UPPSALA
UNIVERSITET

IT 19 012
Examensarbete 15 hp
Maj 2019

Analysis of Tabula

a PDF-Table extraction tool

Gustav Rosén

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Analysis of Tabula

Gustav Rosén

PDF is a widely used text document format used by both the private and the public sector. It is designed to create layouts of text and figures on a virtual page. Research groups often publish reports in this format including raw data in tables. The content of PDF-tables can be difficult to extract, an issue the National Food Agency often runs into. Building a PDF-interpreter from the scratch is a complex and overwhelming task but there are plenty of available PDF-Table extractors. While none meet the specific requirements of the National Food Agency the most effective tool, Tabula, is open source. By analyzing the source code an evaluation of extending Tabula can be made to possibly meet the requirements in the future. However, the lack of documentation and poor class definitions makes the source code arduous to understand. Building a new application using the same library as Tabula appears to be a more promising approach.

Handledare: Kristina Essén
Ämnesgranskare: Lars-Henrik Eriksson
Examinator: Olle Gällmo
IT 19 012
Tryckt av: Reprocentralen ITC

Contents

1 Introduction

1.1 National Food Agency

The National Food Agency(NFA) is a Swedish government agency in Uppsala that among other things analyze food products for their health effects. The agency both performs its own research and uses external research from other organisations to make their own analyses.

The National Toxicological Program(NTP), an agency based in USA, publish lab-reports online as text documents in Portable Document Format (PDF). These documents are constantly used by the department for Risk and Benefit evaluation of NFA. The files, which contain conclusions and discussions, are fluent streams of texts as well as raw data presented as tables. As the Risk and Benefit department is mostly interested in the raw data, the files are regularly scanned for tables and the tables' content extracted. The department is looking for a software solution to ease the process, as to date this is done manually.

1.2 Portable Document Format Background

Portable Document Format (PDF)[9] is a widely used text-document format created by the company Adobe. The official main application for the file format is Adobe Acrobat[9]. Selecting and copying a table from a PDF-file to system clipboard in Adobe Acrobat will not preserve the formatting of the text. For tables this means that the cells loses their connection to the table headers and the records may become illegible. Assuming the table's content is raw text, it is however possible to extract a whole table and insert into a new table by manually copy and pasting each cell one at the time and aligning table headers. Not only do this make for a tedious job with immense time requirement for large and many tables but it is also error prone.

As PDF is modifiable, the describing properties for a table is still contained in the file. In other words the information mapping text segments to their headers in a table ought to be extractable in some way. However by using a raw text-editor it becomes apparent that the source of PDF-code is absolutely illegible for a human. *Figure 1* shows a simple PDF-file opened in a raw text-editor.

Even though PDF was created for Adobe Acrobats official applications it has become a such widely used format that unofficial PDF-handlers are abundant. For example many web-browsers - such as Google Chrome[10] - support reading PDF-files. There are several applications made for table extraction aswell, though neither of the applications fulfills the needs of the NFA fully, see **Chapter 11**.

Figure 1: The picture shows a PDF-file opened in the raw text-editor notepad. The file doesn't contain more than 3 pages and a single word. Yet it is undeniably uninterpretable for a human.

1.3 Terminology

To avoid misconceptions a clarification of the terminology and acronyms used in this report follows.

- List - For the purpose of this report a "list" is a generic collection of values of the same data-type, which may be sorted or unsorted.
- Inheritance - The terms *inheritance* and *extension* of classes are used interchangeably. This report also doesn't make a difference between indirect inheritance or direct inheritance unless specified.
- NFA - National Food Agency (Svenska: Livsmedelsverket). Government agency analyzing food products.
- NTP - National Toxicology Program. Agency located in USA which often publish food analysis reports as PDF-files.
- CSV - Comma Separated Values. A standardized format for plain text documents for representing tables.

The report uses graphs to visualize application structure. Classes are represented by brown rectangle nodes. The difference between abstract classes and interfaces are mostly irrelevant here so they are equally portrayed as green rectangular nodes. Similarly inheriting and implementing are portrayed with the same filled arrow. The node at the arrow's start is the superclass or interface and the node at the arrow's end is the subclass that is inheriting or implementing the superclass. The hollow arrows denote classes using other classes. This could either be by creating objects of the class, by collecting a list of the objects, using objects of the class in functions or output for a function etc. The class at the arrow's end is the class using the class at the arrow's start. *Figure 2* and *Figure 3* demonstrate the meaning of the graphs.

Additionally whenever a class name is referred to, its name will be `in typewriter style` to better differentiate between when the word is used to refer to a particular class as opposed to the word used in general. For example a `Table` would refer to a class called "Table" while table in normal style would refer to a table in general.

Figure 3 shows an example graph. It shows that `Book` uses `Page`. It could be that it collects `Pages` as pages make up a book. It may also include a reference to a `Writer`. `Library` is a subclass of `Building` that collects `Books`. `Person` is an abstract class or an interface and can not be instantiated. `PeopleRegister` collects objects of class `Person`. As `Person` can't be instantiated `PeopleRegister` technically collects `Librarian` and `Writer` but views both types of objects as instances of `Person`.

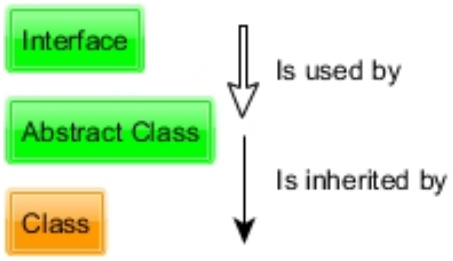

Figure 2: Demonstrating the graphical notations. Interfaces and abstract classes are green nodes. Hollow arrows denotes "used by" and filled arrows denotes "is inherited by".

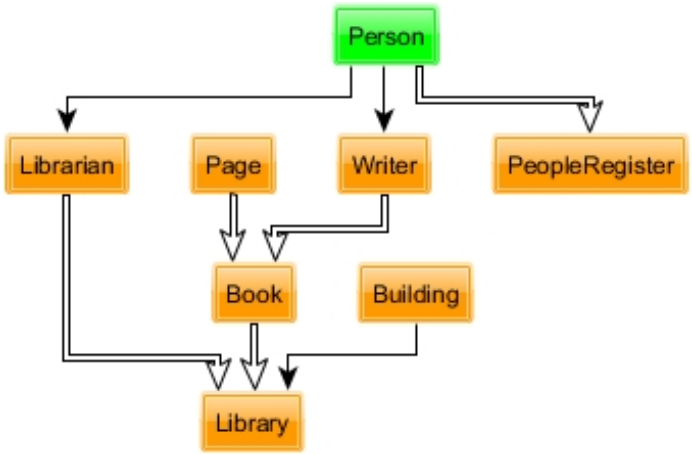

Figure 3: An example of the graphical notations.

2 Problem Description

The extraction problem is defined as such:

> *Given a PDF-file with tables, return the tables in a processable format with the same content as the original tables.*

This can be further divided into several separate subproblems:

1. *Find tables.*
2. *Extract table content.*
3. *Present in a new format.*

For detecting tables a solution could be to only tell the location of tables; not what the tables contain. At a first glance this is only useful for large documents. For smaller documents it is reasonable to let users manually select tables. A solution for this subproblem alone, an output with the page numbers with detected tables would suffice. Although for the next step, identifying which sections of the pages that are parts of a table becomes important.

Extracting table content is mainly about identifying rows, columns and eventually cells. The record that a cell holds is dependent on the cell's related headers. Thus the application's effectiveness in determine cell position in the table is important to keep the data intact.

The extraction of tables is dependent on the detection of it. If the detection is done manually it creates the problem of what happens if the selection is incorrect. If pieces of the selected area deviate from the rest, the extraction algorithm can then either 1) discard them and deem them to be outside of the table, or it can 2) assume the selection is correct and consider them as parts of the table and adjust the analysis appropriately. This could easily happen with page numbers or fluent streams of text. Additionally it isn't intuitive what is supposed to be consider as part of the table. Table titles and captions may seem like important parts of tables although they don't follow the table formation. However if the identification is done automatically the extraction algorithm can assume the selection is correct.

Eventually the extracted tables have to be converted to some suitable output format. The output format has to be both intuitive for a human and also be of a standardized format so that it can be imported into other applications.

The original intent of this project was to build an extractor that solves these issues. Due to the complexity of raw PDF-code and the abundancy of existing tools that almost, but not quite meets the requirements, it is most reasonable to further develop an existing tool. This proved to be more difficult than anticipated as it requires a deep understanding of the chosen tool. This project's focus came to be laying grounds for a possible extension for an existing tool instead, rather than an actual implementation of such. To this end, one application, Tabula, has been chosen as the focus area, see **Chapter 3**. This project aims to analyze Tabula on aspects of its effectiveness and compatibility to improvements.

It is assumed the reader has great understanding of Java as a programming language, Java applications and/or objective orientated programming, as the analysis part of this project is of Java source code.

3 Prior Work - Tabula

Tabula is an application for automatic detection and extraction of tables from PDF-files[2]. Tabula can either automatically detect tables or let users manually graphically select them. It uses two different extraction algorithms to extract selected tables into several different bare-bone formats such as Comma Separated Values (CSV).

Tabula assumes that the target table is text-based. Thus, if a table in a document is represented as a picture Tabula will not be able to extract its content. Therefore the table must have been created with some integrated PDF-creator tool and not be represented by an imported image. Tabula is open source and is missing documentation. In the background Tabula uses Apache.PDFBOX, see **Chapter 11.3**.

To select tables manually means the user graphically selects areas of the PDF-pages by dragging rectangles over tables. It is possible to let Tabula automatically detect tables first and then modify the selection. Whilst the automatic detection works well for simple tables such as *Figure 4* it does not handle NTP's test files.

The two extraction algorithms supported either looks for separation of cells by ruling lines or by white space, see **Chapter 5.6**, *Figure 4* and *Figure 5*. The extraction algorithms assume the selection is correct. Thus, if it isn't, the extraction will be faulty. In general the extraction algorithms handle the same tables as the detection algorithm.

In terms of output Tabula supports several formats including Comma Separated Values (CSV). CSV is a standardized text-format representing a table[8]. It uses commas to denote a new cell and newline-characters to denote end of a row, which suits the needs of NFA. Following figure shows the table *Figure 4* in CSV.

```
a,  b,  c
d,  e,  f
g,  h,  i
```

Tabula is an excellent tool to extract most tables. It is superior to its competitors as it handles many of the general test files, has automatic table detection, is open source and uses a suitable output format. However just like its competitors, see **Chapter 11**, it can't handle the NTP test files.

a	b	c
d	e	f
g	h	i

Figure 4: A simple table where the cells are separated by lines. Each cell contains a letter in alphabetical order for easy reference.

a	b	c
d	e	f
g	h	i

Figure 5: A simple table where the cells are separated by just space. The underlying structure of the PDF-file could use white spaces or some built-in functionality to create this. Each cell contains a letter in alphabetical order for easy reference.

4 Method

Object-oriented programming is built upon the divide and conquer principal of tasks. A problem is divided in several separate subtasks that can be solved individually, and when combined solve the initial problem. Furhter more, object-oriented design is centered around designing classes responsible for subtasks.[3][12] A well designed object-oriented program has well designed responsibilities for each class. However the design is worthless unless the responsibilities of the classes are known to the developer. Since Tabula-Java lacks most documentation its design is entirely dependent on intuitive naming. In other words the function-, variable-, package- and class-names are important aspects of the code.

In order to analyze Tabula's capabilities to be further developed, a map of the classes and their relations to each other, has been created. The analysis is based on investigating the source code, considering names and types, the flow of data, object creation and so on. Furthermore the application has been analyzed by modifying the code to print various values, modifications of input data for internal functions as well as testing the application as a whole on a variety of different tables. The tables that has been used for testing are either simple tables like *Figure 4* and *Figure 5*, or based on NTP's reports provided by the customer such as *Figure 6*.

Summary of Findings Considered Toxicologically Relevant in Rats and Mice Exposed to *o*-Phthalaldehyde for 3 Months

	Male Sprague Dawley Rats	Female Sprague Dawley Rats	Male B6C3F1/N Mice	Female B6C3F1/N Mice
Nonneoplastic effects	Nose: inflammation, suppurative (0/10, 10/10, 10/10, 10/10, 10/10, 10/10); olfactory epithelium, atrophy (2/10, 10/10, 10/10, 10/10, 7/10, 6/10); olfactory epithelium, metaplasia, respiratory (1/10, 1/10, 6/10, 6/10, 2/10, 0/10); olfactory epithelium, regeneration (0/10, 0/10, 1/10, 0/10, 4/10, 0/10); respiratory epithelium, hyperplasia (3/10, 9/10, 9/10, 7/10, 3/10, 0/10); respiratory epithelium, squamous metaplasia (0/10, 10/10, 10/10, 10/10, 10/10, 8/10); respiratory epithelium, necrosis (0/10, 0/10, 3/10, 5/10, 10/10, 10/10); respiratory epithelium, regeneration (0/10, 0/10, 1/10, 0/10, 3/10, 6/10); turbinate, atrophy (0/10, 0/10, 7/10, 10/10, 0/10, 0/10)	Nose: inflammation, suppurative (0/10, 9/10, 10/10, 10/10, 10/10, 10/10); olfactory epithelium, atrophy (0/10, 9/10, 10/10, 10/10, 10/10, 7/10); olfactory epithelium, metaplasia, respiratory (0/10, 1/10, 2/10, 5/10, 7/10, 1/10); respiratory epithelium, hyperplasia (0/10, 9/10, 7/10, 4/10, 3/10, 3/10); respiratory epithelium, squamous metaplasia (0/10, 10/10, 10/10, 10/10, 10/10, 10/10); respiratory epithelium, necrosis (0/10, 2/10, 6/10, 9/10, 10/10, 10/10); respiratory epithelium, regeneration (0/10, 0/10, 2/10, 0/10, 1/10, 3/10); turbinate, atrophy (0/10, 2/10, 4/10, 10/10, 10/10, 0/10)	Nose: inflammation, suppurative (0/10, 10/10, 10/10, 10/10, 10/10, 10/10); glands, olfactory epithelium, hyperplasia (0/10, 10/10, 10/10, 7/10, 6/10, 0/10); olfactory epithelium, atrophy (0/10, 10/10, 10/10, 10/10, 10/10, 10/10); olfactory epithelium, metaplasia, respiratory (0/10, 1/10, 8/10, 10/10, 4/10, 0/10); respiratory epithelium, metaplasia, squamous (0/10, 10/10, 10/10, 10/10, 10/10, 6/10); respiratory epithelium, necrosis (0/10, 2/10, 6/10, 5/10, 9/10, 10/10); respiratory epithelium, regeneration (0/10, 0/10, 1/10, 0/10, 1/10, 4/10); turbinate atrophy (0/10, 4/10, 6/10, 10/10, 8/10, 0/10)	Nose: inflammation, suppurative (0/10, 10/10, 10/10, 10/10, 10/10, 6/10); glands, olfactory epithelium, hyperplasia (0/10, 10/10, 9/10, 10/10, 8/10, 0/10); olfactory epithelium, atrophy (0/10, 10/10, 10/10, 10/10, 9/10, 9/10); olfactory epithelium, metaplasia, respiratory (0/10, 3/10, 6/10, 3/10, 4/10, 0/10); respiratory epithelium, metaplasia, squamous (0/10, 10/10, 10/10, 10/10, 8/10, 0/10); respiratory epithelium, necrosis (0/10, 2/10, 7/10, 6/10, 8/10, 10/10); respiratory epithelium, regeneration (0/10, 0/10, 0/10, 1/10, 3/10, 6/10); turbinate atrophy (0/10, 7/10, 9/10, 10/10, 7/10, 0/10)
	Larynx: inflammation, chronic active (1/10, 2/10, 8/10, 10/10, 10/10, 10/10); metaplasia, squamous (0/10, 1/10, 8/10, 10/10, 10/10, 10/10); necrosis (0/10, 0/10, 1/10, 5/10, 9/10, 10/10); regeneration (0/10, 0/10, 3/10, 2/10, 2/10, 6/10)	Larynx: inflammation, chronic active (0/10, 1/10, 1/10, 9/10, 10/10, 10/10); metaplasia, squamous (0/10, 1/10, 4/10, 10/10, 10/10, 10/10); necrosis (0/10, 0/10, 0/10, 1/10, 7/10, 8/10)	Larynx: inflammation, chronic active (0/10, 0/10, 0/10, 4/10, 10/10, 10/10); metaplasia, squamous (0/10, 0/10, 0/10, 3/10, 10/10, 3/10; necrosis (0/10, 0/10, 0/10, 0/10, 1/10, 10/10)	Larynx: inflammation, chronic active (0/10, 0/10, 0/10, 0/10, 9/10, 10/10); metaplasia, squamous (0/10, 0/10, 3/10, 10/10, 10/10, 8/10); necrosis (0/10, 0/10, 0/10, 0/10, 3/10, 9/10)
	Trachea: fibrosis (0/10, 0/10 0/10, 5/10, 3/10, 0/10); inflammation, chronic active (0/10, 0/10, 4/10, 8/10, 9/10, 10/10); metaplasia, squamous (0/10, 0/10, 4/10, 10/10, 6/10, 6/10); necrosis (0/10, 0/10, 0/10, 3/10, 8/10, 8/10); regeneration (0/10, 0/10, 0/10, 7/10, 7/10, 6/10)	Trachea: fibrosis (0/10, 0/10, 0/10, 2/10, 6/10, 0/10); inflammation, chronic active (0/10, 0/10, 3/10, 5/10, 10/10, 10/10); metaplasia, squamous (0/10, 0/10, 3/10, 10/10, 7/10, 7/10); necrosis (0/10, 0/10, 0/10, 3/10, 3/10, 8/10); regeneration (0/10, 0/10, 1/10, 7/10, 10/10, 9/10)	Trachea: inflammation, chronic active (0/10, 0/10, 0/10, 1/10, 9/10, 10/10); metaplasia, squamous (0/10, 0/10, 0/10, 3/10, 10/10, 3/10); necrosis (0/10, 0/10, 0/10 0/10, 0/10, 9/10)	Trachea: inflammation, chronic active (0/10, 0/10, 0/10, 0/10, 10/10, 10/10); metaplasia, squamous (0/10, 0/10, 0/10, 0/10, 9/10, 2/10); necrosis (0/10, 0/10, 0/10, 0/10, 2/10, 10/10)

Figure 6: An example table from an NTP report. It's on o-Phtalaldehyde health effects on mice considering various aspects.

5 Source Code

5.1 Project structure

Tabula consists of two separate projects. Tabula-Java and Tabula. The latter is an application based on the former written in Ruby. To avoid ambiguity the Ruby version will henceforth be referred to as Tabula-Ruby. Tabula-Ruby's purpose is to implement a graphical user-interface for Tabula-Java which is - as the name implies - written in Java and is the backend of the application. Although its intended use is as a library for Tabula-Ruby it can be executed alone as a command line application. Therefore Tabula-Java can be analyzed separately and is the focus for this project.

The source code of Tabula-Java follows the principle of hierarchical object-oriented design containing about 40 files. For the purpose of clarity of the application's functionalities relevant in this analysis, some classes have been left out or bundled together with other classes. The classes' general functionalities can be divided into five categories: Extraction, Detection, Writing, Pre-Extraction and Page Representation. In this categorization some classes don't fit either category and are left uncategorized. Pre-Extraction - distinct from Extraction - interprets the raw PDF file and converts it into objects of Page Representation. Page Representation is a type of internal representation of the PDF-document. It includes classes like `Table` and `TextElement` among others. Extraction turns Page Representation objects into abstract tables. Writing turns the extracted abstract tables into text files. Detection handles the automatic table detection. Writing, Detection and Extraction classes are each bundled into a Java-package and folder. *Figure 7* shows the classes and their general relations. The rest of this chapter goes through the different classes in detail.

5.2 CommandLineApp

`CommandLineApp` is the main class and runs the program. It interprets the command line arguments and puts all other classes together. This is the only class associated with the command line. Tabula-Ruby implements a graphical user-interface which replaces `CommandLineApp` meaning this class is ignored in the final product.

In `CommandLineApp` there's an enum representing chosen extraction method. It has three possible values, two of them refer directly to an extraction method and the third is just called `Decide`. The extraction method will be set to `Decide` if the method is unspecified by the user. It means that Tabula will automatically choose a suitable extraction method.

5.3 Rectangle

Tabula uses several different classes to represent elements on a PDF-page. These are the classes in the category Page Representation. For any element on a page its location and size are essential attributes. `Rectangle` is used as a

superclass for locations and sizes of elements in a PDF-document. The complete subclass list - including indirect inheritance - of `Rectangle` is `TextElement`, `TextChunk`, `Cell`, `Line`, `Table`, `RectangularTextContainer` and `Page`, see *Figure 9*. A `Rectangle` alone is never created and could've been an abstract class.

5.4 Text

As previously mentioned in **Chapter 3**, Tabula only works if the content of the table is text and not images. This means that text representation is also representation of individual data records. `TextElement` constitute the base of text representation in the table. It represents a single or several characters on a page and contains attributes of what character it is as a String and its font.

`RectangularTextContainer` is an abstract class inherited by `TextChunk` and `Cell`. `TextChunk` contains a list of `TextElements` and `Cell` contains a list of `TextChunks`. As easily seen in *Figure 10* all of these classes inherit `Rectangle` meaning they all have a location and size. The exact purpose and use of these classes are unknown.

5.5 Table

The class `Table` represents tables in the document. It has a field representing the table's cells which is of the data type list of `RectangularTextContainers`. `RectangularTextContainer` is an abstract superclass of `Cell`. Thus it can contain both `Cells` and `TextChunks` although the field name is "cells". `Table` also posseses an extraction algorithm which is activated on the `Table's` content. `Table` has a subclass `TableWithRulingLines` which has a list containing `Cells`.

5.6 Extraction Algorithms

Tabula has two separate extraction methods, one which is based on finding ruling lines in a table. This method is referred to as lattice in the user-interface. Lattice mode will identify cells in a table as separate if they are divided by a ruling line. *Figure 4* shows a table that is well suited for the lattice extraction method. The other extraction method is referred to as "Stream" in the user-interface. This method will instead treat text as separate cells if pieces of text are far apart. *Figure 5* shows a table well suited for the stream extraction method.

In the source code these two methods are represented as two classes implementing the interface `ExtractionAlgorithm`. The class `BasicExtractionAlgorithm` corresponds to stream extraction method and `SpreadsheetExtractionAlgorithm` corresponds lattice extraction method.

The extraction method is globally set for an entire run of Tabula-Java. A `Table` does contain an individual object of an extraction method, so tables could technically contain extraction methods of different classes. However a

`Table's` extraction method is set to what was selected in `CommandLineApp`, see **Chapter 5.2**. Thus all tables get the same extraction method for a single run. If a document contains both types of tables a user will have to run the application twice, once with each extraction method, to properly extract all tables. This is also true for Tabula-Ruby.

5.7 Detection

A package and folder called "Detectors" handles the automatic table detection. In Tabula-Ruby users can use the automatic detection and modify the selections made. It is important that the selection is flawless as the extraction methods assume it is.

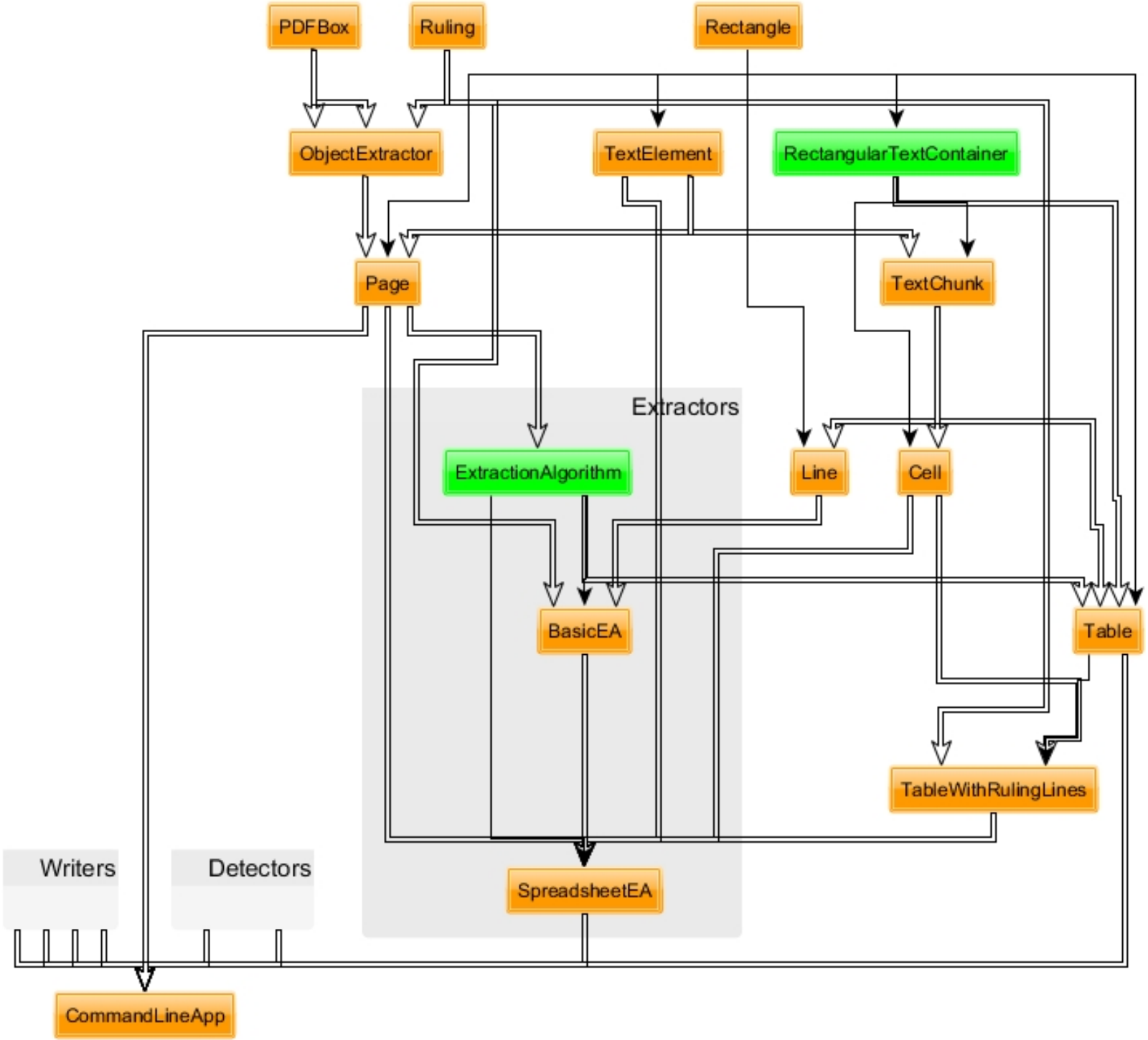

Figure 7: A diagram showing the relations of classes in the source code of Tabula-Java. The black arrows denote classes extensions and the hollow arrows denote a class using another class. Note: Understanding the details of this chart is not important for the rest of this report, neither is the diagram entirely exhaustive

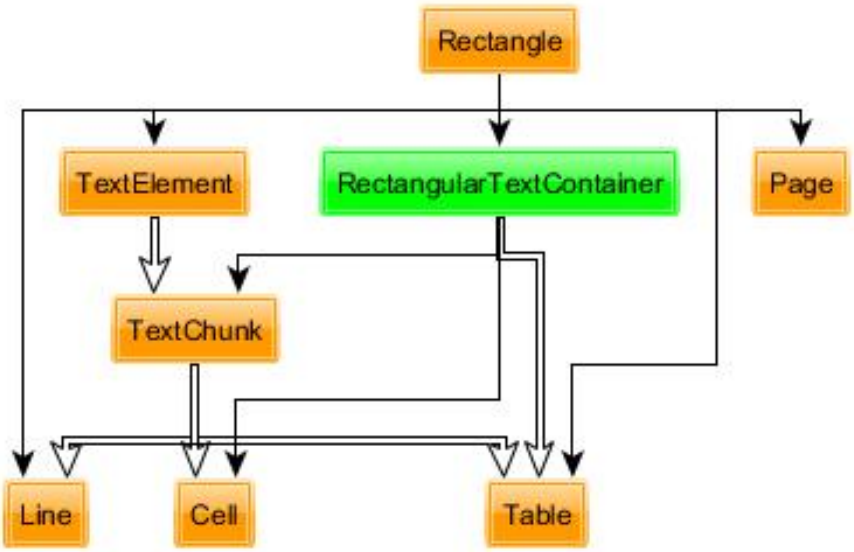

Figure 8: The graph shows the relation of classes relating to the class Rectangle only, for easier deeper analysis.

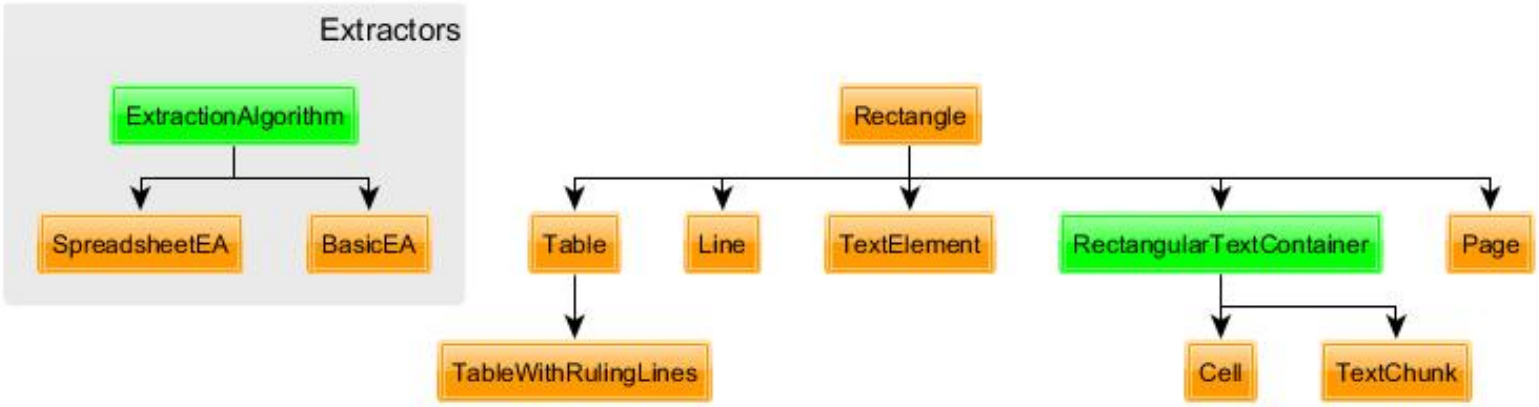

Figure 9: The graph shows only the inheritance of classes.

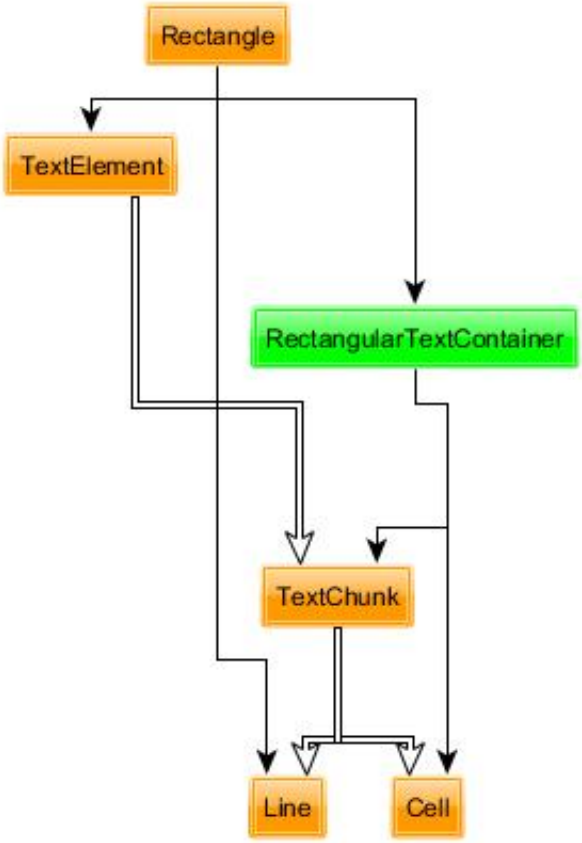

Figure 10: The graph shows the relation of classes relating to text representation only, for easier deeper analysis.

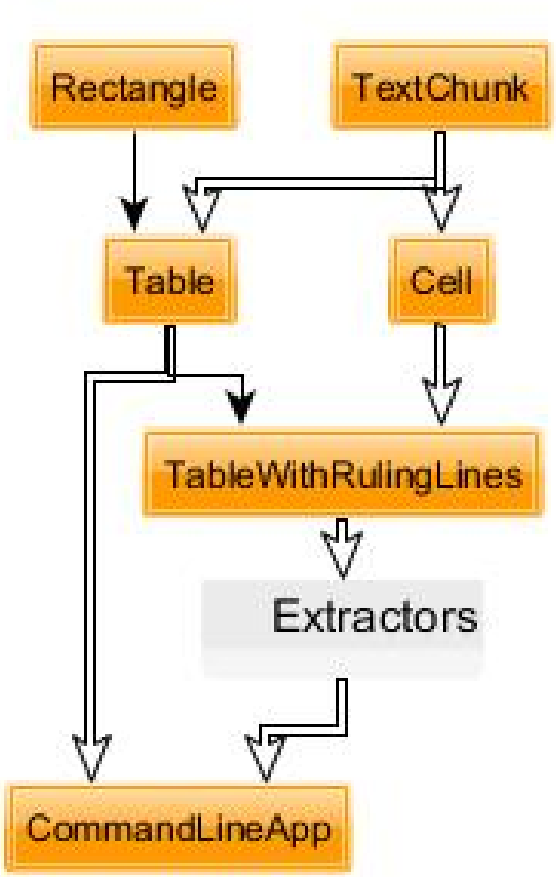

Figure 11: The graph shows the relation of classes relating to cell representation and tables only, for easier deeper analysis.

6 Analysis

6.1 Class Structure

The source code of Tabula-Java is unstructured where the classes' roles are unclear, ambiguous and inconsistent. Tabula-Java is divided into several packages although most classes lie loosely in the root package. *Figure 7* shows the complexity of the class relations.

`TextChunk` is an example of a class which purpose is unclear. The class name is generic and doesn't say much about its function other than that it relates to text. `TextChunk` could represent a single letter, a sentence or a paragraph. `RectangularTextContainer's` has a similar issue. It is an abstract class inherited by `TextChunk` and `Cell` though `TextElement, Page, Table` and `Line` does not inherit `RectangularTextContainer` yet they inherit `Rectangle` and contains text. It is used in `Table` where there's a list called "cells" of datatype `RectangularTextContainer`. Hence `RectangularTextContainer` is likely implemented to enable `Table` to contain `TextChunks` and `Cells` indifferently. However since the field name in `Table` is "cells" and `RectangularTextContainer` doesn't implement any notable features, the benefit of this design remains unknown. This also grants `Table` the ability to contain no objects of class `Cell` in the field "cells" and only `TextChunks`, while a table without cells arguably isn't a table.

`TextElement` seems to represent a single character but it has a String for the character type, not char. The word "text" also suggests there are several characters in a row. If it's intended to hold more than one character the difference to `TextChunk` is seemingly none. If it's intended to only hold a single character it ought to be enforced either by type or with guards to maintain encapsulation principles[3] and transparency.

6.2 Inheritance

The essential idea of inheritance is about defining a common behavior for multiple actors in the code and reusing the code of that behavior for each respective actor. Typically a subclass defines a subtype of its superclass. Depending on the implementation of subtyping, the code becomes more or less adaptable to changes. The inheritances used in Tabula are badly motivated and make the code unintuitive and fragile when subjected to changes.

The existence of `TableWithRulingLines` suggests that `Table` wouldn't have ruling lines. The relation between these two classes is arbitrary and it could in theory be the opposite; that `Table`**`Without`**`RulingLines` inherits `TableWithRulingLines`. This use of inheritance is subclassing for *variance* and ought to be avoided. [3]

`TextChunk, TextElement, Cell, Table` and `RectangularTextContainer` all inherit `Rectangle`, thus they all have geometrical properties. `TextChunk` contains a list of `TextElements`. Hence a `TextChunk` contains the location of each of its `TextElements` **and** an additional location

for the TextChunk itself. This raises some questions. What does the location of TextChunk actually represent? What does it mean if the TextElements' locations are different from that of the TextChunk? What does it mean if they are all the same? Cell works similarly and contains a list of TextChunks, and Table contains a list of RectangularTextContainers. All of these classes inherits Rectangle. So a Table contains its own location and size and indirectly the locations and sizes of all of its Cells, TextChunks and TextElements.

An explanation for this could be that the classes containing the other objects work as bounding boxes for the collections. In other words a Table's area would then mark the area in which all the Table's Cells are located and the Cell's area must always be completely inside that of the related Table. However, there is no guard in the code that guarantees this and proper testing whether this always holds true has not been made. It would also mean that the container class duplicates data as the bounding box could be calculated using only the list. The problem with duplicating data in this scenario is that when Table is created it also creates its rectangle and if Cells are added or modified the Table's bounding box has to be updated too.

7 Suggested Redesign

7.1 Purpose

In order to better demonstrate the complications of the current design, a redesign is proposed. The redesign aims to change names and relations of classes for a more intuitive and expandable code. It maintains the overall functionality and abstract algorithms of the original design. Thus the redesign also keeps some of the original design's structure. *Figure 12* shows the complete suggested redesign.

7.2 Overview

The flow of data is very similar as in the original design. `Detectors` create `PageAreas` as selections to be extracted. `Pre-extraction` uses these selections to turn them into `Tables`, the internal representation of what a table looks like. The `ExtractionAlgorithms` have `Tables` as input and creates tables represented as two-dimensional-arrays with `Strings`. The difference to `Table` is that these arrays represent only the content of the table, not its graphical properties. `Writers` then use those arrays to create `Strings` and prints them to a file.

One noteable difference to the original design is the relation of `ExtractionAlgorithm` and `Table`. As an extraction algorithm is pointless without a table as input, and a table is a concept separated from an extraction of it, `ExtractionAlgorithm` uses `Table` in the redesign, rather than `Table` containing an extractor as in the original design.

A `PageArea` is a rectangle on a page. Its only difference to `Rectangle` is that it contains the page number where the area is located.

7.3 Tables and Records

In the suggested redesign tables are represented by a single class, which always contains a field for ruling lines. If the table doesn't have any ruling lines the field is set to empty. This makes the `Table` class' purpose clear. An alternative for this could be to have an abstract class `Table` and two classes inheriting `Table` directly. One with ruling lines and one without. This design makes it more intuitive how to implement new types of tables if needs be, but either design solves the issue of subclassing for variance.[3] `Table` doesn't have a field for its dimensions but rather has a method for calculating a bounding box for the contained elements.

As previously mentioned in **Chapter 2** a cell's position is an important attribute and `Cell` therefore inherits `PageArea`. While the cells' location is important the location of the text within the cell is not. As long as a piece of text is included in a `Cell` the meaning of it is apparent and the actual text doesn't need to be represented by anything else than a primitive `String`. `Table` stores a list of `Cells`.

Note that this design makes it so that a `Table` can stretch over multiple pages while a `Cell` belongs to one page only, due to their relation to `PageArea`.

This design also locks the application to two-dimensional tables. However since a table with more dimensions are quite unusual in a PDF-document a three-dimensional extractor would probably have a very different behavior and would need another complete redesign anyway.

`Table, Cell` and `Ruling` are all bundled together as a package to better visualize their relation to each other.

7.4 FlowControl

An extra class has been added, closely related to `CommandLineApp`, called `FlowControl`. It separates the user-interface from controlling the application flow. The difference between `FlowControl` and `CommandLineApp` is that in `CommandLineApp` the input string is converted to the internal representation of choices made by the user while in `FlowControl` the internal representation is the input and is used to activate the corresponding classes.[11] Similar to the original design `CommandLineApp` can be omitted for Tabula-Ruby. However the `FlowControl` class is still used. It ought to make new installations of user-interfaces easier as such a class only needs to translate user-actions into a few number of values and send these to `FlowControl`.

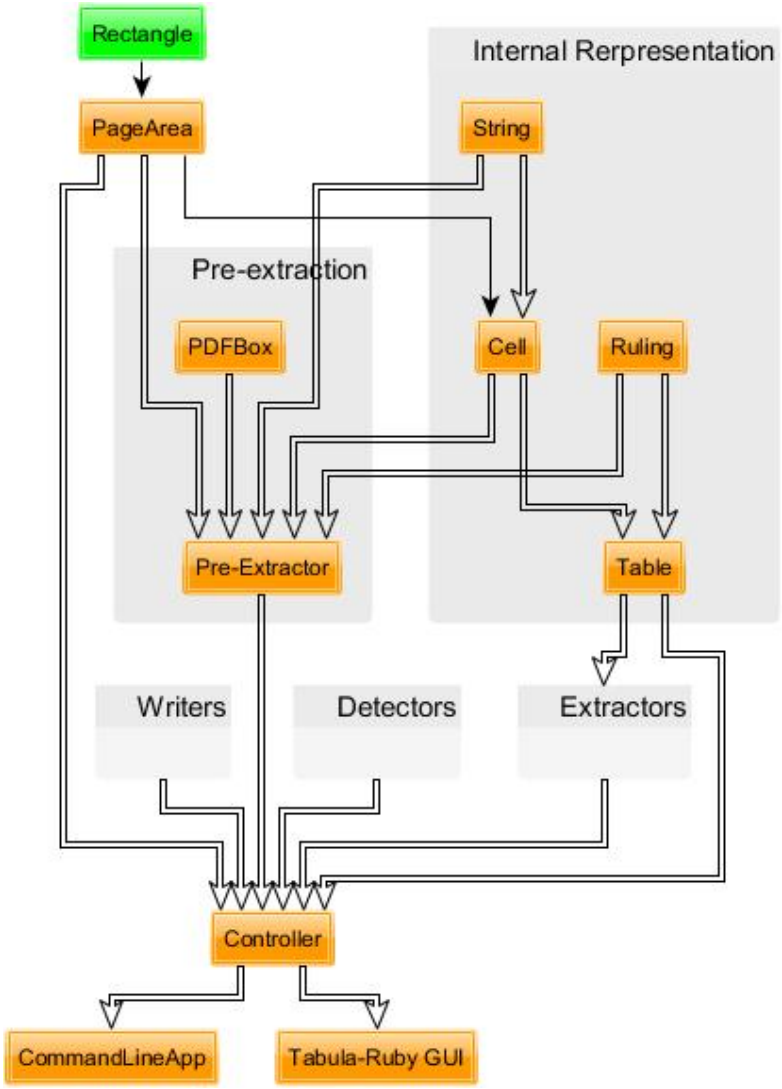

Figure 12: A suggested different design. "String" is visualized in the graph to emphasize the difference of the original design. It is however the standard datatype and not a new class.

8 Conclusions

While Tabula has made it easy to add an additional extraction method as an additional option, it is not as well designed on aspects of integrating the method with supporting classes which makes Tabula an inconvenient development environment. Tabula was chosen mainly for being open source and having highest effectiveness when applied on the test files. But the fact that it is so poorly documented made the project incredibly time consuming and could have resulted in several misconceptions. Some classes' exact functionalities are difficult to determine without documentation or spending immense amount of time on testing a large and diverse input data set. Thus some of the analysis of the functionalities may even be incorrect. In either case, this shows that it is very dependent on the documentation.

More over Tabula may not be as close to solving the NTP files as intially excpected. The existing extraction methods are built on finding ruling lines or a lot of space between text to separate columns. Neither of those methods consider that a single cell may have multiple rows, as many tables do in the NTP test files. This may require a behaviour entirely different from what already exists in Tabula. In other words, Tabula may be far away to solve the extraction of NTP files in terms of implementation.

If Tabula would be expanded, the most reasonable approach would be to insert a new class as an `ExtractionAlgorithm` that uses the minimal amount of other classes. A benefit to gain from using Tabula would be that Tabula-Ruby does implement a seemingly proper graphical user-interface.

9 Evaluation

The original intent of the project was to build a table extractor for practical use. As explained in **Chapter 2**, the focus of the project changed due to unanticipated difficulties. Perhaps a more suitable approach would be to abandon Tabula after some amount of time and pursue some other solution, more likely to be of practical help for NFA. As the conclusion of the analysis, see **Chapter 8**, is that Tabula is not very well suited to be expanded, the deeper analysis of the source code is practically useless if NFA decides to not use Tabula. One alternate approach would be to evaluate several different tools in order to determine which is most suitable to expand. Had the difficulties of understanding Tabula's source code been anticipated then the project could have been directed towards a solution of more practical use.

The specification and documentation of Tabula are missing which makes the analysis dependent on a personal assessment of the code. Additionally, the method of analysis has arbitrary constraints as the code could be tested on an unlimited amount of data. For an application with given specification or documentation a more solid research could've been produced.

10 Future Work

10.1 PDFBox

Through this project it has become apparent that much of a page's content is actually extracted through the library PDFBox which is in this report considered as part of the Pre-extraction. Instead of expanding Tabula another possible approach would be to build a new application with PDFBox; just like Tabula. The redesign, see **Chapter 7**, could work as a first draft for the design of application, though it was created to match Tabula's structure. A completely independent design may be a better starting-point.

10.2 Test Driven Development

A problem in this project is that the constraints are rather arbitrary. The customer wants a system to extract tables from a specific institution. But NTP is free to change the format they use. This makes it difficult to define the scope of target input data as it could include anything from all tables in all reports published the last 10 years to only a couple of tables from the last 6 months.

To meet the requirements, both the scope of the input data and their expected output have to be properly defined. The data set needs to be large, to ensure capturing all aspects of NTP's tables. By using test driven development with a well defined data set, the project can be more easily directed towards the NFA's needs.

10.3 Tabula-Ruby

Tabula is divided into two separate projects, see **Chapter 5.1**. The end user of the product is not necessarily familiar with command-line applications which makes the GUI - Tabula-Ruby - a crucial aspect. Thus, future endeavors to analysis ought to include Tabula-Ruby into the project. From a glance on Tabula-Ruby it appears to be successful on the basics of user interface design. If the source code of Tabula-Ruby is well structured there's still a lot unexplored utility to gain by using this application.

10.4 Image Recognition

Not all PDF-files use the internal structure and text segments to create tables. Some PDF-files have imported pictures instead. Extracting data from an image-file is a completely different task based on image recognition. However from an end-user perspective it may seem indifferent and thus it's also a relevant issue.

An image recognition solution could in theory also extract text-based tables as it's trivial to convert PDF to an image format. Such a solution would be more generic.

10.5 From the publisher's end

As a side note to this project the issue could be handled from the publisher's end. It originates from that PDF is designed to create neatly formated text documents where tables are just one out of many features provided. The format is not designed to share masses of data. Thus, any publisher could provide the raw data in another format and prevent the issue from ever occurring. It could however be a chosen strategy to make the data less accessible for copyright issues. However, that is a subject outside of Computer Science.

11 Related Work

11.1 Application: PDFtables

PDFtables[5] is a commercial product which makes an entire PDF-file into a sheet of tables. The program doesn't attempt to discriminate between tables and just text, nor can the user select this manually. It simply extracts everything as a table. This is unsuitable for large files, as with NTPs. The most convenient way to use this program on large documents is to first remove the pages not containing any tables to then run it through PDFtables. However the user is then performing the task in two separate steps.

11.2 Application: PDFtoXLS

This tool only exports to XLS[7]. Only a few applications read this format, most notebly Microsoft Excel[4]. Unless the target use is one of these few applications, another converstion would be needed.

The application also has problems to automatically detect what is a table and often divides a single table into multiple ones.

11.3 Library: Apache.PDFBOX

Apache.PDFBOX is a Java library that contains functions to handle PDF-files[1]. The features are basic operations to extract elements such as text or pictures and can be used as a base for a table-extractor such as with Tabula.

11.4 Application: PDFtoHTML

This tool exports only to HTML. [6] As HTML is an adaptation of XML it is a format more processable than PDF. If an application doesn't support importing HTML-tables it might be a convenient approach to first convert PDF to HTML and then convert HTML to XML. As it's not open source it may not be possible to integrate it into another application, thus, as with PDFtables (see **Chapter 11.1**), it would require the end user to perform the task in two separate steps.

References

[1] Apache. *Apache PDFBox - A java PDF Library*. URL: https://pdfbox.apache.org.

[2] Manuel Aristarán, Mike Tigas, and Jeremy B. Merill. *Tabula*. 2018. URL: https://tabula.technology.

[3] Timothy Budd. *An introduction to Object-Oriented Programming*. 2nd ed. Oregon State University, 1997.

[4] Annika Hegardt and Robert Hellman. *Excel 2010 för affärsfolk*. 1st ed. Pagina Förlags AB, 2011.

[5] *Official website for PDFtables*. URL: https://pdftables.com/.

[6] *Official website for PDFtoHTML*. URL: https://www.pdftohtml.net/.

[7] *Official website for PDFtoXLS*. URL: https://pdftoxls.com/.

[8] Y. Shafranovich. *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. RFC. Oct. 2005. URL: https://tools.ietf.org/html/rfc4180.

[9] Adobe Creative Team. *ADOBE ACROBAT 7.0 CLASSROOM IN A BOOK*. 1st ed. Pearson Education Inc, 2005.

[10] Wikipedia. *Google Chrome*. Sept. 2018. URL: https://en.wikipedia.org/wiki/Google_Chrome.

[11] Wikipedia. *Grasp (object-oriented design)*. Sept. 2018. URL: https://en.wikipedia.org/wiki/GRASP_(object-oriented_design).

[12] Rebecca Wirfs-Brock and Alan McKean. *Object Design*. 1st ed. Pearson Education. Inc., 2003.