Lucerne University of Applied Sciences and Arts

PAWI

Evaluation of Different Content Extraction Algorithms

Author:
Joel Rolli

Supervisors:
Patrick Huber
Patrik Lengacher
Dr. M. Kaufmann

Examiner:
Prof. M. Jud

A thesis submitted in fulfilment of the requirements for the degree of Bachelor of Informatics

in the

Informatics
Lucerne University of Applied Sciences and Arts

December 2014

Declaration of Authorship

I, Joel Rolli, declare that this thesis titled, 'Evaluation of Different Content Extraction Algorithms' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Lucerne University of Applied Sciences and Arts

Abstract

Software Systems
Lucerne University of Applied Sciences and Arts

Bachelor of Informatics

Evaluation of Different Content Extraction Algorithms

by Joel Rolli

TBD ...

Contents

List of Figures

List of Tables

Abbreviations

PAWI	**P A W I**
HTML	**H**yper **T**ext **M**arkup **L**anguage
BTE	**B**ody **T**ext **E**xtraction
DOM	**D**ocument **O**bject **M**odel
TP	**T**rue **P**ositive
TN	**T**rue **N**egative
FP	**F**alse **P**ositive
FN	**F**alse **N**egative
TV	**T**ele**V**ision
CI	**C**ontinuous **I**ntegration
RSS	**R**ich **S**ide **S**ummary
API	**A**pplication **P**rogramming **I**nterface
JDK	**J**ava **D**evelopment **K**it
f x	**f**eature x
MS x	**M**ile**S**tone x
tc x	**t**est **c**ase x
TPR	**T**rue **P**ositive **R**ate
FPR	**F**alse **P**ositive **R**ate
CSV	**C**omma **S**separated **V**alues
DSL	**D**omain **S**pecific **L**anguage
XML	E**X**tensible **M**arkup **L**anguage
JR	**J**oel **R**olli

For/Dedicated to/To my...

Chapter 1

Problem statement

This chapter describes the problem statement, as well as the topical environment in which the project takes place.

1.1 Introduction

This project is done on behalf of the company Layzapp. Layzapp is specialized in second-screen solutions. It is currently working on a mobile application which brings relevant information to a second screen during a TV show. The Internet is crawled to find relevant information about a specific topic. The outcome of this search is a certain number of web pages. The content of these web pages is made usable by removing irrelevant data, such as navigation elements, advertisement and login pages. Removing the HTML code is not a very hard task. But after this first process of removal there is still a lot of irrelevant content left, for instance descriptions of further articles or advertisements. Removing this part is much more complicated.

1.2 Task

The task is to develop a test environment i.e. to evaluate the performance of text extraction algorithms. The test environment needs to classify the quality of a text extraction performed by the different algorithms. For this purpose, each algorithm is fed with a collection of web pages. The outcome is then inspected for its quality. Based on the results and the new knowledge, a new algorithm can be implemented. However, this is a optional requirement.

1.3 Text extraction and algorithms

This section cover the subject of text extraction and describes the known algorithms JusText and Boilerpipe.

1.3.1 Text extraction

Text extraction or content extraction of web pages is a widely discussed field in research. There are several approaches to this field. The two main approaches are page segmentation via visual or DOM features and boilerplate removal. The main drawback of visual page segmentation is that at some point the web page needs to be rendered and processed as image. This task is very time-consuming and many resources are necessary to fulfill it. This project focuses on algorithms which work with DOM features. The basic idea of the two following algorithms was first introduced with the Body Text Extraction (BTE) algorithm [1]. The assumptions are the relevant part of the web pages source code content is usually a continuous text where the density of HTML tags is lower than in boilerplate content. Hence, by breaking up the source code in single sections and counting the HTML tags per visible word. There will be an area where the number of HTML tags do not increase. One can expect that this is the article text. Unfortunately, BTE's performance is very limited, but it is still used to compare different extraction algorithms to each other. The two algorithms Boilerpipe and JusText improve the performance of BTE.

1.3.2 Boilerpipe

The Boilerpipe algorithm [2] is based on the concept described in the paper "Boilerplate Detection using Shallow Text Features" [3] by Kohlschütter et. al. It uses a variate of HTML tags to divide the HTML document into blocks. Each block is classified according to its shallow text features such as word length, link density etc. The classification uses features of the current, previous and next blocks. The features and the classification algorithm are described more closely in the paper.

1.3.3 JusText

The algorithm JusText [4] is based on the concepts described in paper "Removing Boilerplate and Duplicate Content from Web Corpora" [5] by Pomikalek et. al. It uses similar features as Boilerplate but inspects the data for the presence of stop words as well. Some examples for stop words are 'a', 'and', 'but', 'how', 'or', and 'what'. An

article contains more stop words than boilerplate. Based on this information, a better classification can be achieved.

1.3.4 Classification

The main task of the application is to rate the performance of the algorithms. The performance can be defined as how much of relevant text is classified as relevant and how much is classified as boilerplate. In information retrieval, these performances can be described in a confusion matrix as in table 1.

	Classified as content	**Classified as boilerplate**
Actual content	True positive (TP)	False negative(FN)
Actual boilerplate	False positive (FP)	True negative (TN)

TABLE 1.1: The confusion matrix and its meaning

- True positive is an instance which is relevant content and classified as content by the algorithm
- False positive is an instance which is relevant content but classified as boilerplate by the algorithm
- True negative is an instance which is boilerplate and is classified as boilerplate by the algorithm
- False negative is an instance which is boilerplate but is classified as content by the algorithm

These values are the basis for calculating both the recall (also known as sensitivity) which is the fraction of relevant text that is retrieved and the precision which is the fraction of retrieved text that is relevant. These values, their dependencies and some more values are described more closely in the software requirement specification (C.4.4).

1.4 Goals

- Implement a content extraction test framework which can be fed with test data and compare multiple text extraction algorithm with each other.
- Defining a set of representative test data which consists of HTML source files with according text files which define the relevant content of these HTML files.
- Integrate both the JusText and the Boilerpipe algorithm into the test framework.

- Evaluate the results of the content extraction test framework with statistical methods and compose a statement of the quality of the tested algorithms.
- Implement a new algorithm based on results of the test framework and the newly gathered knowledge about text extraction in cooperation with the supervisors. This requirement is optional.

1.5 Summary

An application is needed which can be fed with a collection of web page source code and the corresponding relevant text. These documents are then processed by text extraction algorithms. The outcome is then compared with the relevant text and classification values are calculated. Based on these values the performance of the algorithms can be compared. Possible strengths and weaknesses should be able determined in this comparison.

Chapter 2

Solution development

This chapter describes the approach developing the application. It does not cover each and every detail of the project. Only the key problems are discussed. However all the topics are described more closely in the related project documents in the appendices.

2.1 Approach

2.1.1 Planning

The planning of this project was a very straight forward task. Since it is a small one man project and the start and end dates were clearly defined. However I am working part time for Layzapp and this project was not the only task during the period of time and the project needed to be coordinated somehow with other tasks. My solution was to work about one to two weeks for this project and switch to other tasks for the next one to two weeks. Switching between multiple projects within one week seemed not very effective because one needs always some hours to get back into the topic. A project plan was done for the whole time line. Five milestones and its delivery objects were defined. As we were working with scrum a milestone was defined as sprint and the delivery objects were divided into stories on the start of each sprint. Since it was a one man team no weekly scrum meetings were done. However on the end of each sprint there was a sprint meeting and the delivery objects were presented to the supervisors and the tasks for the next sprint were defined or adjusted. During the project, the tool toggl (A.10) was used as a time tracker and task manager. The whole planning as well as the time tracking can be found in the abbreviations under Appendix A.

2.1.2 Programming

TODO: some other introduction, infos/ref about tools etc.

As a first step a data model and a first approach of a software architecture was developed. This first approach was implemented such that the basic functionality of the application was working. With this working version the data model and the software architecture were approved. The next step was to eliminate the biggest risk, the integration of the two algorithms JusText and Boilerpipe into the application. Prototypes for each algorithm were programmed. Integrating the Boilerpipe algorithm [2] was an easy task since it is implemented in Java and could be used without the need of any modification. Integrating the JusText algorithm [4] was a more difficult task since it is implemented in Python. The final solution was to call the python application with system calls from the java application and read the output text file generated by the python program. The second big risk was the approach comparing the text of the actual content and the outcome of the algorithms and find the classification values (True positive, false positive, true negative, false negative). The different approaches are described for the following example.

The text file with the correct content contains following text.

```
I am an interesting text.  My content is about advertisement in modern times.
```

The extracted text file by an algorithm contains following text.

```
I am an interesting text.  And i am advertisement about socks.
```

The first approach was counting each word in the content file and check if it is the same amount of words are available in the extracted text file. In doing so words could occur in different places in a text but could still classified as correct. This is presented in the example. The word 'advertisement' in the content file is part of the actual content. In the extracted text file, the sentence 'My content is about advertisement in modern times.' is not found but there is still a text passage classified as content where the word 'advertisement' occurs. As a result, the word advertisement would be classified as correct (True positive) with this first approach because it is counted once in each text. So this approach could not be used. In a second approach, the comparison was done by not only comparing a single word but as well as the word before and after the word of interest. For the example above, for each word, a group of three words is built and this group is then compared with the second text. The word "advertisement" would be defined as the word group 'about advertisement in' and the second text is searched for this pattern. Since there is not such a group, the word advertisement would be classified wrong (False positive) which would be correct. With this approach the results were already a lot

better. But there were still situations where this approach was not working. Instead of implementing an elaborated text comparison algorithm we decided to fall back on existing solutions. Merging tools like diff [6] or meld [7] are doing exactly the task we were looking for. We decided to use the open source implementation merge-diff-patch [8] developed by google. The outcome of the comparison are all the words of the input text grouped in 'EQUAL', 'DELETE' and 'INSERT' which can be easy transferred into the needed values True Positive ('EQUAL'), False Positive ('INSERT'), True Negative ('DELETE') and False Negative (remaining words).

The next task after handling these risks was to integrate the prototypes into the main application. The remaining programming tasks were then refactoring the first approach and testing the application. The final software architecture is described in the abbreviation under Appendix D. The test concept is described under Appendix E.

The final programming task was then adapting the application so that it is possible to evaluate a single test instead of getting the results for a summary of tests. The reason doing this is described in the section 2.2.3. The most information about a single test such as Precision, Recall and the extracted text by the algorithms was already available. The more challenging part was the output of the evaluation based on the bocks generated by the algorithms. The problem is that both algorithm have no way of getting the classification of the single blocks. Which means that the two algorithms needed to be modified.

2.1.3 Modification JusText

Even with no prior knowledge at the programming language python, the task was easier then expected. The algorithm is working with blocks during the whole process and prints the blocks classified as content. All we needed to do is to print the classification values at the beginning of each block and parse the output string accordingly with the main application. This is an example of the printed string after my modification.

```
<p class="bad" cfclass="bad" heading="0" word\_count="3" link\_density="1"
stopword\_count="1" stopword\_density="0"\> Kites with Antennas>
```

This string could then be parsed by the Java application into block objects and all the classification data like 'link density' and 'stop word count' could be set for each block.

2.1.4 Modification Boilerpipe

Adapting the boilerpipe algorithm on the other hand was not as easy as expected. Boilerpipe does merge the single blocks during the algorithm and the classification information for some blocks are lost due to this approach. My solution was then to edit the algorithm that each block is backed before it is merged with others and all the available classification information as well. The problem with this approach is that some blocks are classified different at a later point of time and the backed data is not correct anymore. To solve this problem, bigger changes of the algorithm would be needed. However, this solution is good enough to evaluate the results accordingly.

2.2 Results

2.2.1 Statistical data

With the test framework it was finally possible to produce the classification data for the algorithms using an HTML source file and a content file as input data. Doing this with a few test files, it is not possible to produce a concrete quality criterion for an algorithm. A bigger test data was needed. A gold standard test data was used for this purpose. This test data was used for a text extraction competition called CleanEval [9]. The following table shows the overall results when extracting the CeanEval test files with JusText and Boilerpipe and process the results with the test framework.

Algorithm	**Precision**	**Recall**	**Fallout**	**F-Measure**	**Accuracy**
JusText	95.29 %	91.99 %	35.03 %	91.37 %	85.11 %
Boilerpipe	95.15 %	74.38 %	49.31 %	79.56 %	68.17 %

The meaning of the single result types is described in Appendix section C.4.4.

2.2.2 Analysis

This test data was not only used in this but as well in other papers which broach the issue of text extraction. The paper 'More Effective Boilerplate Removal—the GoldMiner Algorithm' [10] used the same test data to compare their Goldminer algorithm to JusText and Boilerpipe. The results from this paper were used to determine if my approach is heading into the right direction or if the outcome is completely wrong. The results from the paper testing the different algorithms with the gold standard are shown in table 2.1.

The results from my test framework are shown in table 2.2

Algorithm	**Precision**	**Recall**
JusText	95.29 %	91.99 %
Boilerpipe	95.15 %	74.38 %

TABLE 2.1: The results for the CleanEval test set from the Goldminer paper

Algorithm	**Precision**	**Recall**
JusText	95.86 %	87.27 %
Boilerpipe	91.14 %	70.60 %

TABLE 2.2: The results for the CleanEval test set from the test framework

The results are not exaclty the same but they are close. The difference could be explained by several points. First, there is no guaranty that the results from the paper are correct. Second, most of the common approaches comparing these text extraction algorithms were done in comparing the correct classified HTML blocks and not the single words like I did it in this project. This can change the results significantly. Following example should clarify this statement.

Suppose we have an HTML document with ten blocks which have a certain amount of words and are classified by an algorithm as defined in table 2.3.

Block No.	**Word count**	**Classification**
1	10	True Positive
2	100	True Positive
3	50	True positive
4	30	True Positive
5	1	False Positive
6	1000	False Positive
7	20	False Positive
8	300	False Positive
9	200	False Negative
10	50	False Negative

TABLE 2.3: Example Blocks vs. Words

The formulas for calculating Precision and Recall are described in the appendix section C.4.4. The calculated values for Precision and Recall from table 2.3 is shown in table 2.4

Algorithm	**Calculation based on blocks**	**Calculation based on words**
Precision	50 %	12.57 %
Recall	66.67 %	25.68 %

TABLE 2.4: Results example Blocks vs. Words

We can see that the difference between the two approach is quite significant because a block which is classified wrong and contains a lot of words is not weighted as high if the values are calculated based on blocks instead of words. Comparing algorithms based on words seems to be the better approach since the results are more accurate. Furthermore, each algorithm can define the size of a block by itself and the two algorithms do not produce the same amount of blocks. Comparing a different amount of blocks does not produce very accurate results as well.

2.2.3 Detailed analysis

On the MS4 meeting (Appendix J), we decided not to implement an additional text extraction algorithm because we needed to investigate the problems of the existing algorithms first so that we have an idea what we need to improve for our own algorithm. So instead of focusing on a new approach, we decided that we are improving the functions of the test framework in a way that it is possible to investigate the results of a single test case more closely. These adjustments are described in this section.

As described in the introduction, the algorithms split the HTML file into blocks and classify these blocks based on several classification data. To investigate, why a certain classification went very bad, it would be helpful to get the information about the blocks classification. After modifying the implementation of the algorithms (2.1.3) it is possible to get this data for both JusText and Boilerpipe. This example shows the results for one block for both JusText and Boilerpipe.

Boilerpipe

```
[link_density: 1.0; classification: BOILEPLATE; word_count: 3; stop_Word_Count:
NOT_DEFINED; text_Density: 3.0; context_Free_classification: NOT_DEFINED; ]
Kites with Antennas
```

JusText

```
[link_density: 1.0; classification: BOILEPLATE; word_count: 3; stop_Word_Count: 1;
text_Density: NOT_DEFINED; context_Free_classification: CFC_BAD; ]
Kites with Antennas
```

The classification values which can be extracted from the algorithms are:

- Link density,
- Classification,
- Word count,

- Stop word count (only JusText),
- Text density (Only Boilerpipe),
- Context free classification (Only justext).

Having these values for a certain block already helps evaluating how the algorithms work and why a specific block is classified correct or wrong. Some classification values are only used by one algorithm. These values are marked as 'NOT_DEFINED' in the output file for the according algorithm.

Chapter 3

Discussion and Outlook

3.1 Conclusion

With the test framework it is now possible to compare text extraction algorithms against each other and get a general information about the performance. We can now rate the performance of the two algorithms JusText and Boilerpipe and with the implemented software architecture it is easy to integrate and test additional algorithms as well. We can not only compare algorithms with each other but as well investigate the single algorithms and their output more closely for specific test cases. This helps to find problems and strengths of existing algorithms and may help to improve them or implement our own algorithm. So it can be told that all the mandatory requirements are fulfilled and with the implementation of the detailed analysis there are even more functions available. The missing requirement is the implementation of the RSS algorithm but with dropping this requirement it was possible to implement the additional functions for investigating the existing algorithms more closely, which will help to improve the existing ones as well as developing a possible new approach.

3.2 Lessons learned

3.2.1 Planning

We did underestimate the effort needed for the documentation. Especially putting together all the single documents and writing the main part took more time then i expected. There are a lot of little things which we did not took in consideration during the planning phase but which need to be done at some point.

3.2.2 Programming

- I realized once more that it is very important to test software early and often. I tried to test all the critical components right from the beginning. However if I did not, a time loss was almost expected for sure because bugs sneaked into the code and as a result, I was searching for the problems on the wrong places.
- I used the programming language Python the first time for this project. Even I only had to do small adjustments on the JusText project , I now know the basic syntax as well as how to interact with it from a Java application.
- I had very basic knowledge about text processing and text extraction. Working with the topic for this project extended my knowledge quite a bit.

3.3 Further work

With the test framework we have now the tool to examine text extraction algorithms very close. The next steps will be an exact evaluation of the weaknesses from the algorithms so we can improve them or we can implement our own approach, which handles these problems better. The test framework can then be used at any point during the development phase to check the performance of the algorithm very easy and compare it with the existing ones.

Appendix A

Planning

A.1 Version

Version	Date	Change	Author
0.1	15.09.2014	Setup document	JR
1.0	28.09.2014	Draft planning	JR
1.1	18.09.2014	Adding overview	JR
1.2	6.10.2014	Stories for MS2	JR
1.3	23.10.2014	Stories for MS3	JR
1.4	7.11.2014	Stories for MS4	JR
1.5	20.10.2014	Stories for MS5	JR
2	8.12.2014	Adding time tracking / adaption for final version	JR

A.2 Planning concept

So as to plan the project, the planning frameworks scrum was used.
For a first rough planning, the assignment is split into working packages and assigned to milestones. Delivery objects are defined for each milestone.
This plan is then assigned to the given time table of about 12 weeks. The project effort is defined as 180 hours. This results in about 15 hours work load per week.

As working with Scrum, each milestone was defined as a sprint. At each start of a sprint, the working packages were adjusted if needed and were then spitted into stories. In the following section all the milestones are listed with its working packages and stories. The working packages were defined at the very beginning of the project and the stories were defined at the start of each milestone/sprint. To see the adjustments made

during the project, the definition of the working packages was not adapted for this final documentation.

A.3 Milestones overview

Name	**Shortcut**	**Weeks**	**Estimated hours**	**Hours total**	**Closing date**
Milestone one	m1	2.5	39	39	01.10.2014
Milestone two	m2	3	45	84	22.10.2014
Milestone three	m3	2	30	114	05.11.2014
Milestone four	m4	2	30	144	19.11.2014
Milestone five	m5	2.5	38	182	08.12.2014

A.4 Delivery objects

Milestone	**Delivery date**	**Delivery objects**
Milestone one	01.10.2014	• System specification • Sketch software architecture • Short presentation CI environment • Draft risk evaluation
Milestone two	22.10.2014	• Elaborated software architecture • Tested code of test framework (tbd: which components) • Interface definition for justext/boilerplate components • HTML test data
Milestone three	05.11.2014	• Working test environment with both justext and boilerplate components integrated
Milestone four	19.11.2014	• Evaluation environment for output data of test framework • First approach to new algorithm
Milestone five	08.12.2014	• Implementation of new algorithm • Final documentation • Final presentation

A.5 Milestone one - m1

- Closing date date: 1.10.2014
- Available time: ca. 39h

Working package	**Shortcut**	**Estimated time**
Planning	s1	4h
Research HTML / Algorithms	s2	8h
System specification	s3	12h
Risk evaluation	s4	3h
Draft software architecture	s5	8h
Configuration CI environment	s6	4h
Total		39h

A.5.1 Stories m1

Title	Planning
Id	s0
Estimated time	4h
Description	As a project owner, you need to have a time schedule so that you can see when you will achieve which results. The PAWI project is split into several working packages which are then split into single stories. The working packages are assignment to milestones and for each milestone, delivery objects are defined. This can be a document, a piece of test or production code or some other kind of work.

Title	Research HTML / Algorithms
Id	s1
Estimated time	8h
Description	My knowledge of HTML and content extraction algorithms is still limited. In order to find out what challenges I will face and which aspects I will have to take into consideration for performing the first tasks, a short research on these topics is needed.

Title	System specification
Id	s2
Estimated time	12h
Description	The PAWI project is defined through a short project description. This description does not cover all necessary information to both plan and perform this project. The key features, interfaces and delivered objects have to be defined more closely. The system specification should cover all these requirements.

Title	Draft software architecture
Id	s3
Estimated time	8h
Description	A first rough software architecture should be made as soon as possible, so that any misunderstandings between tutors and student can be uncovered. Moreover, it is much easier to plan the further steps when the software is split into several parts.

Title	Risk evaluation
Id	s4
Estimated time	8h
Description	Potential risks should be uncovered with the knowledge that was gathered by defining the specification and the software architecture. What is more, further actions can be defined to minimize the above mentioned risks.

Title	Configuration CI environment
Id	s5
Estimated time	4h
Description	To deliver high quality software a continuous integration environment is required. Following tools should be evaluated and configured for further use: • Version control (git) • Project build automation tool (gradle) • continuous integration service (Travis CI)

A.6 Milestone two - m2

- Closing date date: 22.10.2014
- Available time: ca. 45h

Working package	**Shortcut**	**Estimated time**
Implementation test framework	s6	20h
Prototype Integration of justext/boilerpipe	s7	17h
Collection of test data	s8	8h
Total		45h

A.6.1 Stories m2

Title	Implementation Config Reader
Id	s6
Estimated time	4h
Description	The configuration for the test framework is located in a text file in the resources folder of the project. The data is formated in a key value structure. This text file is read at the startup of the program and saved in a Config object.

Title	Implementation File Reader
Id	s7
Estimated time	6h
Description	The html and content files are located in two folders (content/html) in the resources folder of the project.. For each file pair with the same name, a test object is generated and the content of the file is read and put into the test objects.

Title	Implementation File Writer
Id	s8
Estimated time	4h
Description	The results of a test is written in an output text file into the resources folder of the project.

Title	Implementation Test Manager
Id	s9
Estimated time	6h
Description	The Test Manager contains the business logic of the program and coordinates the reading, testing and writing.

Title	Prototype Integration of boilerpipe
Id	s10
Estimated time	4h
Description	Implementation of a small prototype which uses the existing implementation of boilerpipe. A final interface for boilerpipe is defined for further use.

Title	Prototype Integration of justext
Id	s11
Estimated time	12h
Description	Implementation of a small prototype which uses the existing implementation of justext. A final interface for justext is defined for further use.

Title	Collection of test data
Id	s12
Estimated time	8h
Description	To evaluate the functionality of the text extraction algorithms, a certain amount of test data is needed. This test data contains HTML files of several web pages. The HTML code is categorized into content and boilerplate.

A.7 Milestone three - m3

- Closing date date: 5.11.2014
- Available time: ca. 30

Working package	**Shortcut**	**Estimated time**
Implementation test framework	s13	20h
Final integration of justext / boilerplate	s10	10h
Total		30h

A.7.1 Stories m3

Title	Implementation text comparator analyzer
Id	s13
Estimated time	12h
Description	Implement the analyzer component. The content text and the extracted text by an algorithm needs to be compared and the values TP, FP, TN, FN need to be calculated.

Title	Implementation math analysis
Id	s14
Estimated time	6h
Description	After getting the values for TP, FP, TN, FN further values need to be calculated. These values are Precision, Recall, F-Measure and Fallout.

Title	Final implementation of justext and boilerpipe
Id	s15
Estimated time	6h
Description	After implementing prototypes for both Justext and boilerpipe, they need to be implementend in the main application so that test cases can be extracted by the algorithms.

A.8 Milestone four - m4

- Closing date date: 19.11.2014
- Available time: ca. 30h

Working package	**Shortcut**	**Estimated time**
Evaluation environment for results	s11	20h
Research on new algorithm	s12	10h
Total		30h

A.8.1 Stories m4

Title	Evaluation of results
Id	s16
Estimated time	8h
Description	The test framework does produce a lot of output data. To investigate the results more closely, the data needs to put into some form that specific information can be extracted from it. To do so, an excel sheet is needed, where the output data from the test framework can be imported into excel with ease.

Title	Improve the text analysis
Id	s17
Estimated time	8h
Description	Right now the text analysis is not very precise. Mr. Kaufmann suggested to use a diff tool to compare the text files with each other. So a diff library needs to be found, tested and integrated into the existing application.

Title	Add a test case filter
Id	s18
Estimated time	8h
Description	Investigating a specific test is not possible yet. To do so some kind of filter is needed. It should be possible to set a test case for inspection in the config file and as a result, an extra output file with information about this test case should be generated.

Title	Research on new algorithm
Id	s19
Estimated time	20h
Description	A short research should be done on the idea of the RSS algorithm. Based on this research it should be possible to decide, if we are going to implement it or not.

A.9 Milestone five - m5

- Closing date date: 8.12.2014
- Available time: ca. 38h

Working package	**Shortcut**	**Estimated time**
Implementation of new algorithm	s13	19h
Complete documentation	s15	15h
Prepare final presentation	s15	4h
Total		38h

A.9.1 Stories m5

Title	Adapt the Justext algorithm for block extraction
Id	s20
Estimated time	6h
Description	Right now only the calculated values for True Positive, Precision, etc. are available in the detailed result report. To investigate the results more closely, more information is needed which is the classification from the algorithms for the single HTML blocks. To do so, the Justext algorithm needs to be adjusted so that it is possible to get the classification information for each HTML block.

Title	Adapt the Boilerpipe algorithm for block extraction
Id	s21
Estimated time	6h
Description	Right now only the calculated values for True Positive, Precision, etc. are available in the detailed result report. To investigate the results more closely, more information is needed which is the classification from the algorithms for the single HTML blocks. To do so, the Boilerpipe algorithm needs to be adjusted so that it is possible to get the classification information for each HTML block.

Title	Integrate the adapted algorithms accordingly into the main application
Id	s22
Estimated time	6h
Description	After adapting the algorithms that it is possible to extract the block classification information, the main application needs to be adapted that it can handle the block information and put it into the output file when needed.

Title	Complete documentation
Id	s23
Estimated time	15h
Description	Complete and review all chapters of the documentation.

Title	Prepare final presentation
Id	s24
Estimated time	4h
Description	Prepare the final presentation and the final printed / digital version of the thesis.

A.10 Time tracking

To keep track of the time, the tool toggl was used. Toggl is an easy to use time tracking tool. In following table, all the performed tasks and the needed time are listed.

TODO: put table here

Appendix B

Risk Analysis

B.1 Version

Version	Date	Change	Author
0.1	20.09.2014	Setup document	JR
0.2	28.09.2014	Add risks, evaluation, consequences	JR
1.0	05.10.2014	Grammar, layout	JR

B.2 Introduction

B.2.1 Purpose

This document evaluates and calculates all possible risks and defines actions that can minimize these risks as well as possible.

B.3 Risk evaluation

B.3.1 Unclear requirements

The start of a project is normally no easy task because its requirements are vaguely known. If they are not well defined as soon as possible, the requirements will stay vague throughout the whole project, which can lead to a disaster.

B.3.2 New technologies

The new technologies which are present in this project are the following:

- Gradle
- Travis CI
- Python

Each of them brings its own risk.

B.3.3 Integration Boilerpipe

The Boilerplate algorithm needs to be integrated into the text extraction framework. Every interface of an external component is a possible risk factor.

B.3.4 Integration Justext

The Justext algorithm needs to be integrated into the text extraction framework. Every interface of an external component is a possible risk factor.

B.3.5 Implementation RSS algorithm

The development and implementation of a new algorithm is predestined to generate risks.

B.4 Assessment of risks

Risk	**Impact**	**Probability of occurrence**	**Risk factor**
Unclear requirements	2	4	8
New technologies	3	3	9
Integration Boilerpipe	5	1	5
Integration Justext	5	5	20
Implementation RSS algorithm	1	5	5

B.5 Consequences

B.5.1 Unclear requirements

As I am working with the client everyday, it is very easy to prevent misunderstandings by communicating with the client as soon as any difficulty appears. Nonetheless, misunderstandings can occur between student and expert. In order to prevent this, it is necessary to have a document defining the requirements as soon and as exact as possible. This will be done in the form of the system requirement specification in the first mile stone. Possible ambiguities can be clarified at the first milestone meeting.

B.5.2 New technologies

It is important to do prototyping with new technologies in the first phase of the project to eliminate these risks as soon as possible.

Gralde and Travic CI are needed in the first milestone to launch the programming environment. If there is any problem it will occur in a very early stage of the project and a possible solution can be found.

B.5.3 Integration Boilerpipe

This risk is rated much lower than the Justext interface because its implementation is in Java and it provides a Java API. Nevertheless, a prototype should be done as soon as possible to prevent any unwelcome surprises with the interface.

B.5.4 Integration Justext

This aspect is classified as the highest risk of all. This is because the implementation happens in Python and it is not clarified yet how it will be integrated into the text extraction framework. An analysis of a possible solution with prototypes needs to be done as soon as possible.

Possible solution are:

- jython (http://www.jython.org)
- Implementation in Java
- Java Processor Interface

B.5.5 Implementation RSS algorithm

This risk has a very high probability of occurrence because it is very likely that a development and an implementation of a new algorithm will cause problems. There is no real solution to that risk. However, because this requirement is noncompulsory, the impact on the outcome of the project is very low. Furthermore, Patrik Lengacher, the tutor of this project, is very experienced in this subject area and will be able to assist if any problems occur.

Appendix C

Software Requirement Specification

C.1 Version

Version	Date	Change	Author
0.1	20.09.2014	Setup document	JR
0.2	28.09.2014	Add features	JR
0.3	30.09.2014	Change features, grammar, layout	JR
0.4	02.10.2014	Add overview of application/evaluation	JR
0.5	04.10.2014	Corrections evaluation	JR
1.0	05.10.2014	Grammar, layout	JR
1.1	20.11.2014	Fix FN definition	JR

C.2 Introduction

C

C.2.1 Purpose

The software requirement specification is providing all needed information to develop the context extraction framework and define all delivery objects. All interfaces to external components, input and output data, deployment considerations and quality attribute are well defined within this document.

C.2.2 Scope

The context extraction framework will perform automated text extraction on a set of HTML test data with two to three different text extraction algorithms. After measuring the performance of each algorithm, an output file with the measured results is generated.

C.3 General description

C.3.1 Operating Environment

The operation environment for the text extraction framework is defined in this section.

C.3.1.1 Local environment

Ubuntu	12.04
JDK	1.7.X
Gradle	1.11
Eclipse Keppler	2.X
git	1.9.X
python	2.7.X

C.3.1.2 Continuous Integration Environment

Ubuntu	12.04
Open JDK	1.6.X
Open JDK	1.7.X
Oracle JDK	1.7.X
Oracle JDK	1.8.X
Gradle	2.0
Travis CI	

C.3.2 Design and Implementation Constraints

C.3.2.1 User interface

As parts of the text extraction framework may be implemented in a server environment at a later point in time and a user interface is not desired from the client, there will be

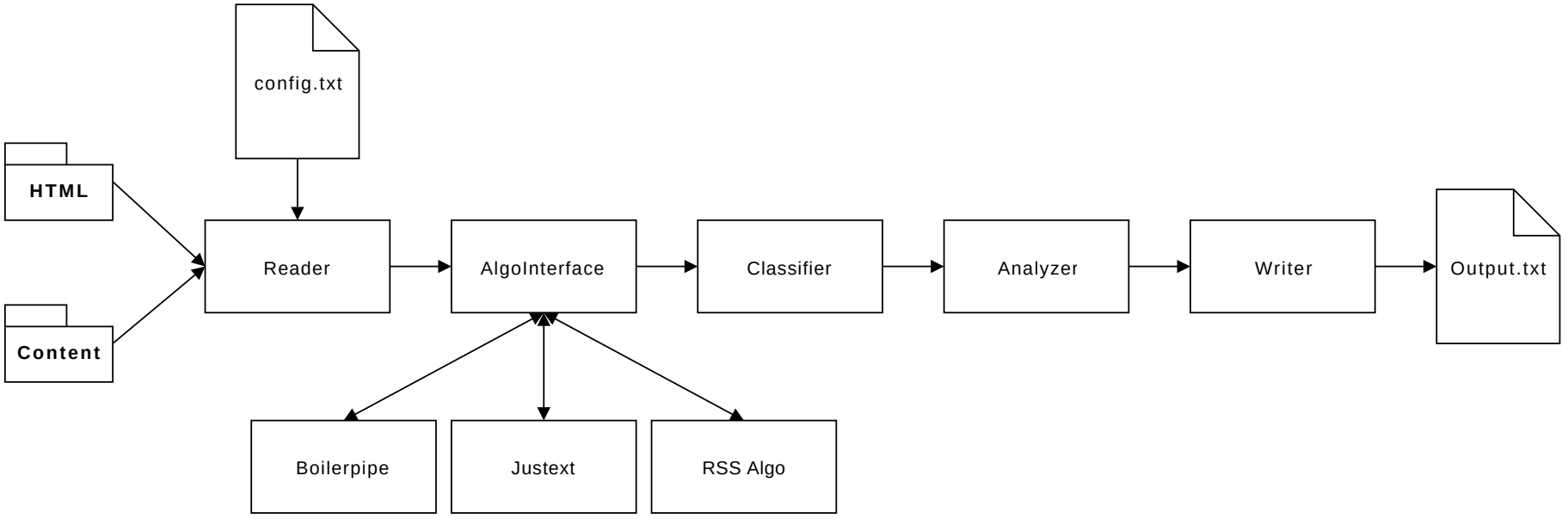
no graphical user interface. The application is built, deployed and started by gradle. While the application is running, no interaction is needed. [**?**] [**?**]

C.4 System Features

This section specifies all system features. Each feature is specified more closely with multiple user stories. However, all the important information, such as external dependencies and output files, are defined in this chapter. The related user stories for each feature are located in the planning section.

C.4.1 Basic functionality

The following diagram and text describes the basic functionality of the application.



There are two folders defined by the configuration file (config.txt). The HTML folder contains HTML files of web pages. The content folder contains text files with the relevant content of the related HTML files. As soon as a test is started, the HTML file and the text file are read and the HTML file is extracted and classified with all the available algorithms. The result of the classification is then compared to the relevant content and performance data is generated. This performance data is then analyzed with statistical methods.

C.4.2 Overview

ID	Name	Chapter	Relevance
f1	Read configuration	C.4.2.1	needed
f2	Create test	C.4.2.2	needed
f3	Integration JusText algorithm	C.4.2.3	needed
f4	Integration Boilerpipe algorithm	C.4.2.4	needed
f5	Evaluation and Implementation RSS feed algorithm	C.4.2.5	nice to have
f6	Evaluation of classification text	C.4.2.6	needed
f7	Evaluation of classification blocks	C.4.2.7	nice to have
f8	Analyze data	C.4.3	needed

C.4.2.1 Read configuration

Name	Read configuration
Feature id	f1
Description	The text extraction framework is configurable with an external text file. The configuration file will contain following items: • Path to folder with HTML files • Path to folder with text files • Path to folder with output files • Configuration for algorithms • etc. The configuration file location is defined as a relative path to the source directory and structured in a key value list: `key:value;` `key:value;` `key:value;`
Relevance	needed
Related stories	tbd

C.4.2.2 Create test

Name	Create test
Feature id	f2
Description	A test contains two input files which are an HTML file and a text file. They are located in the directories defined by the configuration. As soon as the test framework finds an HTML and a text file with the same name, the files are read and the test is started.
Relevance	needed
Related stories	tbd

C.4.2.3 Integration JusText algorithm

Name	Integration JusText algorithm
Feature id	f3
Description	JusText is implemented in python. That is the reason why a service is needed to call the python script and get the extracted text or the extracted blocks.
Relevance	needed
Related stories	tbd

C.4.2.4 Integration Boilerpipe algorithm

Name	Integration Boilerpipe algorithm
Feature id	f4
Description	Boilerplate is implemented in Java. An interfae is needed in order to call the Boilerplate component and get the extracted text or the extracted blocks.
Relevance	needed
Related stories	tbd

C.4.2.5 Evaluation and Implementation RSS feed algorithm

Name	Evaluation and implementation RSS feed algorithm
Feature id	f5
Description	The basic idea of the RSS feed algorithm is to match the content of an HTML document with the related RSS feed and in doing so, define the relevant content. This needs to be evaluated, implemented and integrated into the text extraction framework.
Relevance	nice to have
Related stories	tbd

C.4.2.6 Evaluation of classification text

Name	Evaluation of classification
Feature id	f6
Description	All the text extraction algorithms return an extracted document as text. This document needs to be checked for accuracy, which is achieved by comparing the result of the algorithms with the actual content. • Check each classified block from the algorithms if its content can be found in the actual content • Categorize text as boilerplate or content • Insert results in an output text file Both the evaluation and classification are defined in more detail in section C.4.4.
Relevance	needed
Related stories	tbd

C.4.2.7 Evaluation of classification blocks

Name	Evaluation of classification blocks
Feature id	f7
Description	A more detailed evaluation of the algorithms could be done if not only the text but also each block of an HTML file is classified. So as to achieve the more detailed evaluation, the implementation of JusText and Boilerpiple has to be adapted so that they return classified blocks instead of the extracted text. These blocks are afterwards compared with the actual content and classified. • Check each classified block from the algorithms if its content can be found in the content file • Categorize all blocks as boilerplate or content • Insert the results in an output text file (structure output file: tbd) Both the evaluation and classification are defined in more detail in section C.4.4.
Relevance	nice to have
Related stories	tbd

C.4.3 Analyze data

Name	Analyze data
Feature id	f8
Description	From the results of the comparison further values can be evaluated for a better understanding of the results. These values are described in more detail in section C.4.5.
Relevance	needed
Related stories	tbd

C.4.4 Evaluation of classification

The general meaning of the expressions true positive, true negative, false positive and false negative related to the text extraction topic is shown in following table.

When the results are compared based on words, the expressions are interpreted as follows.

	Classified as content	**Classified as boilerplate**
Actual content	True positive (TP)	False negative(FN)
Actual boilerplate	False positive (FP)	True negative (TN)

	Classified as content	**Classified as boilerplate**
Actual content	Word classified as content by algorithm and is content	Word classified as boilerplate by algorithm but is content
Actual boilerplate	Word classified as content by algorithm but is boilerplate	Word classified as boilerplate by algorithm and is Boilerplate

When the results are compared based on HTML blocks, the expressions are interpreted as follows.

	Classified as content	**Classified as boilerplate**
Actual content	Block is classified as content by algorithm and is content	Block is classified as boilerplate by algorithm but is content
Actual boilerplate	Block is classified as content by algorithm but is boilerplate	Block is classified as boilerplate by algorithm and is boilerplate

In conclusion, TP and FN is the correct outcome of the algorithm i.e. content classified as content and boilerplate as boilerplate. On the other hand, TN and FP is the wrong outcome of the algorithm i.e. content classified as boilerplate and boilerplate as content.

C.4.5 Analytical values

In this paragraph we use the notion of instance instead of word/block. The results of the comparison deliver basic characteristics which can be used to calculate statistical values which help you analyze the test outcome.

Sensitivity / Recall /True positive rate / TPR / Hitrate

Recall is the probability that content is actually classified as content which in our case is

$$Recall = \frac{TP}{TP + FN} \tag{C.1}$$

correct classified content objects divided by correct classified objects.

Precision / True negative rate / TNR

Precision is the probability the as content classified objects are actually content which in our case is

$$Presicion = \frac{TP}{TP + FP} \tag{C.2}$$

correct classified content objects divided by the sum of all objects classified as content.

F-measure / F1-score / F-score

F-measure is the harmonic mean of precision and recall which in our case is

$$Fmeasure = 2 * \frac{presicion * recall}{presicion + recall} \tag{C.3}$$

a measure of the test's accuracy.

Fallout / False positive rate / FPR

Fallout is the proportion of non-relevant objects that are retrieved out of all non-relevant objects available which in our case is

$$Fallout = \frac{FP}{FP + TN} \tag{C.4}$$

C.5 External Interface Requirements

C.5.1 Boilerpipe

The boilerpipe algorithm is implemented in Java and the documentation is found under https://code.google.com/p/boilerpipe/.

C.5.2 JusText

The JusText algorithm is implemented in python and the documentation is found under https://code.google.com/p/justext/. It is not yet defined how it will be integrated into the text extraction framework. See risk analysis for further information.

Appendix D

Software Architecture

D.1 Version

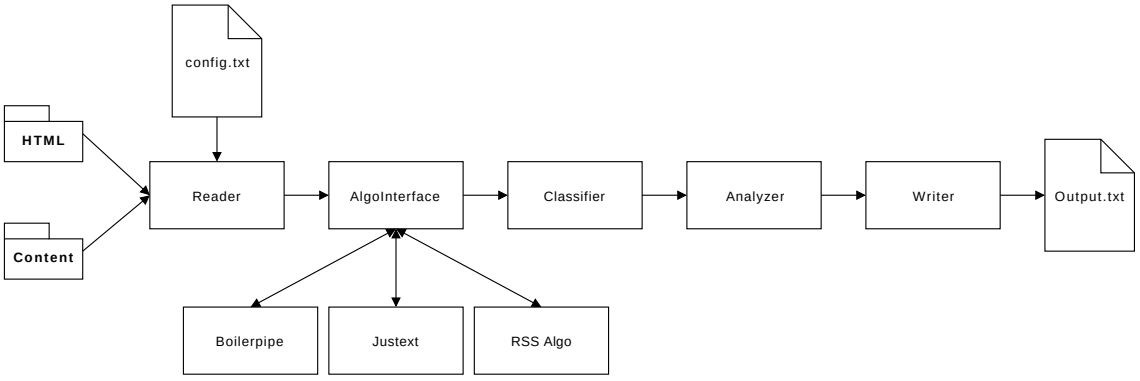
Version	Date	Change	Author
0.1	8.10.2014	Setup document	JR
0.2	12.10.2014	Class diagrams	JR
0.3	16.10.2014	Text	JR
0.4	20.10.2014	Activity diagram	JR
1.0	21.10.2014	Grammar, Diagram fixes	JR
1.1	22.10.2014	Adding description of all business logic classes, approach and extended introduction	JR
1.2	23.10.2014	Grammar	JR,LR

D.2 Introduction

This document describes the software architecture of the context extraction test framework. The context extraction test framework will perform automated text extraction on a set of HTML test data with two to three different text extraction algorithms. After measuring the performance of each algorithm, an output file with the measured results is generated.

D.3 System Overview

This section describes the approach for elaborating the software architecture and gives an overview over the software architecture. The following figure from the software requirement specification is the basis for elaborating the software architecture.



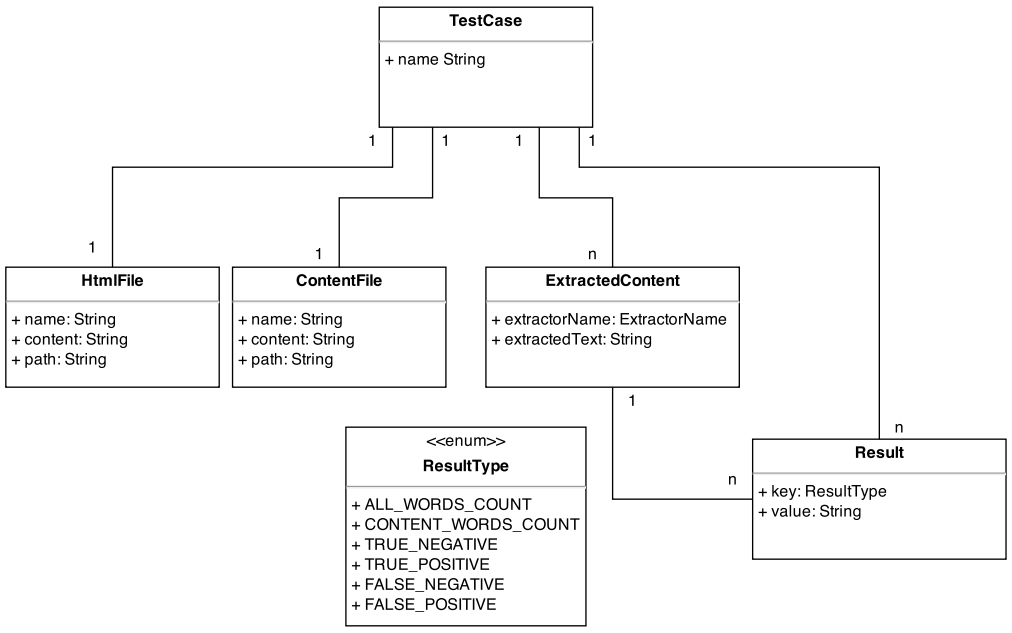
First all possible entities in the domain are identified and a data model is elaborated. The data model is described in section D.4.1. Then the business logic is researched based on the figure above. With this knowledge, the five packages main, reader, classifier, analyzer and writer are defined. The functions for each package is then evaluated and split into single classes. During the implementation of the first approach, some classes are adapted due to unforeseen circumstances. The outcome is described in section D.4.7.

D.4 Logical view

This section describes how the system is structured in terms of units of implementation.

D.4.1 Data model

The following diagram shows the data model of the application.



D.4.2 TestCase

A TestCase object is generated for each HTML/content file pair in the input folders. A TestCase has a a name which is unique and which matches the name of the content and HTML file.

D.4.3 HtmlFile

Each TestCase has an HtmlFile object. It contains the content of the actual HTML file as String and the file path.

D.4.4 ContentFile

Each TestCase has a ContentFile object. It contains the content of the actual text file as String and the file path.

D.4.5 ExctractedContent

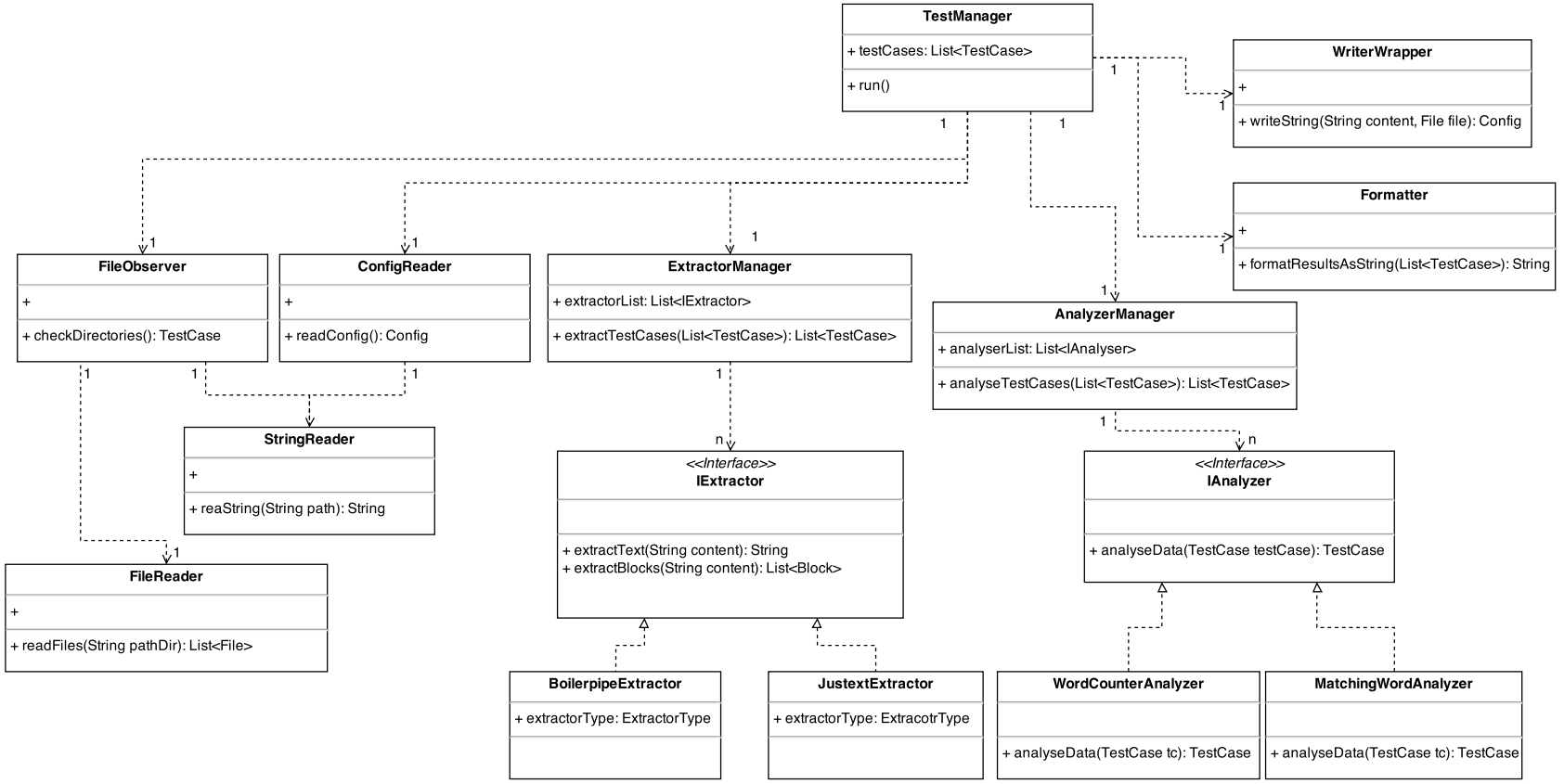
Each TestCase can have multiple ExctractedContent objects. Each of them represents a result of a content extraction from an extractor such as JusText or Boilerpipe.

D.4.6 Result

A TestCase or an ExctractedContent object can have Result objects. The results are key value pairs which represent analytical data. An example for a Result related to a TestCase would be the word count of the content file, which is generally valid. An example for a Result related to an ExctractedContent would be the word count of true negative words, which is only valid for one specific ExctractedContent.

D.4.7 Business Logic

The following diagram shows the business logic of the application. The diagram does not show all of the classes but the most important ones.



D.4.8 Description of single classes

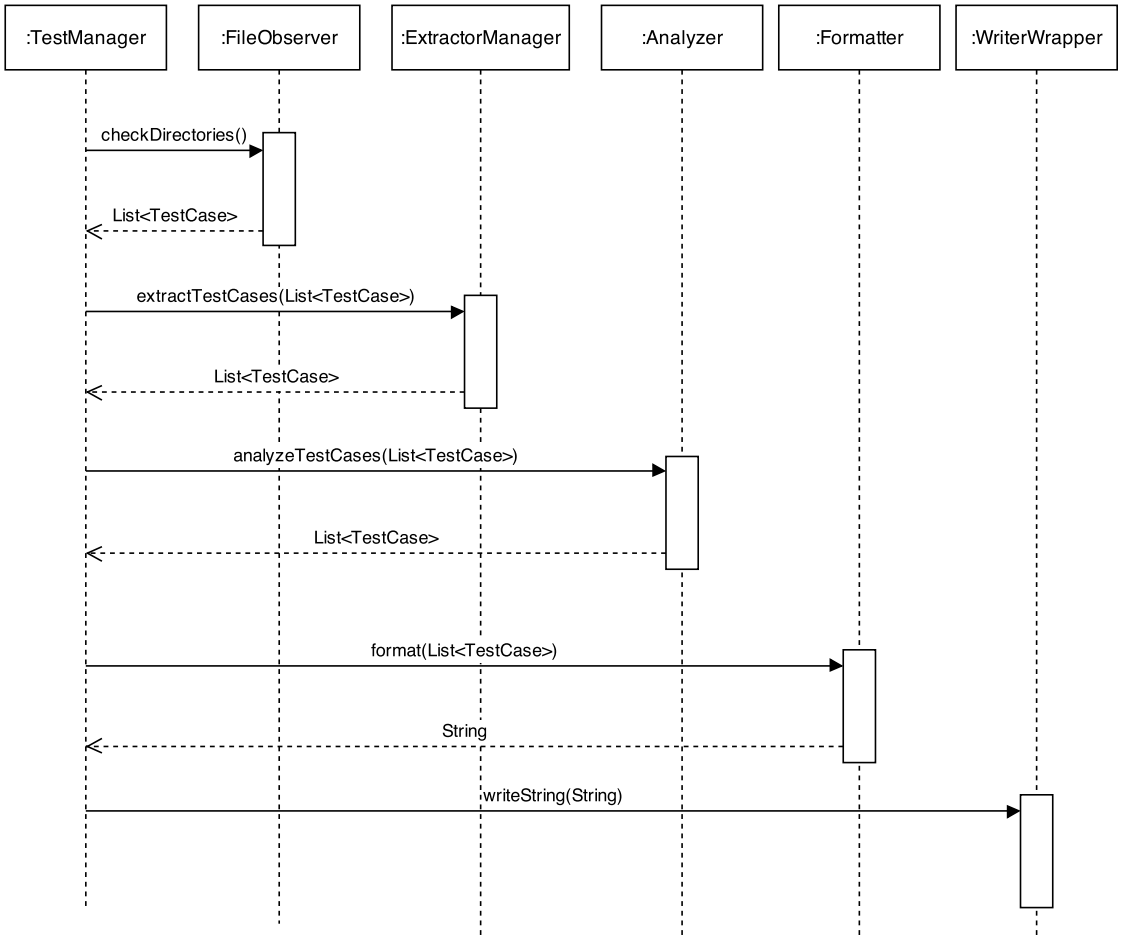
Class	Package	Description
TestManager	testManager	The TestManager class manages the whole business logic that manages TestCase objects through the whole test process from reading the file content to writing the test results into an output file.
FileObserver	reader	The FileObserver class checks the HTML and content directory for files of the same name and creates TestCases from each found pair. The folders are checked with the FileReader class and the content of the files are read with the StringReader class.
StringReader	reader	The StringReader class reads a text file and returns the content as String. The class is made for easier mocking of the BufferedReader so that testing of other classes which are dependent on external files becomes much easier.
FileReader	reader	The FileReader class returns a File objects for each found file in a directory given by a parameter.
ExtractorManager	classifier	The ExtractorManager manages all available Extractors. Each extractor which is used for the actual test must be initialized in this class and added to the ExtractorList. Each TestCase is then extracted by every IExtractor in the ExtractorList.
IExtractor	classifier	The IExtractor is the interface to the different extractor. The interface is very lightweight. The parameter is the text that should be extracted and the return value is the extracted text.
BoilerpipeExtractor	classifier	The BoilerpipeExtractor implements the IExtractor and is the interface to the Boilerpipe package. It handles all dependencies on the Boilerpipe package and returns the extracted content as a string.

JusTextExtractor	classifier	The BoilerpipeExtractor implements the IExtractor and is the interface to the JusText python program. The Java ProcessBuilder is used to create operating processes. One can then perform operating system commands and run the python script. The python script creates a text file with the extracted content which is read by the JusTextExtractor class and returned as String.
AnalyzerManager	analyzer	The AnalyzerManager manages all available analyzers. Each analyzer which is used for the actual test must be initialized in this class and added to the AnalyzerList. Each TestCase is then analyzed by every IAnalyzer in the AnalyzerList.
IAnalyzer	analyzer	The IAnalyzer interface is a simple interface to the different analyzers. Each analyzer can generate one or more Result objects.
WordCounterAnalyzer	analyzer	The WordCounterAnalyzer is a simple analyzer which counts all words of the content file, the HTML file and each extracted content.
MatchingWordAnalyzer	analyzer	The MatchingWordAnalyzer compares the content file with each extracted content and calculates the values for true positive, true negative, false positive and false negative.
Formatter	writer	The Formatter class formats a string from all TestCase objects as a CSV file structure. This means that a table with the Result keys as table header for each column is created. For each TestCase a row with the Result value field as values is added to the table. The outcome is a CSV file which can easily be imported into another program such as Excel so that one can work with the data.

WriterWrapper	writer	The writer wrapper writes a string into a text file. It is used to wrap the BufferedWriter so that mocking and testing of dependent classes is easier.
ConfigReader	reader	The ConfigReader class reads the config.txt file and puts the key value pairs into a HashMap.

D.5 Process view

This section describes the dynamic aspects of the application.



The data model is passed through the business logic and is enriched with data during the test procedure. First the input directories are checked for files by the FileObserver and TestCases are generated for each file pair with the same name. Then all the TestCases are then handed over to the ExtractorManager. The ExtractorManager extracts each TestCase with all available implementations of IExtractor and puts the Results into

ExtractionResults. After that, all the TestCases are handed over to the Analyzer which runs each implementation of IAnalyzer. Each IAnalyzer produces at least one Result and puts it into the TestCase. To simplify the diagram, only two Analyzers are drawn. After generating some Results, the Formatter serializes the Result Objects into a String as a CSV table and the WriterWrapper persists the CSV data into an output file.

D.6 Development view

This chapter describes the used tools and libraries which are used during the development process as well as the continuous integration process.

D.6.1 Continuous Integration Process

The continuous integration process for this process is handled with the version control tool git, the continuous integration server Travic CI and the project automation tool Gradle. The single tools are described in more details in the following sections. The general process is however defined like this. Since this is a one man project, there is only one git branch present for the most time of the development phase. The application is developed on the local branch and unit tests are written for critical components. The application is then built locally with Gradle. Possible problems are solved and the changes are pushed on the master git repository. As soon as there are any new commmits on the master repository, the application is built as well on the Travic CI server. If the build is not successful. An error report is sent to the user.

D.6.2 git

Git is a distributed version control system. Unlike other version control systems like Subversion, git is not using a central server but each user has his own copy with the complete history on the local system. It is much easier to work with additional branches or tags. Because of these advantages and because Layzapp is working with it as well, git was chosen to use for this project for the code sources and as well for the documentation. The resources are open source and are available under following links.

- code: https://github.com/heya87/pawiTwo
- documentation: https://github.com/heya87/pawi_doc

D.6.3 Gradle

Gradle is a project automation tool which uses a Groovy-based domain-specific language (DSL) instead of the more traditional XML form of declaring the project configuration. All the dependencies of the project are handled with Gradle and it is very easy to deploy on a new system. The user only needs to download the git repository and can build the project with the Gradle wrapper without installing any new software. All dependencies are then downloaded automatically and the user can start working without caring about missing packages. The build process is defined in the build.gradle file which is located in the source directory of the project.

D.6.4 Travic CI

Travic CI is an open source build server. It is easy to use in combination with git. One only needs to add a configuration file in the source directory of the project and define the git repository. Afterwards the project is build for every change. If the build does not pass, a mail is sent to the user. The actual status of the build can be found under following link https://travis-ci.org/heya87/pawiTwo.

D.6.5 JusText

An implementation of the JusText algorithm is available in Python. The ressources are available under following link https://code.google.com/p/justext/. There is no implementation in Java so for the first approach the python application is called from Java with a ProcessBuilder. The command to extract an HTML file with justext is very simple:

```
justext -s English /path/page.html > cleaned-page.txt
```

This command extracts the HTMl file page.html into a text file called cleaned-page.txt. The Java ProcessBuilder performs system commands as they are used in a console and can handle the outcome if needed. For the project there is no return values needed. The needed content is the generated text file which is then read and processed for further use.

D.6.6 Boilerpipe

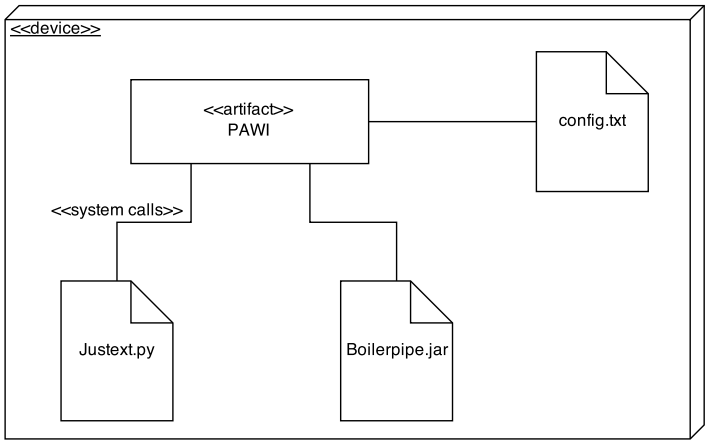
An implementation of the Boilerpipe Algorithm is available in Java. The resources are available under following link https://code.google.com/p/boilerpipe/. This algorithm can be used out of the box with calls against the Java API.

D.6.7 diff-match-patch

The Java library diff-match-patch [**?**] is used to compare text files and find differences. For this file, it is used to extract the values TP, TN, FP, FN when comparing the actual content files with the outcome of the algorithms.

D.7 Physical view

Following diagram shows the physical view of the application. The application is running on a single device, which results in a very simple diagram. The external components are the Boilerpipe algorithm implemented in Java and the JusText algorithm implemented in Python. The python program is called with system calls. All the components are located on the same device. The config.txt file contains configuration parameter for the application and the different output text files contain the results of the test framework.



Appendix E

Test plan

E.1 Introduction

E.1.1 Test concept

The project was tested on two levels. The first level is Unit Testing in Java. This tests are not documented in the report itself. Since the project was developed in a continuous integration environment, all unit tests were run after committing any changes to the git repository. If any test did not pass, the build would fail. With this functionality, the most important and critical functions of the functionalities are already covered. The continuous integration environment is described in the system requirement specification (C). However the second level are test cases which are defined against the software requirement specification and are performed as blackbox test. To do so, test cases were defined so they cover all the features defined in the software requirement specification.

E.1.2 Test cases

This section contains all test cases which cover the features described in the software requirement specification. The results of the single test runs can be found in the test protocol (F).

Name	Test case handling
Test case id	tc1
Description	Create 10 tests and check the detailed output file for the right amount of tests.
Related features	f2
Test steps	1. Put ten text files into the content input folder. 2. Put ten html files into the html input folder. They need to have the same name as the related content files. 3. Run the test
Preconditions	At least one extractor needs to be activated in the config file.
Preconditions	An output file óutput_detailed.txtńeeds to be created. The text file needs to contain exactly ten result lines, for each test one line. The tests need to be named the same as the input files in the content resp. the html folder.

Name	Config single test
Test case id	tc2
Related features	f1, f2
Description	The detailed results for a single test need to be put into an extra output file when it is defined in the config file.
Test steps	1. Add following line to the config file: "inspect:testToInspect" where testToInspect is a file name of any content/HTML file pair in the input folder. 2. Run the test 3. Check the outputfolder for following file: "result_testToInspect.txt"
Preconditions	tc1 needs to be fulfilled
Acceptance criteria	A file with the name "result_testToInspect.txt" is present in the output folder.

Name	Run the boilerpipe algorithm with the test framework
Test case id	tc3
Related features	f4
Description	Check if the test framework is working with the boilerpipe algorithm.
Test steps	1. Add any content/HTML file pair into the input folders. 2. Add following line into the config file: "extractor:boilerpipe" 3. Run the test 4. Check the output files for results produced for the boilerpipe extractor.
Preconditions	tc1/tc2 need to be fulfilled
Acceptance criteria	The outputfile output.txt needs to have results for the boilerpipe extractor.

Name	Run the Justext algorithm with the test framework
Test case id	tc4
Related features	f3
Description	Check if the test framework is working with the Justext algorithm.
Test steps	1. Add any content/HTML file pair into the input folders. 2. Add following line into the config file: "extractor:justext" 3. Run the test 4. Check the output files for results produced for the Justext extractor.
Preconditions	tc1/tc2 need to be fulfilled
Acceptance criteria	The output file output.txt needs to have results for the Justext extractor.

Name	Test analysis
Test case id	tc5
Related features	f6, f8
Description	Run a test with one test file and check the results (TP, TN, FP, FN). To do so, run a detailed test (needs to be defined in the config file) for the according test and check the results by hand. The contains the actual content file as well as the extracted files by the algorithms. This output file can be used to check the results.
Test steps	1. Put an HTML file into the HTML input folder. 2. Put an according content file into the content input folder. 3. Add following line into the config file: "inspect:testToInspect" where "testToInspect" accords to the name of the test files. 4. Run the test 5. Open the file results_testToInspect.txt 6. Check the values for TP, TN, FP, FN by hand
Preconditions	tc2 needs to be fulfilled
Acceptance criteria	The results in the output file for TP, TN, FP, FN need to be correct.

Name	Evaluation of Precision, Recall, F-Measure, Fallout
Test case id	tc6
Related features	f6, f8
Description	Check the calculated values Precision, Recall, F-Measure and Fallout for any test case.
Test steps	1. Put any test file pair into the input folders. 2. Add following line into the config file: "inspect:testToInspect" where "testToInspect" accords to the name of the test files. 3. Open the file test_testToInspect.txt. 4. Calculate the values for Precision, Recall, F-Measure and Fallout from the TP, FP, TN, FN values and compare them with the values in the output file.
Preconditions	tc1, tc5
Acceptance criteria	The values Precision, Recall, F-Measure and Fallout need to be calculated correctly.

Name	Block data evaluation
Test case id	tc7
Related features	f7
Description	Check that the detailed results contains the classification information from each algorithm.
Test steps	1. Put any test file pair into the input folders. 2. Add following line into the config file: "inspect:testToInspect" where "testToInspect" accords to the name of the test files. 3. Open the file test_testToInspect.txt. 4. Check for the block information
Preconditions	tc1, tc3, tc4
Acceptance criteria	The detailed output file for one test needs to contain the classification information for each block the algorithms classified.

Appendix F

Test protocol

F.1 Introduction

This section contains the results of the black box test which were made three times during the projects. The test cases are described in the test plan (E). Each test case was tested as described and the results were checked and documented. All the three test runs are documented below.

F.1.1 Test run one

- Date: 24.10.2014
- Test engineer: Joel Rolli

Test case id	**Restult**	**Comment**
tc1	pass	-
tc1	pass	-
tc2	pass	-
tc3	fail	only prototype working
tc4	fail	only prototype working
tc5	fail	Working in general but the results are not very exact (comparison is done based one single word counting)
tc6	fail	not implemented
tc7	fail	not implemented

F.1.2 Test run two

- Date: 25.11.2014
- Test engineer: Joel Rolli

Test case id	Restult	Comment
tc1	pass	-
tc1	pass	-
tc2	pass	-
tc3	pass	-
tc4	pass	-
tc5	pass	-
tc6	pass	-
tc7	fail	not implemented

F.1.3 Test run three

- Date: 8.12.2014
- Test engineer: Joel Rolli

Test case id	Restult	Comment
tc1	pass	-
tc1	pass	-
tc2	pass	-
tc3	pass	-
tc4	pass	-
tc5	pass	-
tc6	pass	-
tc7	pass	-

Appendix G

User manual

G.1 Introduction

This document describes how to install and use the content extraction test framework.

G.2 System requirements

To use the text extraction test framework, the software artifacts listed in the following table needs to be installed on your system. These artifacts are all for free and can be downloaded from the links below.

Software	Version	Source
Ubuntu	12.XX	http://releases.ubuntu.com/12.04/
git	2.2.X	http://git-scm.com/download/linux
python	2.7.X	https://www.python.org/download/releases/2.7.6/

G.3 Installation

1. Checkout the repository with git to your favorite directory.

```
git clone https://github.com/heya87/pawiTwo
```

2. Use your favorite console application and navigate into the source directory of the checked out repository.

3. Build the project with the gradle wrapper.

```
./gradlew build
```

G.3.1 Installation JusText

If you want to run the test framework using the justext extractor, you have to install the according python package first.

1. Open your favorite console application.
2. Download the needed resources to your favorite directory.

```
wget http://justext.googlecode.com/files/justext-1.2.tar.gz
```

3. Extract the downloaded files.

```
tar xzvf justext-1.2.tar.gz
```

4. Install the package.

```
cd justext-1.2/
python setup.py install
```

G.4 Configuration & Usage

G.4.1 Configuration

To addapt the application functionalities to your wishes, you can use several configuration items.

G.4.1.1 Classifier

The classifier configuration defines, which classifiers are tested with the test framework. The implemented classifier are JusText and Boilerpipe. To use these classifier, add following lines into the configuration file:

JusText

```
classifier:justext;
```

Note: To run the test framework with JusText, python and the python package justext needs to be installed. Please check G.3.1 for more details.

Boilerpipe

```
classifier:boilerpipe;
```

G.4.1.2 Language

The test framework needs to know which language the test files are written in. The two languages english and german is available. To configure the language, add following lines to the configuration file.

English

```
language:english;
```

German

```
language:german;
```

Note: The default language is English.

G.4.1.3 Inspect

If you want to inspect one test case and get specific details for it, you can do this with the inspect confiugration item. If you have a test file pair in the input folders which are named "testOfInterest" you have to add following line to the configuration file for inspection.

```
inspect:testOfInterest;
```

G.4.1.4 Filter

It is possible to filter the output results for specific values. Several configuration items are needed to set the filter.

Extractor
Defines for which extractor the filter is used. The possible options are:

- justext
- boilerpipe

Filter entity
Defines for which entity the filter is used. The possible options are:

- precision

- recall
- fmeasure
- fallout
- accuracy

Filter type
Defines the filter type. The possible options are:

- min
- max

Filter value
Defines for which value the filter is used. Since the filter entities are all percentages, the filter value can be any float value between 0 and 1. All filters with other values are ignored.

Example
If you want to filter the results for following conditions. All test cases which are tested by JusText and have a precision lower than 50.%

```
filter:justext:precision:max:0.5;
```

G.4.2 Run a test

1. Put the content files into following folder: ~/build/resources/main/content
2. Put the HTML files into following folder: ~/build/resources/main/html
3. Adjust the configuration file to you wishes. The configuration is located at: ~/build/resources/main/config.txt
4. Run the test with the gradle wrapper.

   ```
   ./gradlew run
   ```

5. Find the results in the output folder: ~build/resources/main/

G.4.3 Results

After running a test multiple result files are generated and written in following folder: ~/build/resources/main/

The different result files are:

- **overall_results.txt**: This file contains the average values for Precision, Recall, etc. from all tested files.
- **detailed_results.txt**: This file contains the results for each test.
- **resut_testOfInterest.txt**: If you have defined the test case testOfInterest for further inspection (see G.4.1.3), an additional text file is generated which contains elaborated information about this test.

Appendix H

MS1 meeting report

H.1 Introduction

This document is a short report about the MS1 meeting. The meeting took place on the 1.10.2014.

H.2 Version

Version	Date	Change	Author
1.0	1.10.2014	Setup document / text	JR
1.1	22.10.2014	forgot Manu!! / grammar / rework	JR

H.3 Attendees

- Joel Rolli
- Michael Kaufmann
- Patrick Huber
- Patrik Lengacher
- Manuel Schneider

H.4 Delivery objects

- System specification

- Sketch software architecture
- Short presentation CI environment
- Draft risk evaluation

H.5 Decisions

The risk evaluation and the CI environment are approved. The evaluation of the text extraction was discussed again and it was decided that the evaluation is still done with Words instead of HTML blocks. Doing the evaluation with HTML blocks can still be done but is not part of the PAWI project and would be bonus content. Furthermore it was decided that no handmade test data is needed and the data from cleanEval and Gold standard are used for this project. The draft of the software architecture is ok but needs to be digitalized. The software architecture needs to be extended with additional data about the analysis of the extracted data. Which means all the formula for TN, FP, TP, FN etc. needs to be defined.

H.6 Rework

Following rework needs to be done until the 5.10.2014

- Extended software specification
- Digital version of software architecture sketch

Update: The listed delivery objects were delivered and approved on the 5.10.2014.

Appendix I

MS2 meeting report

I.1 Introduction

This document is a short report about the MS1 meeting. The meeting took place on the 24.10.2014.

I.2 Version

Version	Date	Change	Author
0.1	20.10.2014	Setup document	JR
1.0	27.10.2014	add meeting report	JR

I.3 Attendees

- Joel Rolli
- Michael Kaufmann

I.4 Delivery objects

- Elaborated software architecture
- Tested code of test framework (reader / writer)
- Interface definition for justext/boilerplate components
- HTML test data

I.5 Decisions

All delivery objects are approved. We discussed the text comparison and came to a conclusion that I am going to search for diff tool libraries to do the comparison of text files. Mr. Kaufmann sent me some proposal later on.

I.6 Rework

There is no rework to do. The proposed diff tool libraries are

- https://code.google.com/p/google-diff-match-patch/
- https://commons.apache.org/proper/commons-lang/javadocs/api-2.6/org/apache/commons/lang

The next milestone, which is the implementation of the whole test framework and integration of justext and boilerpipe, is already achieved as well. We decided that MS3 is obsolete and we are going to meet again if there are any ambiguity and if there is not, for MS4.

Appendix J

MS4 meeting report

J.1 Introduction

This document is a short report about the MS4 meeting. The meeting took place on the 21.11.2014.

J.2 Version

Version	Date	Change	Author
0.1	21.11.2014	Setup document	JR
1.0	21.11.2014	add meeting report	JR

J.3 Attendees

- Joel Rolli
- Michael Kaufmann
- Patrick Lengacher

J.4 Delivery objects

- Evaluation environment for output data of test framework
- First approach to new algorithm
- Interface definition for justext/boilerplate components

- HTML test data

J.5 Decisions

All delivery objects are approved. A first draft of the final documentation was reviewed by Mr. Kaufmann. The structure is good in general. However some chapters are renamed and it was decided that the project documents belong into the appendix and are not integrated into the main report. Furthermore, the statistical results which were produced with the application were re- viewed and some parts of the source code were presented. We decided that implementing an extra algorithm is too time consuming for the time left and that the focus for the last milestone is on the documentation, the analysis of the existing algorithms and im- provement of the existing work. One more point was discussed concerning the javadoc. There is no need to do javadoc for each method. It is ok to do javadoc for the important classes and methods. We decided that the final presentation is held on the 11. December if everyone (Michael Kaufmann, Patrick Huber, Patrick Lengacher, Manuel Schneider) is free on this day.

J.6 Rework

No rework needs to be done.

[**?**]

Bibliography

[1] Aidan Finn, Nicholas Kushmerick, and Barry Smyth. Fact or fiction: Content classification for digital libraries, 2001.

[2] Christian Kohlschütter. Boilerpipe algorithm, 2010. URL https://code.google.com/p/boilerpipe/. (Visited on 2.10.2014).

[3] Christian Kohlschütter. Boilerplate detection using shallow text features, 2010. URL http://www.l3s.de/~kohlschuetter/publications/wsdm187-kohlschuetter.pdf. (Visited on 2.10.2014).

[4] Jan Pomik´alek. Justext algorithm, 2011. URL https://code.google.com/p/justext/. (Visited on 2.10.2014).

[5] Jan Pomik´alek. Justext algorithm, 2011. URL http://is.muni.cz/th/45523/fi_d/phdthesis.pdf. (Visited on 2.10.2014).

[6] Wikipedia. Plagiarism — Wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/wiki/Diff_utility. (Visited on 8.12.2014).

[7] Wikipedia. Plagiarism — Wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/wiki/Meld_(software). (Visited on 8.12.2014).

[8] Google. diff-match-patch, 2014. URL https://code.google.com/p/google-diff-match-patch/. (Visited on 4.11.2014).

[9] Cleaneval. http://cleaneval.sigwac.org.uk/. (Visited on 10.10.2014).

[10] Istvan Endredy and Attila Novak. More effective boilerplate removal—the goldminer algorithm, 2013. URL http://polibits.gelbukh.com/2013_48/More%20Effective%20Boilerplate%20Removal%20-%20the%20GoldMiner%20Algorithm.pdf. (Visited on 2.10.2014).