphics.shapes import Drawing
from reportlab.graphics.charts.lineplots import LinePlot
from reportlab.lib.pagesizes import A4
from reportlab.lib import colors
from reportlab.platypus import SimpleDocTemplate
from reportlab.lib.styles import getSampleStyleSheet

styles = getSampleStyleSheet()

doc = SimpleDocTemplate("graph.pdf",
                          pagesize=A4)

elements = []

drawing = Drawing(400,200)

linePlotData = [((0.2,0.5), (0.5,0.7),
                 (1.2,0.9), (1.8,1.1),
                 (2.1,1.4), (2.7,3.5))]

lp = LinePlot()
lp.x = 50
lp.y = 50
lp.height = 125
lp.width = 300
lp.data = linePlotData
lp.joinedLines = 1
lp.strokeColor = colors.black
lp.xValueAxis.valueMin = 0
lp.xValueAxis.valueMax = 5
lp.yValueAxis.valueMin = 0
lp.yValueAxis.valueMax = 5
lp.xValueAxis.valueSteps = [0,1,2,3,4,5]
lp.yValueAxis.valueSteps = [0,1,2,3,4,5]
drawing.add(lp)

elements.append(drawing)

doc.build(elements)
```
5.14 Mixed page orientation (portrait/landscape) in same document

5.14.1 \LaTeX

The package `pdflscape` can be used to orient pages into landscape orientation [26]. A landscape oriented page is initiated and terminated with the `\begin{landscape}` respectively `\end{landscape}` commands. Text and objects within these two commands will be written on a landscape oriented page, whereas text and objects before and after these commands will be written on a portrait oriented page. This method does not adjust page numbers or footers however; the result can be seen in appendix D. By adding the command `\thispagestyle{empty}` within the `\begin{landscape}` and `\end{landscape}` commands, one can turn off page numbering and footers for that specific page.

5.14.2 Groff

In Groff, page orientation can be changed with command line options, by passing arguments to the postprocessor [5]. The \texttt{-l} option is used to denote landscape orientation. The following command is from Groff’s manual, and is used to output a PS file of paper size A4 in landscape orientation:

```
groff -Tps -dpaper=a4l -P -pa4 -P -l -ms inputFile.txt > outputFile.ps
```

Notice that every page will be landscape oriented, and not just one specific page. To the authors knowledge, there exists no dedicated method
of mixing orientations within the same document.

5.14.3 HTMLDOC

With HTMLDOC, one can either initiate landscape orientation with a command option, or with HTML comments [15]. The command option \texttt{--landscape} will change the page orientation to landscape for every page generated, whereas the HTML comment \texttt{<!-- MEDIA LANDSCAPE YES -->} changes the orientation to landscape for the current page, breaking to a new page if the current page is already marked (changing the "YES" into a "NO" in this comment will change the page orientation back to portrait). The latter method can be combined with the HTML comment \texttt{<!-- NEW PAGE -->} for better control over which pages get their orientation changed. Table 12 displays a small example where one page out of three get its orientation changed.

5.14.4 ReportLab

Similar to HTMLDOC, ReportLab has two ways of changing page orientation. One can define landscape orientation for every page using the \texttt{BaseDocTemplate} class. One of the arguments its constructor take is \texttt{pagesize},

Table 12: HTMLDOC page orientation example

```html
<!DOCTYPE HTML PUBLIC "−//W3C//DTD HTML 3.2 Final//EN">
<html>
<head></head>
<body>
<p>This page is in portrait orientation</p>
<!-- NEW PAGE -->
<!-- MEDIA LANDSCAPE YES -->
<p>This page is in landscape orientation</p>
<!-- NEW PAGE -->
<!-- MEDIA LANDSCAPE NO -->
<p>Back to portrait orientation</p>
</body>
</html>
```
specifying the dimensions of the generated PDF. Using the function `landscape()`, the user can define landscape oriented dimensions for the generated PDF. Below is a small code snippet, displaying how a landscape oriented A4 PDF can be created:

```python
doc = BaseDocTemplate("outputFileName.pdf", pagesize=landscape(A4))
```

If a mix of landscape and portrait oriented pages is desired, then one can declare instances of the class `PageTemplate` along with instances of the class `Frame`. These templates can be added to the `BaseDocTemplate` instance with the function `addPageTemplates()`, and to the list of elements to be drawn on the document one can append the return value of the function `NextPageTemplate()`. This function ensures that a new template, the one specified as its argument, is used on the next page generated. An example program is listed in appendix E, which is based on both the program in appendix C and the example of user Kyle Heuton on stackoverflow.com (see reference) [27].

## 5.15 Disk usage

An installation guide for every tool is presented in appendix F to I, to justify where the numbers presented in the coming subsections are retrieved from. The installation guides will cover both an installation on Windows 10 and an installation on Debian 8.8 being run on a virtual machine using the software `VirtualBox`. The Debian 8.8 installation guides were used to install every tool on one of Biotage’s Extrahera PC-console, running an operating system based on Debian Jessie 8.0. For some tools, additional steps had to be taken in order to get a working installation on Extrahera. These steps are included in the guides.

### 5.15.1 \LaTeX

Windows:

Disk usage after installation: 938 MB

Disk usage after installation of additional files and packages: 973 MB
Debian:
Disc usage after installation: Approximately 1.1 GB

5.15.2 Groff

Windows:
Disc usage: 29.9 MB
Disc usage after installation of grap: 32.1 MB

Debian:
Disc usage after installation of Groff and grap: 10.78 MB

5.15.3 HTMLDOC

Windows:
Disk usage after installation: 15 MB

Debian:
Disk usage after installation: 9.697 MB

5.15.4 ReportLab

Windows:
Disc usage after installation of Python and ReportLab: 78.3 MB

Debian:
Disc usage after installation of Python, dependency files and ReportLab: Approximately 200 MB

5.16 Software maturity

5.16.1 MiKTeX

MiKTeX, a Windows TeX distribution used on the Windows 10 side of this thesis, had its latest update (as of writing) in 2017-08-27 with the version 2.9.6448 [28]. TeX Live, a TeX system with binaries for most Unix
systems, including GNU/Linux had its latest update (as of writing) in 2017-06-04 [29]. The \LaTeX{} community is big, with a user base that goes into the millions [33], and on their website they have a page specifically for links to contributed documentation, FAQ’s, Question and Answer Websites, tutorials, user groups and news groups [34]. From this information, the author of this thesis concludes that \LaTeX{}’s level of maturity is high.

5.16.2 Groff

Groff had its latest update (as of writing) of version 1.22.3 in 2014-11-04 [5]. The binary file for windows, used on the Windows 10 side of this thesis, was latest modified (as of writing) in 2015-05-09 [30]. On Groff’s home page, one can find links to both Groff’s manual, and links to different mailing lists, including a general discussion mailing list, a bug report mailing list, and a read-only mailing list showing changes to the git repository [6]. The general discussion mailing list reveals that the Groff community is quite active, showing the most recent thread (as of writing) to be from 2017-09-06 [35]. However, there is a lack of resources and learning materials for how to use Groff, which makes the learning process more difficult. From this information, the author of this thesis concludes that Groff’s level of maturity is moderate.

5.16.3 HTMLDOC

HTMLDOC had its latest update (as of writing) in 2017-07-05 of version 1.9 [31]. The software is included as a part of the Debian OS [36]. During the year of 2017, there have been four releases of the software, but prior to that the latest release was in August of 2006 [31]. As of writing, there are nine open issues being discussed between the developer and users. The developer of the software, Michael R Sweet, maintains HTMLDOC during his spare time [15]. The HTMLDOC manual explains every feature of the software in an easy-to-follow manner. From this information, the author of this thesis concludes that HTMLDOC’s level of maturity is high.
5.16.4 ReportLab

ReportLab had its latest update (as of writing) in 2017-06-23 of version 3.4.14 [32]. The software is being used by a number of clients, including Wikipedia, Newcastle University, Oxfam and Hewlett Packard [37]. The community is active and there exists a mailing list, monitored by ReportLab’s development team, that is suitable for asking questions about the software [38]. As of writing, there are 45 open issues being discussed between the development team and users of the software [32]. On ReportLab’s website, one can find documentation about the software, such as user guides, developer FAQ and a link to their mailing list archives [39]. From this information, the author of this thesis concludes that ReportLab’s level of maturity is high.

5.17 License

5.17.1 \LaTeX

\LaTeX{} is released under the \LaTeX{} Project Public License (LPPL) [2]. It is based on Donald Knuth’s original license for TeX, which states that the source code for TeX may be used for any purpose, but one is only allowed to call a system built with it ”TeX” if it strictly conforms to his canonical program [40]. The LPPL attempts to preserve the fact that if one writes a \LaTeX{} document and send it to someone else, you expect it to work at their end just like it does at your end. The \LaTeX{} Project does not necessarily hold the copyright for a work released under the LPPL; it is the author of the work that holds the copyright for it and the responsibility for enforcing any violations of the license. The license is DFSG (Debian Free Software Guidelines) compatible, FSF (Free Software Foundation) approved, OSI (Open Source Initiative) approved and it allows linking from code with a different license. However, the license is not GPL (General Public License) compatible, and it is not copylefted, which means that any derivative work does not have to be distributed under the same license [41].

5.17.2 Groff

Groff is released under the GNU General Public License version 3 (GNU GPL v3) [6]. It is a free software license and it guarantees the end user the freedom to run, study, share and modify the software [41]. The license gives
permission to modify any work released under the license, as well as copy and redistribute the work or any derivative version. It places not restrictions on commercial use, and GPL works may be sold at any price. The license states that programs distributed as pre-compiled binaries has to be accompanied by a copy of the source code, a written offer to distribute the source code via the same mechanism as the pre-compiled binary, or a written offer to obtain the source code that the user got when they received the pre-compiled binary under the GPL [41]. The license also requires giving all recipients a copy of the license along with the program. The license is DFSG compatible, FSF approved, OSI approved and copylefted, meaning that any derivative work can only be distributed under the same license. However, one is not allowed linking from code with a different license, except for software licensed under GLPv3 compatible licenses. There exists a few opinions regarding linking and how linking interacts with the GPL. The key dispute is whether non-GPL software can legally statically link or dynamically link to GPL libraries. The FSF asserts that an executable which uses a dynamically linked library is a derivative work, and thus needs to be released under the GPL license. This however does not apply to separate programs communicating with one another, through e.g. command-line arguments. Other argues that while static linking produces derivative work, it is not clear whether dynamical linking to GPL code should be considered a derivative work. Some argues that the method of linking is mostly irrelevant to question about whether a piece software is a derivative work. Lawrence Rosen (a one-time Open Source Initiative general counsel) argues that the more important question is whether the software was intended to interface with the client software and/or libraries. He states that if a software uses the source code of the original program in a copy-paste sense, modified, translated or otherwise changed way, then this new software should indeed be considered a derivative work.

Bundling separate programs together and distributing them together on the same CD-ROM or other media creates what is knows as a aggregate, and the following is a quote from gnu.org GPL FAQ, describing to what extent a software is allowed to communicate with and be-bundled-with GPL programs [41]: "The GPL permits you to create and distribute an aggregate, even when the licenses of the other software are non-free or GPL-incompatible. The only condition is that you cannot release the aggregate under a license that prohibits users from exercising rights that each
program’s individual license would grant them.”

5.17.3 HTMLDOC

HTMLDOC is released under the GNU General Public License version 2 (GNU GPL v2) [15]. In the license the following is stated: “GNUGPLv2, or (at your option) any later version”. This allows upgrading to version 3 of the license. According to Richard Stallman, the original writer of the license, the most important changes made in version 3 of the license (i.e. changes that are not present in version 2 of the license) are in relation to software patents, free software compatibility, the definition of source code, and hardware restrictions on software modifications [41]. GNU GPL v2 is not compatible with several open source licenses such as Apache License and GNU Affero General Public License, which version 3 is compatible with.

5.17.4 ReportLab

ReportLab is released under the BSD license, and it is stated on ReportLab’s website that “...you can basically do anything you want with it, including using it in a commercial package” [14]. By looking in the LICENSE.txt file that one gets when downloading and installing ReportLab, one can see that the license appears to be the 2-clause BSD license (or FreeBSD). This license is DFSG compatible, FSF approved, OSI approved, GPL compatible, and linking from code with a different license is allowed. The license is not copylefted, which means that any derivative work does not have to be distributed under the same license [42]. The license is short and simple, and can be seen in figure 6.

5.18 Performance

In order to test the performance of the tools, performance tests were written and executed on both a laptop using Windows 10 and an Extrahera PC-console whose operating system is based on Debian Jessie 8.0. Since the tools do not share every feature these programs differed a little bit, but every test program contained equally many pages (seven pages), a right header, a right footer and page numbering, equally many sections (seven sections), tables containing equal data, text containing equal data, and embedded images.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

These test programs were then timed, with the Unix `time` command issued on the computer used by Extrahera, and with the `timeit` command (part of the Windows Server 2003 Resource Kit Tools) on Windows 10. The results will be presented in the summary table as an average between the elapsed time on both operating systems (i.e. \((\text{Elapsed Time} + \text{Real Time})/2\)).

The hardware used for the Windows 10 performance tests were:

- **Processor**: Intel® Celeron N2840
- **RAM**: 4 GB

The hardware used for the Extrahera performance tests were:

- **Processor**: AMD G-T40E
- **RAM**: 4 GB

5.18.1 \LaTeX

Windows 10:

- Elapsed Time: 0:00:09.490 s
5.18.1 Groff

Windows 10:
- Elapsed Time: 0:00:02.007 s

Extrahera:
- Real Time: 0:00:00.579 s

5.18.2 HTMLDOC

Windows 10:
- Elapsed Time: 0:00:00.658 s

Extrahera:
- Real Time: 0:00:00.063 s

5.18.3 ReportLab

Windows 10:
- Elapsed Time: 0:00:01.606 s

Extrahera:
- Real Time: 0:00:00.698 s

5.19 Standalone installation

5.19.1 \LaTeX

By downloading MiKTeX portable edition, from https://miktex.org/portable, one can install MiKTeX on a portable storage device and run \LaTeX on any Windows computer. The installed folder can also be copied to other devices and still work as intended.
5.19.2 Groff

After downloading Groff and installing grap, the folder containing the program can be copied to a portable storage device in order to run Groff on any Windows computer.

5.19.3 HTMLDOC

It is not possible to copy the HTMLDOC folder on a portable storage device and run it on another Windows computer. Files are hardcoded to a specific path during installation. When trying to copy the HTMLDOC folder to an USB, and run the program on a different computer, the author of this thesis got the following error: "ERR005: Unable to open font file C:/Program Files/HTMLDOC/font/...". This problem can presumably be fixed by modifying the source code of HTMLDOC, but the author of this thesis decided not to do this.

5.19.4 ReportLab

After installing Python 2.7, the folder containing the program can be copied to a portable storage device in order to run python programs on any Windows computer.
## 5.20 Summary

The following table presents a summary of the evaluation:

<table>
<thead>
<tr>
<th>Hard requirements</th>
<th>( \LaTeX )</th>
<th>Groff</th>
<th>HTMLDOC</th>
<th>ReportLab</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Outcome must be PDF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Table</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Embedded images</td>
<td>PDF, PNG, JPEG, MetaPost</td>
<td>PS, PDF</td>
<td>BMP, GIF, JPEG, PNG</td>
<td>JPEG, PNG, BMP, GIF, IM etc.</td>
</tr>
<tr>
<td>Footnotes</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Footers</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Table headings</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Page numbering</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Free of charge</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Works without the Internet</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Works on Windows</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Works on Linux</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### Soft requirements

| Graphs                | ✓            | ✓     | ×       | ✓         |
| Mixed page orientation | ✓            | ×     | ✓       | ✓         |
| Disk usage (Windows)  | 973 MB       | 32.1 MB | 15MB    | 78.3 MB   |
| Disk usage (Debian)   | 1.1 GB       | 10.78 MB | 9.697 MB | 200 MB    |
| Software maturity     | High         | Moderate | High    | High      |
| License               | LPPL         | GNU GPL v3 | GNU GPL v2 | BSD      |
| Performance           | 6.302 s      | 1.293 s | 0.361 s | 1.152 s   |
| Standalone installa-  | Yes          | Yes   | No      | Yes       |

---

1. Does not stick to the bottom of the page, see appendix B
2. Workarounds possible with additional installations
6 Related Work

During the search for prior work, two studies were found that are related to the study described in this thesis. In this section, these two studies are described in detail.

In a 2010 study, Yang and Wagner [3] described the problem that libraries faces; successfully compete with big companies like Amazon and Google, that implements next-generation functionalities, or be replaced by them. One way for libraries to successfully compete against the big commercial companies is to replace their current OPAC in favor of one of the new standalone, next-generation discovery tool. Yang and Wagner continues the study by presenting 12 commonly acknowledged features for next-generation catalogs, some of these features include ”single point of entry for all library information”, ”state-of-the-art web interface” and ”enriched content”. They then identify major open source and proprietary discovery tools used at library sites around the world, some of which are ”Blacklight”, ”SOPAC” and ”Aquabrowser”. In total they identify 7 open source and 13 proprietary discovery tools, which they then proceed to evaluate and compare under the features previously presented. They summarize and present their findings with tables, where each discovery tool is checked for whether or not it has (marked with a ✓ and an x respectively) a feature. They concluded that open source discovery tools had taken the lead in progress towards a next-generation goal, but also that none of the tools evaluated in the study could be truly called next-generation, since no tool had all 12 features that characterize next-generation catalogs.

In 2011, a study made by Shahid and Ibrahim [4] describes the essentiality of software testing to determine and improve the quality of software. Organizations not only implement software testing, but also seek to measure the completeness and goodness of such tests. This can be done with test coverage tools. Shahid and Ibrahim continues their study by listing and presenting test coverage tools, amounting to a total of 22 tools, including e.g. ”JavaCodeCoverage”, ”Jester” and ”eXVantage”. They then continue to present 5 evaluation criteria; language support, instrumentation, coverage measurement and GUI and reporting. The tools were then evaluated under these criterias (3 tools were excluded due to lack of activity and maintenance), and the results were presented in a table (see figure 7), where a tool was marked with a × if it fulfilled a criteria and was left unmarked if it did
not. They concluded that each tool had its strong and weak points and that users and developers should pick a tool according to their need, and hoped that their table will help in this choice.

7 Discussion of the Results

In this section the results from the evaluation and comparison will be discussed. The pros and cons for every tool will be presented, followed by a motivated conclusion on what tool I recommend for Biotage.
7.1 \LaTeX

7.1.1 Pros

From the evaluation one can see that \LaTeX was the only tool that completely fulfilled all requirements, without any workarounds. The tool is well maintained and used by millions, and there exists a lot of documentation and tutorials on how to use the tool for specific things.

The syntax for the \LaTeX programming language is intuitive and easy to understand, and thus easy to learn, and the shell commands used to execute \LaTeX programs are very small (the only command used in this thesis was `\texttt{pdflatex fileName.txt}`), which means that no complicated combination of commands and command options has to be remembered to use this tool.

With \LaTeX one can mix landscape and portrait oriented pages in a single run, without having to first generate portrait oriented pages and then landscape oriented pages to finally merge them together.

On Windows it is possible run the program from a portable storage device, and the installed program along with its files can be copied from one storage device to another without causing the program to malfunction.

7.1.2 Cons

Out of all tools examined, \LaTeX requires the most space on the disc after installation. It also has the worst performance out of all the tools.

When new functionalities are desired in the generated output of \LaTeX, one has to download and install additional files and packages, if that new functionality is not already supported by the current installation of the program. This, of course, requires internet connection.

7.2 Groff

7.2.1 Pros

Groff is comparably small to the other tools examined in this thesis, with the second to lowest disc usage after installation. It is similar to \LaTeX in that it is meant to be used for creating scientific reports and articles of good quality. Its performance is very good, both on Windows and Debian.
Just like \LaTeX, one can copy Groff and its files on Windows from one storage device to another without causing the program to malfunction.

### 7.2.2 Cons

Groff did not fulfill every requirements completely, for one had to install additional software to work around its problems. It was not possible to create PDF’s on Windows 10 with Groff without installing either GhostScript or \LaTeX.

The only types of images Groff is able to include in its output are PS and PDF files, which is not commonly an image type format. Furthermore, I found it difficult to get the macros for embedding images to generate the desired output, due to a less educational description of the macros in the user manual. In general, Groff lacked educational documentation and tutorials on how to use the tool.

A mix of landscape and portrait oriented pages can not be achieved in a single run with Groff. If a mix of the two orientations is desired in the output one has to merge two documents of different orientation together.

The only documentation reference one can find on Groff’s homepage is a link to its manual (which in my opinion was not all to educational), and a link to mailing archives. Without proper documentation and tutorials, it becomes difficult to learn how to use the tool.

The syntax for the Groff programming language consists of macros on the form .XX (i.e. a dot followed by a two letters contraction). Once learned these macros can be easy to understand, but for a new user it might be difficult read and understand a Groff file (one can take the macros .TS and .TE as an example). Furthermore, the shell commands used to execute a Groff program can become quite complicated when a lot of command options are required.

### 7.3 HTMLDOC

#### 7.3.1 Pros

Just like Groff, HTMLDOC is very small program, using the least amount of memory out of all examined tools after installation. Its performance is also the best out of all the tools.
The program uses HTML files in order to generate PDFs, a well documented programming language with a lot of tutorials and guides. The program itself is also well documented with an easy-to-understand manual. It was also easy to install, both on Windows and Debian.

Since HTML is used for creating web pages, it is missing features that one would like in a PDF file, such as headers and footers, but HTMLDOC complements these missing features with its own functionality (such as HTML-comments and command options).

A mix of landscape and portrait oriented pages can be achieved with HTMLDOC in a single run, using its HTML-comments functionality.

7.3.2 Cons

HTMLDOC does not support footnotes in a desired manner, nor does it support graphs. It is also limited to what HTML features it can support, with features such as HTML 4.0 elements, attributes, stylesheets and scripting not being supported.

HTMLDOC is not portable since files seems to be hardcoded to a specific path during installation. This means that one can not provide the complete software installation of HTMLDOC as a bundle to Windows laptops.

7.4 ReportLab

7.4.1 Pros

Since ReportLab is a Python package, the high level functionalities of Python as a programming language is used when generating PDFs with ReportLab. It does not take up a lot of space on the disc after installation and its performance is good.

ReportLab generates output of high quality, similar to \LaTeX\ and Groff, and is very much suitable for generating reports. It supports a vast variety of image type formats to be included in its output.

A mix of landscape and portrait oriented pages can be achieved with ReportLab in a single run.

ReportLab has a high quality documentation and user guide. The
user community is large and one can often find help either from the development team or other users.

Commands to execute Python programs are easy and short (the only one used in this thesis was \texttt{python fileName.py}), which means that no complicated combination of commands and command options has to be remembered to use this tool.

Python along with ReportLab is portable, meaning that one can run ReportLab from a portable storage device and copy the program from one storage device to another without causing the program to malfunction.

7.4.2 Cons

Since ReportLab is a Python package, some knowledge in the Python programming language is required to use the tool, and the learning curve is in my opinion quite high.

ReportLab does not have a "dedicated" method of creating many things, and the user often has to use creativity in order to "invent" solutions. Comparing footers in \LaTeX{} and ReportLab for example, I hope to clarify what I mean with "dedicated". In \LaTeX{}, one can easily create footers with the command \texttt{\backslash rfoot\{Footer text\}}, whereas in ReportLab one has to create a complicated class and functions (see appendix C) in order to do this. This might be the case for footnotes; there might be a creative solutions in order to create footnotes with ReportLab, but I did not find a way to do this.

7.5 Conclusion

To conclude this evaluation and comparison, I would strongly recommend \LaTeX{} as the PDF generating tool for Biotage to incorporate in their instruments and laptops. The reasons for this are that \LaTeX{} has a high amount of functionalities and fulfills every requirement Biotage has on their reports. Furthermore, the tool is easy to use easy to learn, with an intuitive programming language and a lot of high quality documentation and tutorials. It works without any troubles on both Windows and Debian, and \LaTeX{} files written does not differ in any way between the two operating systems. The downsides of using \LaTeX{} are in my opinion negligible; the process times from the performance tests are not long enough to affect the user in a negative
way. The large amount of space on the disc required by the tool is in today’s standards not that much of a problem, and the fact that internet connection is required when new packages, providing new functionalities, are desired can be somewhat solved by planning ahead, and installing every package that one needs now and thinks one will need in the future.

I should also mention that ReportLab is a promising option. Would it not be for the fact that ReportLab along with Python have such a high learning curve, and the fact that footnotes could not be created with ReportLab, it might be able to compete with \LaTeX. And the fact that you are writing a Python program when using ReportLab might have some benefits when it is incorporated with other applications (such as Java). Speculations like these can not be confirmed in this thesis though, since they are out of the scope of the work.

8 Reflection and Future Work

In this section a reflection of this thesis will be presented, where I discuss my methods and ways-of-working and what I could have done better. Lastly a discussion on what future work this thesis might create will be presented.

8.1 Reflection

When I started this study, I was not only meant to find, evaluate and compare different software tools that could generate PDF files, but I was also meant to design a proof-of-concept or a prototype for Biotage that could later be used for developing a general solution for their instruments. As work progressed however, both my supervisor Alexander Bogardi and I realized that it would not be possible to design something within the time frame of the work. We furthermore concluded that designing a prototype was of little academic interest, since the interesting problem and the central part of this study was to find a tool that suited Biotage’s needs, and motivate why this tool suited Biotage above other tools. The design part of the study was therefore dropped, and more focus was spent on evaluation and comparison.

I think only focusing on evaluating and comparing the tools really helped improve the results of this study. This gave me time learn how to use the tools and made the evaluation more fair. This becomes even more
important given the fact that I had experience in both \LaTeX and Python prior to this work, which could have made the comparison (and subsequently the results) unfair without proper time to learn Groff and HTMLDOC. The problem itself was solved well because the results are backed up by both literature and practical testing, and I do not think I could have reached another result by doing anything differently.

The methods I used to achieve the results was inspired by other studies, as stated in the "Method" section. The task of examining software tools to see what they can and can not do, how they perform, and other attributes around the software is a fairly concrete task, and thus the methods for solving this task is "straight-forward". By researching literature, manuals and user guides on these software tools one can in most cases quickly find information that solves this task. In the case of this study, since the tools examined are used to produce something concrete, the information from literatures could be backed up by practical testing, which in my opinion was a nice addition to the evaluation. With tests, claims can be confirmed and they allow the reader to recreate them. The problem of this method however is when there is not much information to find about a tool in literatures, or when the information is difficult to find and understand. I ran into this problem a lot when examining Groff. I could not find any books about Groff and the only manual I found was difficult to understand. This problem likely affected my results, both in the way that it became difficult to evaluate Groff, but also in the way that became less fun and more frustrating to learn how to use Groff. This was obviously a clear problem with my methods, but it is also a difficult problem to solve, especially when working within a time frame. My recommendations to minimize this problem, both for myself and for anyone else doing a similar work, is to start the literature research well before the actual work starts. Make sure that you have a good and educational source of information about every tool that is to be examined before the evaluation begins.

Another problem, related to the one just described, is that it for some tools was difficult to jump to a specific topic and learn about it without reading previous topics and chapters. For \LaTeX and HTMLDOC the topic of say mixed page orientation was easy to jump to and to learn without having to visit previous topics and chapters, but for Groff and ReportLab this method was not as successful. Thus, in the case of Groff and ReportLab, it would have been better if I took my time to read the manuals and user
guides from start to finish to ensure that I could present every topic to its full potential. Only then can I say that the evaluation and comparison was completely fair. However, reading an advanced software manual or user guide from start to finish takes a lot of time, and it was difficult to fit this task within the time frame that I worked in without having to make sacrifices in other areas of the thesis.

8.2 Future Work

The most interesting future work that this thesis opens the doors to is to implement a prototype for Biotage that works on their instruments. This work would confirm whether the conclusions made in this thesis are practical or just theoretical.

Another future work would be to extend the evaluation and comparison made in this thesis. There might be other tools for PDF generation that would be a good candidate for Biotage, and thus needs to be evaluated and compared to the tools presented in this thesis. There might also be features and functionalities in the tools presented in this thesis that I did not cover or justify well enough, that might change the results and conclusions made in this thesis. Since there was a limited time for me to learn and evaluate these tools, this could well enough be the case. Thus a more in-depth evaluation of the tools might be a future work.
References


Appendices

A Today’s reports

---

**Biotage® Extrahera™**

**SPE Method Report**

<table>
<thead>
<tr>
<th>Execution Status</th>
<th>Success</th>
</tr>
</thead>
</table>

**Method Name**

EVOLUTE AIN/EXPRESS (acid, base, neutral)

**Run Identifiers**

<table>
<thead>
<tr>
<th>Sample Plate/Rack ID</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extraction Media Lot No.</td>
<td></td>
</tr>
<tr>
<td>Collection Plate/Rack ID</td>
<td></td>
</tr>
</tbody>
</table>

**Run Details**

<table>
<thead>
<tr>
<th>Run date</th>
<th>2016-10-23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run time</td>
<td>14:45</td>
</tr>
</tbody>
</table>

**Method Details**

<table>
<thead>
<tr>
<th>Process type</th>
<th>Steps</th>
<th>Solvents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preluent</td>
<td>1</td>
<td>2% Formic acid</td>
</tr>
<tr>
<td>Load</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Wash</td>
<td>1</td>
<td>Water/MeOH (BILS)</td>
</tr>
<tr>
<td>Elution</td>
<td>1</td>
<td>Methanol</td>
</tr>
</tbody>
</table>

**Starting sample volume in plate/rack (µL)**

<table>
<thead>
<tr>
<th>Sample type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aqueous Sample</td>
</tr>
</tbody>
</table>

**Comments**

---

Page 1 (1)
B  HTML footnote example
C  ReportLab footer

```python
from reportlab.pdfgen import canvas
from reportlab.platypus import (SimpleDocTemplate, Paragraph, PageBreak)
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib.pagesizes import A4

class FooterCanvas(canvas.Canvas):
    def __init__(self, *args, **kwargs):
        canvas.Canvas.__init__(self, *args, **kwargs)
        self.pages = []
    def showPage(self):
        self.pages.append(dict(self.__dict__))
        self._startPage()
    def save(self):
        page_count = len(self.pages)
        for page in self.pages:
            self.__dict__.update(page)
            self.draw_canvas(page_count)
        canvas.Canvas.showPage(self)
        canvas.Canvas.save(self)
    def draw_canvas(self, page_count):
        page = "Page %s of %s" % (self._pageNumber, page_count)
        x = 128
        self.saveState()
        self.setStrokeColorRGB(0, 0, 0)
        self.setLineWidth(0.5)
        self.line(66, 78, A4[0] - 66, 78)
        self.setFont('Times-Roman', 10)
        self.drawString(A4[0]-x, 65, page)
        self.restoreState()

styles = getSampleStyleSheet()
elements = []
elements.append(Paragraph("Hello", styles["Normal"]))
elements.append(Paragraph("World", styles["Normal"]))
elements.append(PageBreak())
elements.append(Paragraph("You are in page 2", styles["Normal"]))
doc = SimpleDocTemplate("my_file.pdf", pagesize=A4)
doc.multiBuild(elements, canvasmaker=FooterCanvas)
```
D \LaTeX{} landscape orientation

This page is in landscape orientation
E ReportLab page orientation

```python
from reportlab.lib.pagesizes import A4, landscape
from reportlab.platypus import (Paragraph, PageBreak,
    PageTemplate, NextPageTemplate,
    BaseDocTemplate, Frame)

from reportlab.pdfgen import canvas
from reportlab.lib.styles import getSampleStyleSheet

# Function that sets the pagesize to portrait oriented A4
def make_portrait(canvas, doc):
    canvas.setPageSize(A4)

# Function that sets the pagesize to landscape oriented A4
def make_landscape(canvas, doc):
    canvas.setPageSize(landscape(A4))

# Class for adding pagunumbering and footers
# Kept in this example only to show that page numbers
# and footers gets adjusted when page orientation is changed
class FooterCanvas(canvas.Canvas):
    def __init__(self, *args, **kwargs):
        canvas.Canvas.__init__(self, *args, **kwargs)
        self.pages = []
    def showPage(self):
        self.pages.append(dict(self.__dict__))
        self._startPage()
    def save(self):
        page_count = len(self.pages)
        for page in self.pages:
            self.__dict__.update(page)
            self.draw_canvas(page_count)
        canvas.Canvas.showPage(self)
        canvas.Canvas.save(self)
    def draw_canvas(self, page_count):
        page = '%s' % (self._pageNumber)
        x = 128
        self.saveState()
        self.setStrokeColorRGB(0, 0, 0)
        self.setLineWidth(0.5)
        self.line(66, 78, self._pagesize[0] - 66, 78)
        self.setFont('Times-Roman', 10)
        self.drawString(self._pagesize[0]/2, 65, page)
        self.drawString(x, 65, page1)
        self.restoreState()

styles = getSampleStyleSheet()
```
doc = BaseDocTemplate("outputFile.pdf",
    pagesize=A4, showBoundary=0)

elements = []

# Define a frame for portrait oriented pages
framePortrait = Frame(doc.leftMargin, doc.topMargin,
    doc.width, doc.height,
    leftPadding = 0, rightPadding = 0,
    topPadding = 0, bottomPadding = 0,
    id='framePortrait')

# Define a frame for landscape oriented pages
frameLandscape = Frame(doc.topMargin, doc.leftMargin,
    doc.height, doc.width,
    leftPadding = 0, rightPadding = 0,
    topPadding = 0, bottomPadding = 0,
    id='frameLandscape')

# Define templates for both page orientations. On page, call their resp. functions
ptemplate = PageTemplate(id='portrait', frames = [framePortrait],
onPage=make_portrait)
ltemplate = PageTemplate(id='landscape', frames = [frameLandscape],
onPage=make_landscape)

# Add the templates to doc
doc.addPageTemplates([ptemplate, ltemplate])

elements.append(Paragraph("This page is portrait oriented",
    styles["Normal"]))

elements.append(NextPageTemplate('landscape'))

elements.append(PageBreak())

elements.append(Paragraph("This page is landscape oriented",
    styles["Normal"]))

elements.append(NextPageTemplate('portrait'))

elements.append(PageBreak())

elements.append(Paragraph("Back to portrait oriented",
    styles["Normal"]))

doc.multiBuild(elements, canvasmaker=FooterCanvas)

del elements
F  \LaTeX installation guide

F.1 Windows 10

1. Download basic-miktex-2.9.6361-x64.exe (size 195.718 MB) from https://miktex.org/download

2. Run the downloaded file

3. Follow the installation guide, after the installation the program takes up 938 MB on the disc (checked by viewing the properties of the MiKTeX 2.9 folder)

4. MiKTeX installs files and packages when it detects that something is missing. If the test programs and commands presented in this thesis are executed the following files and packages will have be installed, in order for the programs to work:

- **File**: tex\latex\arabi\bblopts.cfg
  - Part of package: arabi
  - Size: 3943.043 KB

- **File**: tex\latex\fancyhdr\fancyhdr.sty
  - Part of package: fancyhdr
  - Size: 407.288 KB

- **File**: tex\latex\caption\caption.sty
  - Part of package: caption
  - Size: 1920.856 KB

- **File**: tex\latex\pgfplots\pgfplots.sty
  - Part of package: pgfplots
  - Size: 15507.598 KB

- **File**: tex\generic\pgf\systemlayer\pgfsys-luatex.def
  - Part of package: pgf
  - Size: 10594.623 KB

- **File**: tex\latex\ms\everyshi.sty
  - Part of package: ms
  - Size: 2102.632 KB

64
F.2 Debian 8.8

1. Open the terminal and log on as the root user using the command `su`

2. Execute the command `apt-get install texlive` (329 MB will be downloaded and 493 MB of memory will be used on the disc after the installation is complete. Approximately 1.1 GB of memory on the disc was required for Extrahera, which can be explained by the fact that more dependency files had to be installed)

3. All \LaTeX\ test programs and commands presented in this thesis are now runnable
G Groff installation guide

G.1 Windows 10

1. Download groff-1.22.3-3-w32-bin.zip (size 8390 KB) from https://sourceforge.net/projects/ezwinports/files/groff-1.22.3-3-w32-bin.zip/download

2. Unzip the file (the unzipped folder will use 29.9 MB memory on the disc)

3. Name and place the folder at desired location

4. For easier use, add the path to Groff’s bin-folder to environment variables

5. Download grap-1.43-setup (size 1 MB) from http://gnuwin32.sourceforge.net/packages/grap.htm

6. Run the file and follow the installation guide (after installation an additional 2.2 MB of memory will be used on the disc)

7. All Groff test programs and commands presented in this thesis are now runnable

G.2 Debian 8.8

1. Open the terminal and log on as the root user using the command su

2. Execute the command apt-get install groff (3240 KB will be downloaded and 10.4 MB of memory will be used on the disc after the installation is complete)

3. Execute the command apt-get install grap (141 KB will be downloaded and 380 KB of memory will be used on the disc after the installation is complete)

4. All Groff test programs and commands are now runnable
H HTMLDOC installation guide

H.1 Windows 10

1. Download htmldoc-1.9-windows.msi (size 3.76 MB) from https://github.com/michaelrsweet/htmldoc/releases

2. Run the downloaded file

3. Follow the installation guide (here one can see how much memory the installation requires, which is 15 MB)

4. All HTMLDOC test programs and commands presented in this thesis are now runnable

H.2 Debian 8.8

1. Open the terminal and log on as the root user using the command su

2. Execute the command apt-get install htmldoc (5.872 MB will be downloaded and 9.697 MB of memory will be used on the disc after the installation is complete)

3. All HTMLDOC test programs and commands presented in this thesis are now runnable
I ReportLab installation guide

I.1 Windows 10

1. Download python-2.7.13 (size 18712 KB) from https://www.python.org/

2. Run the file and follow the installation guide. Make sure "pip" and "Add python.exe to Path" are enabled during installation

3. Open the terminal and execute the command pip install reportlab (this will download 1.2 MB of data). Python with ReportLab installed will use 78.3 MB of memory on the disc

4. All ReportLab test programs and command are now runnable

I.2 Debian 8.8

Python should already be installed on the Debian 8.8 OS. If it is not installed, follow the steps in numerical order. Otherwise proceed to step 4.

1. Open the terminal and execute the command sudo apt-get install python

2. Execute the command sudo apt-get install python-dev build-essential libtiff5-dev zlib1g-dev libfreetype6-dev liblcms2-dev libwebp-dev tcl8.5-dev tk8.5-dev python-tk

3. Execute the command sudo apt-get install python-pip

4. Execute the command pip install --upgrade reportlab

5. Approximately 200 MB of memory will be used on the disc after these installations are complete

6. All ReportLab test programs and commands are now runnable