

HOCHSCHULE LUZERN

PAWI

---

# Evaluation of different content extraction algorithms

---

*Author:*  
Joel Rolli

*Supervisor:*  
Patrick Huber / Patrik  
Lengacher

*A thesis submitted in fulfilment of the requirements  
for the degree of some HSLU degree*

*in the*

Research Group Name  
Department or School Name

November 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:

---

HSLU

# *Abstract*

Faculty Name

Department or School Name

some HSLU degree

## **Evaluation of different content extraction algorithms**

by Joel Rolli

Bla Bla Bla Bla Bla Bla Bla Bla Bla Bla Bla Bla Bla Bla Bla Bla Bla Bla  
Bla Bla Bla Bla Bla Bla Bla Bla Bla Bla ...

# *Acknowledgements*

Bla  
Bla Bla Bla Bla Bla Bla Bla Bla Bla Bla ...

# Contents

<b>Declaration of Authorship</b>	<b>i</b>
<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>Contents</b>	<b>v</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>x</b>
<b>Abbreviations</b>	<b>xi</b>
<b>1 Problem statement</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 Task . . . . .	1
1.3 Text extraction and algorithms . . . . .	2
1.3.1 Text extraction . . . . .	2
1.3.2 Boilerpipe . . . . .	2
1.3.3 Justext . . . . .	2
1.3.4 Classification . . . . .	3
1.4 Conclusion . . . . .	3
<b>2 Approach</b>	<b>4</b>
2.1 Planning . . . . .	4
2.2 Requirement Engineering . . . . .	5
2.3 Knowledge . . . . .	5
2.4 Programming . . . . .	5
<b>3 Results</b>	<b>6</b>
3.1 Main Section 1 . . . . .	6
<b>4 Lesson learned</b>	<b>7</b>
4.1 Main Section 1 . . . . .	7

<b>5 Further work</b>	<b>8</b>
5.1 Main Section 1 . . . . .	8
<b>A Planning</b>	<b>9</b>
A.1 Version . . . . .	9
A.2 Planning concept . . . . .	9
A.3 Milestones overview . . . . .	10
A.4 Delivery objects . . . . .	11
A.5 Milestone one - m1 . . . . .	12
A.5.1 Stories m1 . . . . .	12
A.6 Milestone two - m2 . . . . .	14
A.6.1 Stories m2 . . . . .	14
A.7 Milestone three - m3 . . . . .	16
A.7.1 Stories m3 . . . . .	16
A.8 Milestone four - m4 . . . . .	17
A.8.1 Stories m4 . . . . .	17
A.9 Milestone five - m5 . . . . .	17
A.9.1 Stories m5 . . . . .	18
<b>B Risk Analysis</b>	<b>19</b>
B.1 Version . . . . .	19
B.2 Introduction . . . . .	19
B.2.1 Purpose . . . . .	19
B.3 Risk evaluation . . . . .	19
B.3.1 Unclear requirements . . . . .	19
B.3.2 New technologies . . . . .	20
B.3.3 Integration Boilerpipe . . . . .	20
B.3.4 Integration Justext . . . . .	20
B.3.5 Implementation RSS algorithm . . . . .	20
B.4 Assessment of risks . . . . .	20
B.5 Consequences . . . . .	21
B.5.1 Unclear requirements . . . . .	21
B.5.2 New technologies . . . . .	21
B.5.3 Integration Boilerpipe . . . . .	21
B.5.4 Integration Justext . . . . .	21
B.5.5 Implementation RSS algorithm . . . . .	22
<b>C Software Requirement Specification</b>	<b>23</b>
C.1 Version . . . . .	23
C.2 Introduction . . . . .	23
C.2.1 Purpose . . . . .	23
C.2.2 Scope . . . . .	24
C.3 General description . . . . .	24
C.3.1 Operating Environment . . . . .	24
C.3.1.1 Local environment . . . . .	24
C.3.1.2 Continuous Integration Environment . . . . .	24

C.3.2	Design and Implementation Constraints . . . . .	24
C.3.2.1	User interface . . . . .	24
C.4	System Features . . . . .	25
C.4.1	Basic functionality . . . . .	26
C.4.2	Overview . . . . .	27
C.4.2.1	Read configuration . . . . .	27
C.4.2.2	Create test . . . . .	28
C.4.2.3	Integration Justext algorithm . . . . .	28
C.4.2.4	Integration Boilerpipe algorithm . . . . .	28
C.4.2.5	Evaluation and Implementation RSS feed algorithm . . . . .	29
C.4.2.6	Evaluation of classification text . . . . .	29
C.4.2.7	Evaluation of classification blocks . . . . .	30
C.4.3	Analyze data . . . . .	30
C.4.4	Evaluation of classification . . . . .	30
C.4.5	Analytical values . . . . .	31
C.5	External Interface Requirements . . . . .	32
C.5.1	Boilerpipe . . . . .	32
C.5.2	justext . . . . .	32
<b>D</b>	<b>Software Architecture</b>	<b>33</b>
D.1	Version . . . . .	33
D.2	Introduction . . . . .	33
D.3	Logical view . . . . .	34
D.3.1	Approach . . . . .	34
D.4	Data model . . . . .	35
D.4.1	TestCase . . . . .	35
D.4.2	HtmlFile . . . . .	35
D.4.3	ContentFile . . . . .	35
D.4.4	ExctractedContent . . . . .	36
D.4.5	Result . . . . .	36
D.5	Business Logic . . . . .	37
D.5.1	Description of single classes . . . . .	39
D.6	Development view . . . . .	41
D.6.1	git . . . . .	41
D.6.2	Gradle . . . . .	41
D.6.3	Travic CI . . . . .	42
D.6.4	Justext . . . . .	42
D.6.5	Boilerpipe . . . . .	42
D.7	Process view . . . . .	42
D.8	Physical view . . . . .	42
<b>E</b>	<b>MS1 meeting report</b>	<b>43</b>
E.1	Introduction . . . . .	43
E.2	Version . . . . .	43
E.3	Attendees . . . . .	43
E.4	Delivery objects . . . . .	43
E.5	Decisions . . . . .	44

---

E.6	Rework . . . . .	44
<b>F</b>	<b>MS2 meeting report</b>	<b>45</b>
F.1	Introduction . . . . .	45
F.2	Version . . . . .	45
F.3	Attendees . . . . .	45
F.4	Delivery objects . . . . .	45
F.5	Decisions . . . . .	46
F.6	Rework . . . . .	46



# List of Figures

# List of Tables

# Abbreviations

**HTML**   **H**yper **T**ext **M**arkup **L**anguage

*For/Dedicated to/To my...*

# Chapter 1

## Problem statement

This chapter describes the problem statement as well as the topical environment where the project takes place.

### 1.1 Introduction

This project is done in behalf of the company Layzapp. Layzapp is specialized on second-screen solutions and they are currently working on a mobile application which brings relevant content to a second screen during a TV show. To do so the Internet is crawled for relevant information about a certain topic. The outcome of this search are a certain amount of web pages. To use the content of these web pages irrelevant data such as navigation elements, advertisement and login pages need to be removed. Removing the HTML content is not a very hard task but after doing so there is still a lot of content left such as description of further articles or advertisement which are not relevant for the user. Removing this part is much more complicated.

### 1.2 Task

As described in the introduction, irrelevant content such as advertisement, navigation elements and further articles which are also called boilerplate, need to be removed from a web page. There are already several algorithm which fulfill this task. To compare the performance of the known algorithm and contrast them to possible new algorithms a test environment is needed. The test environment needs to classify the quality of a text extraction performed by the different algorithms. To do so each algorithm is feed with a certain amount if HTML content and the outcome is then inspected for its quality.

## 1.3 Text extraction and algorithms

This chapter is a short description of the text extraction subject and the known algorithms.

### 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 and 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 which is a very time and resource consuming task. In this project the focus is on algorithms which work with boilerplate removal. The basic idea of boilerplate removal was first introduced with the BTE (Body Text Extraction) algorithm. The assumptions are the relevant part of the HTML content is usually a contiguous stretch, the density of HTML tags is lower in it than in boilerplate content. Hence, by breaking up the HTML page in single sections and counting the HTML tags, there will be an area where the number of HTML tags will not increase. It is quite simple to define an objective function. 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 are improving the performance of BTE and which are used in this project to work with in a first approach.

### 1.3.2 Boilerpipe

The Boilerpipe algorithm is based on the concept described in Kopfschüttels paper "Boilerplate Detection using Shallow Text Features". It uses a variate of HTML tags to divide the HTML document into blocks. Each block is classified with shallow text features and depending the classification of the previous and the next blocks. Some examples for shallow text features are average word length, average sentence length or the absolute number of words. The features are described more closely in the paper.

### 1.3.3 Justext

Justext uses similar features as Boilerplate but is inspecting 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 and based on this information, a better classification can be achieved.

### 1.3.4 Classification

The main task of the application is to classify the performance of the algorithms somehow. 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 performance can be described in a confusion matrix.

	<b>Classified as content</b>	<b>Classified as boilerplate</b>
<b>Actual content</b>	True positive (TP)	False negative(FN)
<b>Actual boilerplate</b>	False positive (FP)	True negative (TN)

- True positive is the amount of text which is relevant content and classified as content by the algorithm
- False positive is the amount of text which is relevant content but classified as boilerplate by the algorithm
- True negative is the amount of text which is boilerplate and is classified as boilerplate by the algorithm
- False negative is the amount of text which is boilerplate but is classified as content by the algorithm

These values are the basis for calculation the recall (also known as sensitivity) which is the fraction of relevant text that is retrieved and precision which is the fraction of retrieved text that is relevant. These values their dependencies and some more values are described more close in the software requirement specification ([C.4.4](#)).

## 1.4 Conclusion

An application is needed which can be feed with HTML documents which then are processed by text extraction algorithms. The outcome is then compared with the relevant content and classification values are calculated. Based on these values the performance of the algorithms can be compared and possible strengths and weaknesses can be determined.

## Chapter 2

# Approach

This chapter describes the approach developing the application. Most of the content in this chapter is covered by the project documents as well. It describes the general idea how the application is developed.

### 2.1 Planning

The planning of this project was a very straight forward task. Since it is a small one man project and the Eckdaten were clearly defined there was not a lot of spielraum. However I am working part time for Layzapp and this project was not the only task during the semester and the project needed to be coordinated somehow with other tasks. My solution was to work about one to two weeks for on this project and switch to other task for the next one to two weeks. Switching between multiple projects within three days is 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 of the project. Several milestones and its delivery objects were defined. The first milestone was then planned more closely with stories. After finishing the first milestone, the second one was planned more closely. So the vorgehen is a combination of the RUP vorgehensmodel and Scrum. Since it was a one man team no weekly scrum meetings were done but the milestones were defined as sprint meetings where the delivery objects were reviewed with the supervisors and tasks for the next milestone were defined or adjusted. During the project, the tool toggl was used as a time tracker and task manager. The whole planning as well as the time tracking auswertung can be found in the abbreviations ??.



## **2.2 Requirement Engineering**

## **2.3 Knowledge**

I had no knowledge about text extraction at all. This was a big risk at the beginning at the project and needed to be handled somehow. I researched the topic in the internet and got help from my advisors as well and

## **2.4 Programming**

## Chapter 3

# Results

### 3.1 Main Section 1

## Chapter 4

# Lesson learned

### 4.1 Main Section 1

## Chapter 5

# Further work

### 5.1 Main Section 1

# 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

### A.2 Planning concept

So as to plan the project, a combination of the two well known planning frameworks scrum and RUP are 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.

A more detailed planning is done for the incoming milestone / sprint. The predefined working packages are split into smaller packages. For the first draft, only the first milestone is split into smaller packages. The later milestones are going to be defined in more detail as soon as all needed information is available.

The effort needed for the documentation is not listed separately. All the tasks already contain additional time for updating the documentation.

The milestones dates are not finally defined, which means that the meeting dates can vary by up to some days. sadfjsalödkfjsadlk

### A.3 Milestones overview

<b>Name</b>	<b>Shortcut</b>	<b>Weeks</b>	<b>Estimated hours</b>	<b>Hours total</b>	<b>Closing date</b>
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	<ul style="list-style-type: none"><li>• System specification</li><li>• Sketch software architecture</li><li>• Short presentation CI environment</li><li>• Draft risk evaluation</li></ul>
Milestone two	22.10.2014	<ul style="list-style-type: none"><li>• Elaborated software architecture</li><li>• Tested code of test framework (tbd: which components)</li><li>• Interface definition for justext/boilerplate components</li><li>• HTML test data</li></ul>
Milestone three	05.11.2014	<ul style="list-style-type: none"><li>• Working test environment with both justext and boilerplate components integrated</li></ul>
Milestone four	19.11.2014	<ul style="list-style-type: none"><li>• Evaluation environment for output data of test framework</li><li>• First approach to new algorithm</li></ul>
Milestone five	08.12.2014	<ul style="list-style-type: none"><li>• Implementation of new algorithm</li><li>• Final documentation</li><li>• Final presentation</li></ul>

## A.5 Milestone one - m1

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

Story	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

<b>Title</b>	Planning
<b>Id</b>	s0
<b>Estimated time</b>	4h
<b>Description</b>	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.

<b>Title</b>	Research HTML / Algorithms
<b>Id</b>	s1
<b>Estimated time</b>	8h
<b>Description</b>	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.



<b>Title</b>	System specification
<b>Id</b>	s2
<b>Estimated time</b>	12h
<b>Description</b>	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.

<b>Title</b>	Draft software architecture
<b>Id</b>	s3
<b>Estimated time</b>	8h
<b>Description</b>	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.

<b>Title</b>	Risk evaluation
<b>Id</b>	s4
<b>Estimated time</b>	8h
<b>Description</b>	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.

<b>Title</b>	Configuration CI environment
<b>Id</b>	s5
<b>Estimated time</b>	4h
<b>Description</b>	<p>To deliver high quality software a continuous integration environment is required. Following tools should be evaluated and configured for further use:</p> <ul style="list-style-type: none"> <li>• Version control (git)</li> <li>• Project build automation tool (gradle)</li> <li>• continuous integration service (Travis CI)</li> </ul>

## A.6 Milestone two - m2

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

Story	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

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

<b>Title</b>	Implementation File Reader
<b>Id</b>	s7
<b>Estimated time</b>	6h
<b>Description</b>	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.

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

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

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

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

<b>Title</b>	Collection of test data
<b>Id</b>	s12
<b>Estimated time</b>	8h
<b>Description</b>	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

Story	Shortcut	Estimated time
Implementation test framework	s9	20h
Final integration of justext / boilerplate	s10	10h
Total		30h

### A.7.1 Stories m3

<b>Title</b>	Implementation test framework
<b>Id</b>	s9
<b>Estimated time</b>	20h
<b>Description</b>	Final implementation of the test framework. This story will be divided into smaller stories as soon as the software architecture and the system specification is reviewed.

<b>Title</b>	Prototype Integration of justext/boilerpipe
<b>Id</b>	s10
<b>Estimated time</b>	4h
<b>Description</b>	Complete integration of the justext and boilerplate algorithms into the test framework. This story will be divided into smaller stories as soon as the software architecture and the system specification is reviewed.

## A.8 Milestone four - m4

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

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

### A.8.1 Stories m4

<b>Title</b>	Evaluation environment of results
<b>Id</b>	s11
<b>Estimated time</b>	20h
<b>Description</b>	The test framework will produce a lot of output data, which has to be reviewed using an evaluation environment. This should process this data and present the results in a descriptive way. This story will be divided into smaller stories as soon as the software architecture and the system specification is reviewed.

<b>Title</b>	Research on new algorithm
<b>Id</b>	s12
<b>Estimated time</b>	20h
<b>Description</b>	A first research on the new algorithm should be performed. After this research it should be possible to decide if this solution is possible and if an implementation with the remaining time resources is realistic. This story will be divided into smaller stories as soon as the software architecture and the system specification is reviewed.

## A.9 Milestone five - m5

- Closing date date: 8.12.2014

- Available time: ca. 38h

Story	Shortcut	Estimated time
Implementation of new algorithm	s13	19h
Complete documentation	s14	15h
Prepare final presentation	s15	4h
Total		38h

### A.9.1 Stories m5

<b>Title</b>	Implementation of new algorithm
<b>Id</b>	s13
<b>Estimated time</b>	19h
<b>Description</b>	Implementation of the new algorithm and analysis of the test results with the existing evaluation environment.

<b>Title</b>	Complete documentation
<b>Id</b>	s14
<b>Estimated time</b>	15h
<b>Description</b>	Complete and review all chapters of the documentation.

<b>Title</b>	Prepare final presentation
<b>Id</b>	s15
<b>Estimated time</b>	4h
<b>Description</b>	Prepare the final presentation and the final printed / digital version of the thesis.

# 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 milestone. 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

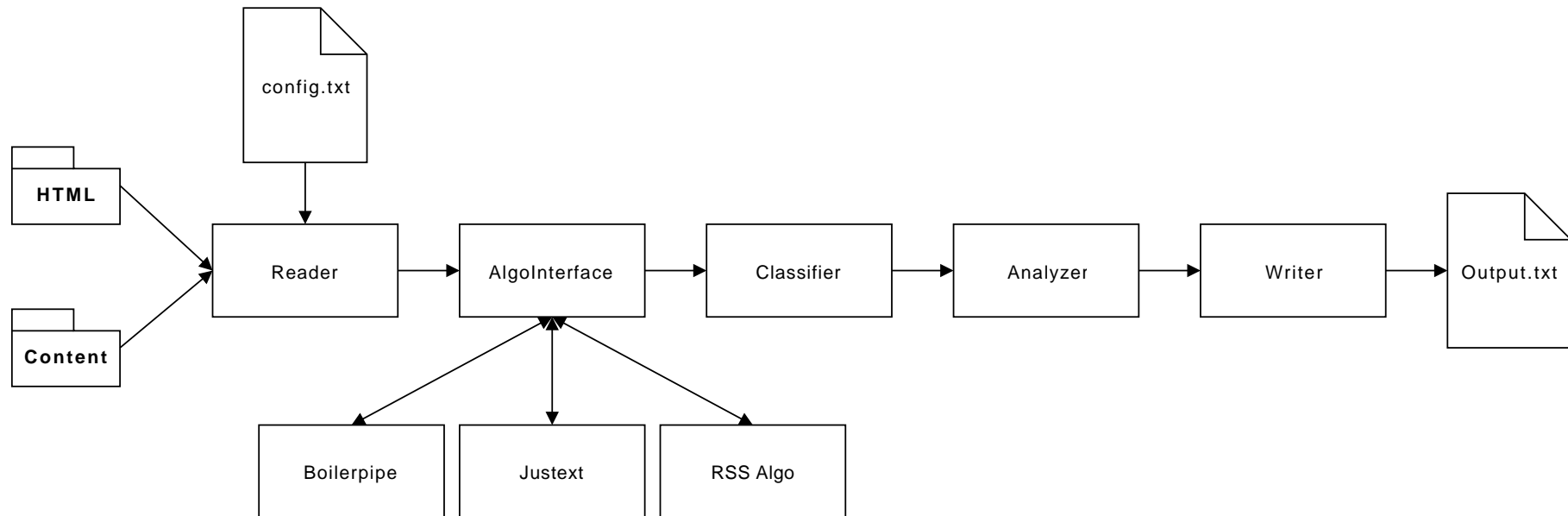
no graphical user interface. The application is built, deployed and started by gradle. While the application is running, no interaction is needed.

## **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	<a href="#">C.4.2.1</a>	needed
f2	Create test	<a href="#">C.4.2.2</a>	needed
f3	Integration Justext algorithm	<a href="#">C.4.2.3</a>	needed
f4	Integration Boilerpipe algorithm	<a href="#">C.4.2.4</a>	needed
f5	Evaluation and Implementation RSS feed algorithm	<a href="#">C.4.2.5</a>	nice to have
f6	Evaluation of classification text	<a href="#">C.4.2.6</a>	needed
f7	Evaluation of classification blocks	<a href="#">C.4.2.7</a>	nice to have
f8	Analyze data	<a href="#">C.4.3</a>	needed

### C.4.2.1 Read configuration

<b>Name</b>	Read configuration
<b>Feature id</b>	f1
<b>Description</b>	<p>The text extraction framework is configurable with an external text file. The configuration file will contain following items:</p> <ul style="list-style-type: none"> <li>• Path to folder with HTML files</li> <li>• Path to folder with text files</li> <li>• Path to folder with output files</li> <li>• Configuration for algorithms</li> <li>• etc.</li> </ul> <p>The configuration file location is defined as a relative path to the source directory and structured in a key value list:</p> <hr/> <pre>key:value; key:value; key:value;</pre> <hr/>
<b>Relevance</b>	needed
<b>Related stories</b>	tbd

**C.4.2.2 Create test**

<b>Name</b>	Create test
<b>Feature id</b>	f2
<b>Description</b>	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.
<b>Relevance</b>	needed
<b>Related stories</b>	tbd

**C.4.2.3 Integration Justext algorithm**

<b>Name</b>	Integration Justext algorithm
<b>Feature id</b>	f3
<b>Description</b>	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.
<b>Relevance</b>	needed
<b>Related stories</b>	tbd

**C.4.2.4 Integration Boilerpipe algorithm**

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



**C.4.2.5 Evaluation and Implementation RSS feed algorithm**

<b>Name</b>	Evaluation and implementation RSS feed algorithm
<b>Feature id</b>	f5
<b>Description</b>	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.
<b>Relevance</b>	nice to have
<b>Related stories</b>	tbd

**C.4.2.6 Evaluation of classification text**

<b>Name</b>	Evaluation of classification
<b>Feature id</b>	f6
<b>Description</b>	<p>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.</p> <ul style="list-style-type: none"><li>• Check each classified block from the algorithms if its content can be found in the actual content</li><li>• Categorize text as boilerplate or content</li><li>• Insert results in an output text file</li></ul> <p>Both the evaluation and classification are defined in more detail in section <a href="#">C.4.4</a>.</p>
<b>Relevance</b>	needed
<b>Related stories</b>	tbd

#### C.4.2.7 Evaluation of classification blocks

<b>Name</b>	Evaluation of classification blocks
<b>Feature id</b>	f6
<b>Description</b>	<p>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 Boilerpipe 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.</p> <ul style="list-style-type: none"><li>• Check each classified block from the algorithms if its content can be found in the content file</li><li>• Categorize all blocks as boilerplate or content</li><li>• Insert the results in an output text file (structure output file: tbd)</li></ul> <p>Both the evaluation and classification are defined in more detail in section <a href="#">C.4.4</a>.</p>
<b>Relevance</b>	nice to have
<b>Related stories</b>	tbd

#### C.4.3 Analyze data

<b>Name</b>	Analyze data
<b>Feature id</b>	f7
<b>Description</b>	<p>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 <a href="#">C.4.5</a>.</p>
<b>Relevance</b>	needed
<b>Related stories</b>	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.

	<b>Classified as content</b>	<b>Classified as boilerplate</b>
<b>Actual content</b>	True positive (TP)	False negative(FN)
<b>Actual boilerplate</b>	False positive (FP)	True negative (TN)

	<b>Classified as content</b>	<b>Classified as boilerplate</b>
<b>Actual content</b>	Word classified as content by algorithm and is content	Word classified as boilerplate by algorithm but is content
<b>Actual boilerplate</b>	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.

	<b>Classified as content</b>	<b>Classified as boilerplate</b>
<b>Actual content</b>	Block is classified as content by algorithm and is content	Block is classified as boilerplate by algorithm but is content
<b>Actual boilerplate</b>	Block is classified as content by algorithm but is boilerplate	Block is classified as boilerplate by algorithm and is boilerplate

In conclusion, TP + FN is the correct outcome of the algorithm i.e. content classified as content and boilerplate as boilerplate. On the other hand, TN + 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 objects 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 a relevant document is retrieved in a search which in our case is

$$Recall = \frac{TP}{TP + FN} \quad (C.1)$$

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

##### **Precision / True negative rate / TNR**

Precision is the probability that a retrieved document is relevant which in our case is

$$Presicion = \frac{TP}{TP + FP} \quad (C.2)$$

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

$$Fmeasure = 2 * \frac{presicion * recall}{presicion + recall} \quad (C.3)$$

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

$$Fallout = \frac{FP}{FP + TN} \quad (C.4)$$

## **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

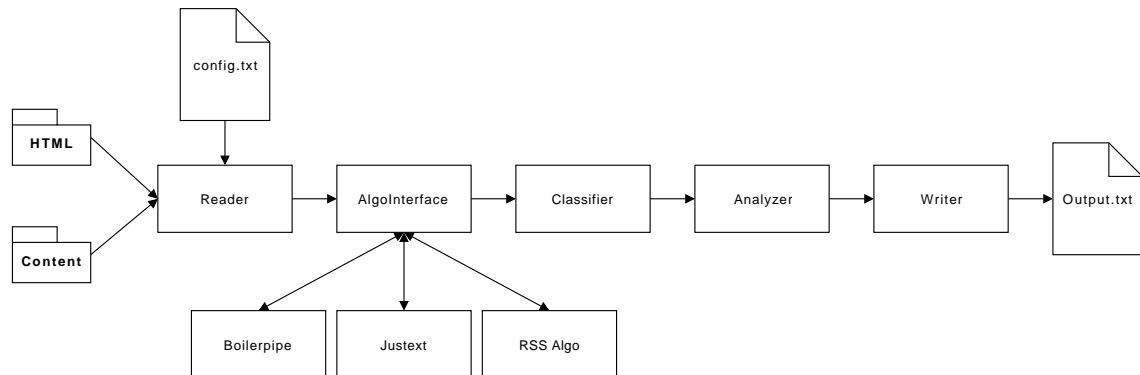
### D.2 Introduction

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 Logical view

### D.3.1 Approach

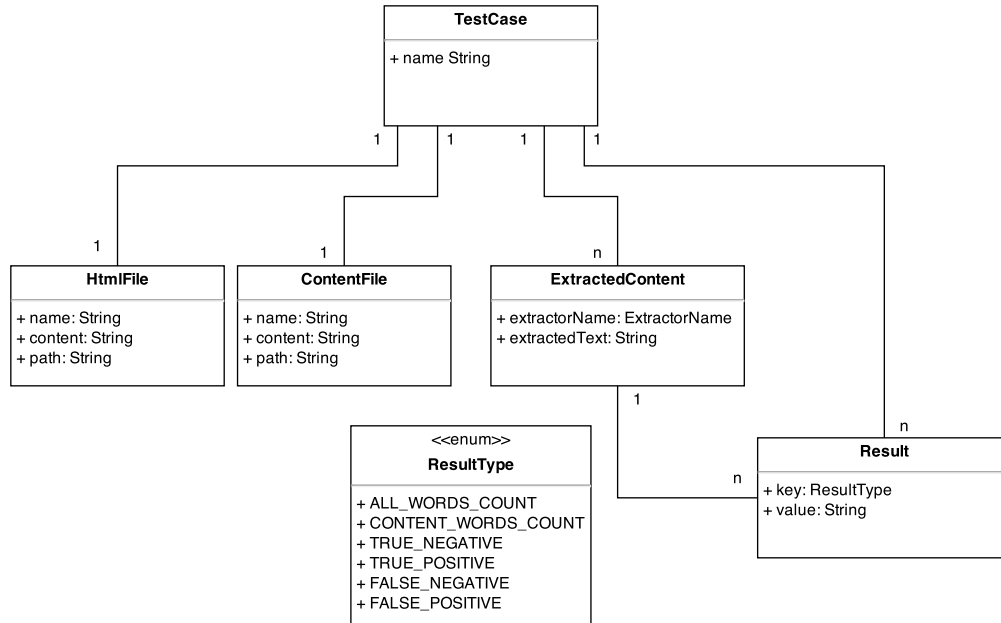
This section describes the approach for elaborating the software architecture. The following figure from the software requirement specification is the basis for elaborating the software architecture.



First all possible entities in the domain are identified and a data model is elaborated. The data model is described in [D.4](#). 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 [D.5](#).

## D.4 Data model

The following diagram shows the data model of the application.



### D.4.1 TestCase

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

### D.4.2 HtmlFile

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

### D.4.3 ContentFile

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

#### **D.4.4    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.

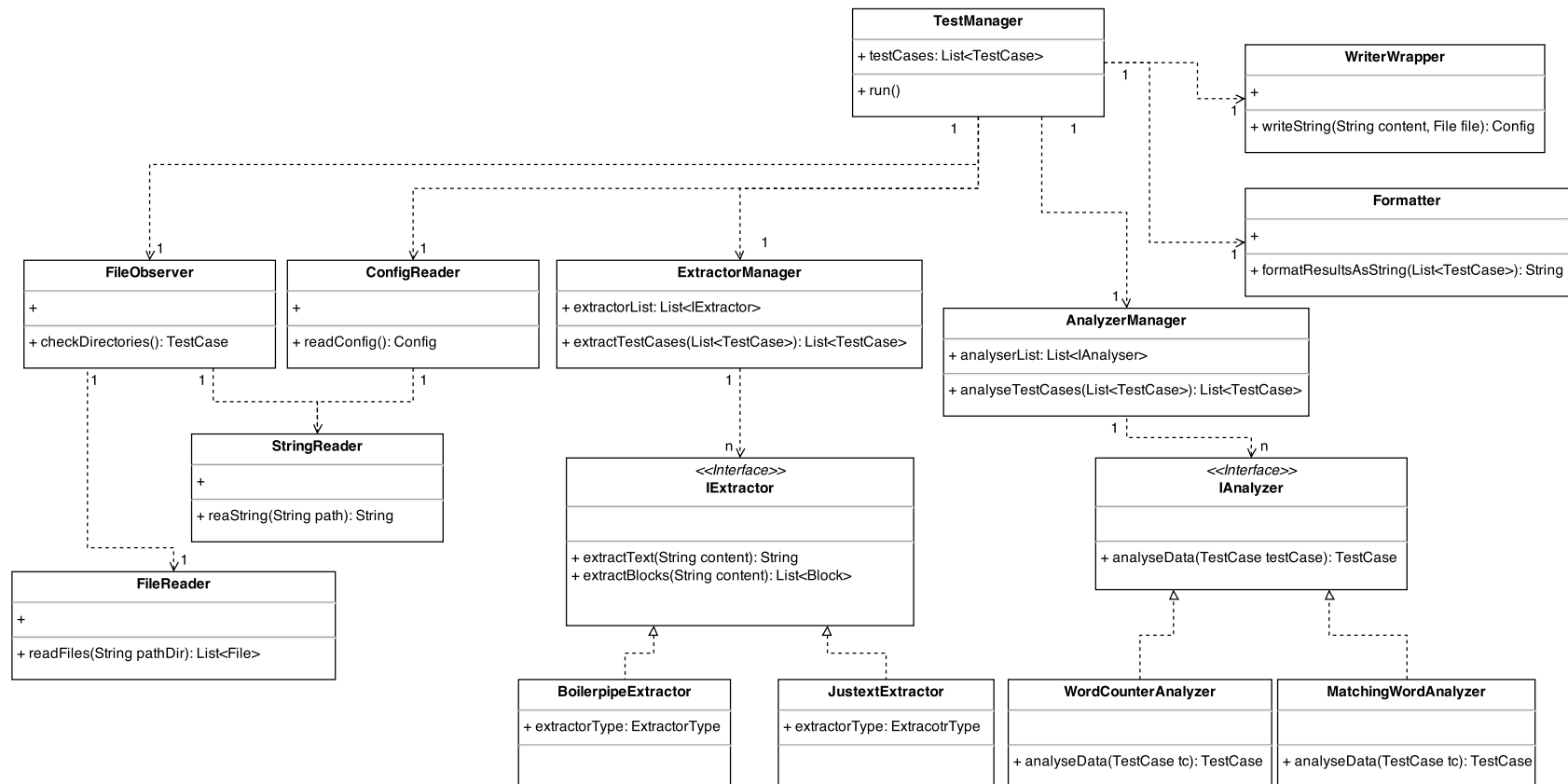
#### **D.4.5    Result**

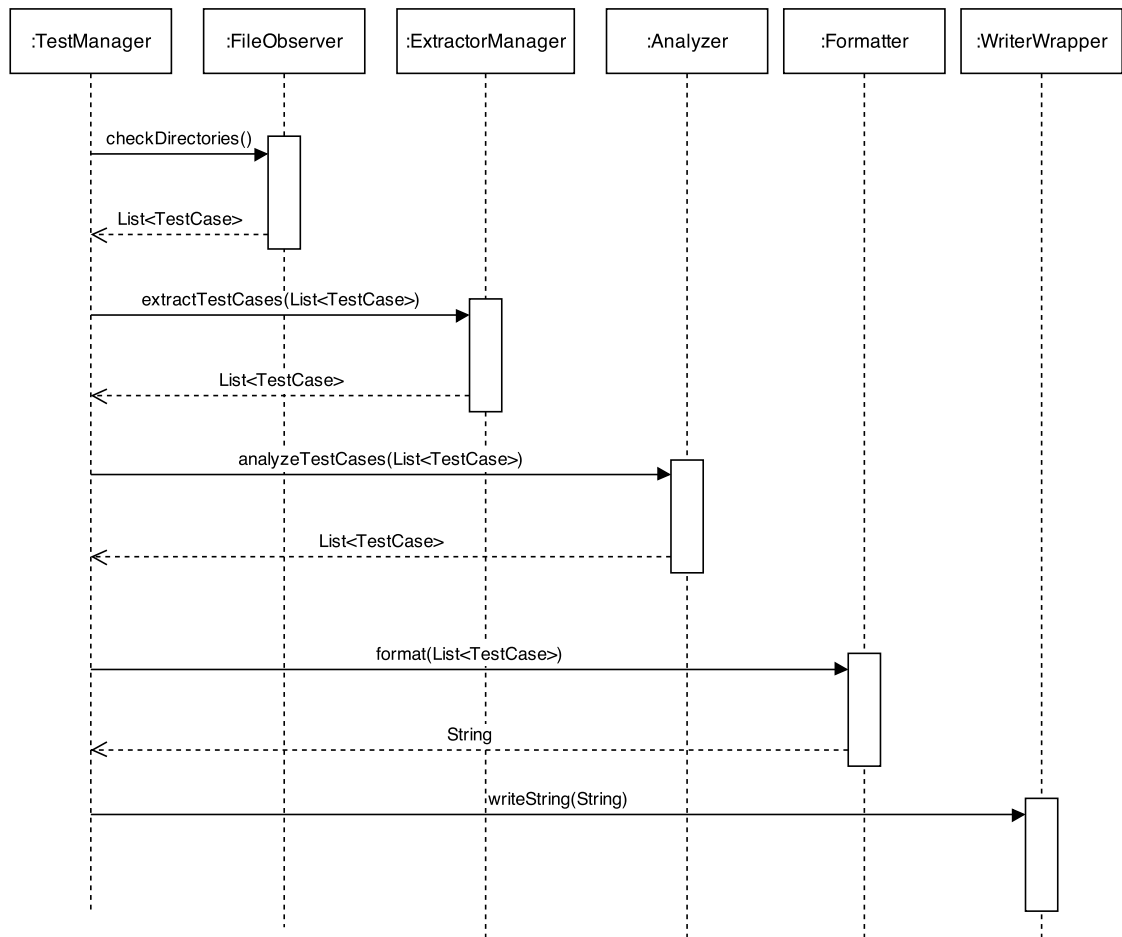
A TestCase or an ExctractedContent object can have Result objects. The results are key value pairs which represent analytical data. An example for a Result related to a TestCase would be the word count of the content file, which is generally valid. An example for a Result related to an ExctractedContent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



## D.5 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.





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.5.1 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.6 Development view

This chapter describes the used frameworks and tools which are used during the development process.

### D.6.1 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](https://github.com/heya87/pawi_doc)

### D.6.2 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.6.3 Travis CI

Travis CI is an open source build server. It is easy to use in combination with git. One only needs to add a config 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.6.4 Justext

An implementation of the Justext algorithm is available in Python. The resources 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.6.5 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.7 Process view

## D.8 Physical view

# Appendix E

## MS1 meeting report

### E.1 Introduction

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

### E.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

### E.3 Attendees

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

### E.4 Delivery objects

- System specification

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

## E.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.

## E.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 F

## MS2 meeting report

### F.1 Introduction

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

### F.2 Version

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

### F.3 Attendees

- Joel Rolli
- Michael Kaufmann

### F.4 Delivery objects

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

## F.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.

## F.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.