

# *Readabix*

## **BEng Individual Project Report**

Avgoustinos Kadis  
Supervised by Francesca Toni

June 15, 2010

## Abstract

Reading texts in a second language is challenging for many people. The plenty of unknown words and complicated syntax makes the life of the reader a lot harder. Readabix is a service that radically changes the reading experience from unpleasant and demotivating to enjoying and encouraging.

How does it do that? First, it tries to find the easiest articles on the newspapers of the day. Then it displays them to the user to select one to read. While the user reads the page through a highly interactive interface that allows marking of unknown words and instant translation, the system adapts to the user's level and tries to find him easier articles. When the user goes back to the articles list he will notice that new, easier articles appear. At the same time he can track his progress by seeing how many words he knows in the language and how many he has learned. User can then read newly recommended articles and if he enjoys reading one of them, he can recommend it to others at his level. Readabix has a smart way of matching user's level and recommending articles between the users.

And that's not all! Readabix was very carefully designed and optimised so all these smart calculations would be as quick as possible. Additionally, in order to allow the system to support hundreds or even thousands of users, we hosted it on Google's cloud. We used a distributed database for storage and optimised our algorithms to access it effectively.

Users liked it, as evaluation showed they found it very useful for improving their language skills. Hope you'll like it too.

<http://readabix.appspot.com>

### **Acknowledgements**

Firstly, above all, I want to thank my fiancée Karolina for her unconditional support and guidance through these months. She was giving me the courage and hope I needed, in moments that only she could do it.

Secondly, I want to thank my parents for dedicating their lives to me, my brother and my sister and giving me everything I needed to come so far. I hope I make you proud.

Thirdly, I want to thank Neophytos for believing in me and insisting on me studying abroad. I wouldn't be here without him.

I want to also apologise to my best friend Giorgos, for coming here alone. I grew up now.

I want to also thank all the friends and colleagues that helped with evaluating Readabix and giving me such motivating and great feedback!

Lastly, I want to thank my supervisor Francesca for accepting Readabix with such enthusiasm and prospect. I'll always remember her endless ideas, improvements and calls for action. Thanks for the positive atmosphere and pleasant meetings. They helped :)

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Road Map . . . . .	3
<b>2</b>	<b>Background</b>	<b>4</b>
2.1	Reading Comprehension . . . . .	4
2.2	Measuring Text Readability . . . . .	6
	Baseline . . . . .	7
	Vocabulary . . . . .	7
	Syntactic . . . . .	8
	Entity Coherence . . . . .	9
	Discourse Relations . . . . .	9
2.3	Content Extraction . . . . .	9
2.4	Collaborative Filtering . . . . .	10
<b>3</b>	<b>Retrieving Articles</b>	<b>11</b>
3.1	RSS Feeds . . . . .	11
3.2	Articles . . . . .	13
	Download HTML . . . . .	14
	Extract Content . . . . .	15
	Tokenisation . . . . .	17
	Word Filtering . . . . .	18
	Count Word Frequency . . . . .	18
	Create Reverse-Index . . . . .	19
	Evaluating Difficulty . . . . .	19
<b>4</b>	<b>Reading Articles</b>	<b>21</b>
4.1	Requirements . . . . .	21
4.2	Possible Solutions . . . . .	22
	Translating Words . . . . .	22
	Marking Words . . . . .	23
	Saving User Progress . . . . .	23
4.3	Our Solution . . . . .	24
	Interactivity . . . . .	25
	Translation . . . . .	29

Saving Marked Words . . . . .	29
Saving Paragraph or Text . . . . .	30
4.4 Performance . . . . .	30
<b>5 Recommending Articles</b>	<b>32</b>
5.1 Interface . . . . .	32
5.2 User Independent Measures . . . . .	33
Word Frequency . . . . .	34
Word Repetition . . . . .	35
5.3 User Dependent Measures . . . . .	36
Known Words . . . . .	37
Unknown Words . . . . .	37
Weighted Mode . . . . .	37
Easy & Challenging Mode . . . . .	38
5.4 Combined Measures . . . . .	38
5.5 Incremental Counting . . . . .	39
<b>6 Recommending Words</b>	<b>41</b>
6.1 User Interface . . . . .	41
6.2 Measures . . . . .	42
6.3 Recommendations . . . . .	43
6.4 User Progress . . . . .	44
<b>7 User Feedback</b>	<b>45</b>
7.1 Motivation . . . . .	45
7.2 Profile Matching . . . . .	46
7.3 Recommendation Algorithm . . . . .	47
7.4 Incremental Counting . . . . .	48
<b>8 Software Engineering</b>	<b>50</b>
8.1 System Overview . . . . .	50
Retrieval Module . . . . .	52
Processing Module . . . . .	53
Feedback Module . . . . .	53
8.2 De-Coupling Database Operations . . . . .	54
8.3 Running on the cloud . . . . .	56
Datastore . . . . .	56
Task Queues . . . . .	57
8.4 User Interface . . . . .	58
8.5 Code Base . . . . .	59
<b>9 Evaluation</b>	<b>61</b>
9.1 Objectives . . . . .	61
9.2 Evaluation Environment . . . . .	62
9.3 Tracking Users . . . . .	62
9.4 Survey . . . . .	64

9.5 Existing Tools . . . . .	69
<b>Bibliography</b>	<b>72</b>

# Chapter 1

## Introduction

Reading in a second language is a quite challenging and rewarding task. The reader needs to spend lot of time disassembling the meaning of the text, looking up words on the dictionary etc. The most challenging part is finding a good text to read. Children start reading small stories with simple words and plot. Even though children stories seem excellent source of easy texts to read, adults find them very boring. From the other side, adults don't have much time available to be looking up many words on the dictionary and spend lot of time on getting the meaning of the text. The reader wants something interesting to read without too much effort and at the same time to allow him to expand his knowledge of the language.

The question now is: where can the user find such texts? Novels are quite long and usually use harder and complicated tenses and words. Language learning books have texts at a more suitable level but they are not accessible to every person and the texts can be rather boring or just outdated. Newspapers though contain "fresh" texts, probably more interesting than language books. They cover many different subjects of interest (sports, politics, health etc) which increases the probability of finding something interesting. Language used is not necessarily the easiest to read but there is a lot of word repetition in their vocabulary. This makes it rewarding for the reader when he learns a word and encounters it later on.

Newspaper sounds like a great source. But how can the user find newspaper articles? Visiting the newspaper's website is one option but even better is to use aggregated news services (like Google News) which gather articles from various newspapers and present them in a single website. This gives quick access to the most recent news from all around the world. There is an issue though. The articles that are available on Google News are not guaranteed to be easy for you to read. If an article has too many unknown words, reading it can be very demotivating and tiring. Here comes the need for Readabix, the service that

finds you the most suitable news articles at the right level of difficulty (ex. not too many unknown words).

Readabix is a new web-service where the reader can quickly find articles that are easy for him to read (not many unknown words). While reading an article the user can look-up the word translation and add words to his vocabulary lists. Vocabulary lists contain words he knows and words he wants to learn (unknown words he translated). Additionally, the user is able to rate the difficulty of the text he read and the system will recommend it to users of lower, same or higher level (of reading skills in that language).

Readabix is offering news articles in English, Spanish, French, Polish and Greek. The system design allows it to expand to other languages easily.

To develop this service we use a combination of techniques from various fields: *Language Learning* (non-technical), *Natural Language Processing*, *Information Retrieval*, *Recommender Systems*, *Social Computing* and *Software Engineering*.

The aims of this project are:

- Collect news articles from various sources and provide them under a single interface.
- Find easy articles for the user to read that will allow him to firm his language skills.
- Find a bit more challenging articles for the user that wants to expand his language skills.
- Allow the user to give feedback on which words he doesn't know.
- Allow the user to give feedback on how easily he could read an article or not.
- Provide news recommendations using information from the users network.
- Provide a simple web interface for browsing the news articles.

Additional aims would be to:

- Recommend new words for the user to learn.
- Inform user about his progress with which and how many words he learned by reading the texts.
- Provide instant translation of words through online dictionaries.



## 1.1 Road Map

We start with the *Background* (chapter 2) in order to understand the parameters that affect reading comprehension for second language learners and to show us what has been done on the field of measuring text readability.

With that and a couple more algorithms for content extraction and network recommendations we go off to understand how Readabix retrieves the articles in the *Retrieving Articles* (chapter 3).

We then present the interactive interface we designed for *Reading Articles* (chapter 4) on Readabix. In this chapter we present the kind of feedback we receive from the user and we explain how we use this feedback for *Recommending Articles* (chapter 5).

In chapter 6, we present our method for *Recommending Words* and the reasoning behind it. In chapter 7, we see how we use *User Feedback* to recommend articles to other users.

Chapter 8 explains the *Software Engineering* challenges we faced in order to run Readabix on the cloud. When we run it on the cloud, we asked users to do *Evaluation* of the system. In chapter 9 we present the findings and we make conclusions.

## Chapter 2

# Background

Readabix wants to help the user that wants to use/improve his reading skills in a second language (L2). To do that we need to find out which aspects need to be taken into consideration when processing the news articles. That means, to find out what makes a text easy (or hard) to read. Additionally, to improve reading skills we need to see what methods are used in traditional language learning and what are the most recent practises (see Section 2.1).

After that, we investigate some existing methods for measuring text readability (Section 2.2) in order to find the most suitable technique that will be working among different languages and will be easy to implement (due to limited time available).

In Section 2.3, we see some Content Extraction methods, which are useful for extracting the news article body from HTML (format in which news articles appear online), avoiding noisy text such as advertisements, copyright statements etc.

Section 2.4 talks about efficiency issues of the Collaborative Filtering algorithm. This algorithm will be used for recommending news articles based on information gathered through the users network.

### 2.1 Reading Comprehension

Learning a language is a long and complex undertaking with many variables involved in the acquisition process [Bro00]. From this broad field of language learning, we are particularly interested in reading comprehension (the level of understanding a writing). We are interested on the factors that affect reading comprehension in a second language (L2) and how can the reading skills be improved.

The two main factors that have been shown that affect L2 readers are:

1. **Percentage of known words in the text:** According to [Wal03], in order to read comfortably, skilled readers need to know 95% or more of the words in a text and be able to recognise them easily. Additionally, "those who know 90 percent of the words in a text will understand its meaning and, because they understand, they will also begin to learn the other 10 percent of the words. Those who do not know 90 percent of the words, and therefore do not comprehend the passage, will now be even further behind on both fronts: They missed the opportunity to learn the content of the text and to learn more words." [EDH03].
2. **Background knowledge on the topic:** When a reader has background knowledge on the subject of a text, it has been observed that he can understand and recall better the text. As cited in [Wal03] "it has been shown that even across passages on the same general theme, which had identical structure and syntax and very similar vocabulary, the more familiar version is better recalled". Additionally, background knowledge enables the readers to make sense of word combinations and choose among multiple possible word meanings [EDH03].

A smaller factor that can affect the comprehension is the knowledge about the type of the text. When a reader knows what he is reading (ex. a newspaper article rather than a page from a novel), he unconsciously knows what sort of information to expect, and has an influence on comprehension [Wal03].

The factor of how much the reader is interested in the topic of the text (topic interest) does not seem to be important in L2 readers [Lee09]. This is completely different in the case of first language (L1) readers, where topic interest influences significantly reading comprehension. It may be reasonable, since L2 readers might be focusing on low-text processing rather than the information given by the text. This might not be the case with fluent L2 readers, which are reading because of interest in the subject.

We have seen that vocabulary and background knowledge are significant to reading comprehension. Now, how can the reader improve his reading skills? **Extensive reading** - high exposure to texts at or just below a comfortable level of comprehension. It is vital for development of automaticity<sup>1</sup> in low-level processing, providing as it does repeated exposure to frequent vocabulary items [Wal03]. By extensive reading, the reader is being exposed to unknown words and their frequent appearance makes it easier for the reader to recognise and infer their meaning.

This method of acquiring vocabulary is classified as *shallow* strategy. *Deep* strategy takes more time and focuses on vocabulary learning. It includes vocabulary lists, extensive use of dictionaries and lot of revising to ensure the word has

---

<sup>1</sup>unconscious and quick recognition and understanding of a word

been memorised [Wak03]. In other words, by improving vocabulary knowledge you improve reading and reading improves vocabulary knowledge [GS97].

Each reader has some background knowledge on various fields, especially adult readers. The challenge with L2 reading is that in order to fully understand a text, the reader needs a basic knowledge of the culture and way of thinking of the people in the new language [Bro00]. Reading about the new culture in the reader's mother language may help developing a better level of understanding. As the reader's comprehension skills improve he will probably be able to learn cultural facts through extensive reading as well. The choice of reading materials is important and this leads us to investigation of newspapers as a possible resource for extensive reading.

"Newspaper is the most widely and consistently read piece of literature published. Much can be taught from the newspaper because it contains much" [Che71]. Newspaper as reading material covers many topics of interest and becomes source of different opinions on the same subject. They are rich in cultural and historical information and older printings are accessible through archives in libraries. Various studies have been made on how effective can newspaper be in improving reading comprehension. A particular case described in [GS97], presents a person learning Portuguese mainly by reading newspaper and translating words from a bilingual dictionary on a daily basis. The tests he was taking over the time showed that his reading comprehension skills (measured by translation tests) went from 30% accuracy to 90% in just 3 months! Bill, the person in the study, stated that the topics and vocabulary of the texts in the newspaper were keeping him motivated to continue reading. The use of English newspaper was also helpful to obtain background knowledge on some news stories.

## 2.2 Measuring Text Readability

Readability is what makes some texts easier to read than others [DuB04]. Many readability measures have been developed over the years and have been widely used in journalism, research, health care, law, insurance and industry. The main use of these formulae was to give feedback to editors that wanted their texts to be easily read by an average person. This is quite essential for any kind of manuals or written instructions that are connected with medicine or other life crucial texts. Other popular uses of readability measures are for categorising books for children, into different levels of difficulty, and helping publishers target different ages.

Readability measures can be put into the 5 categories (shown in Figure 2.1) and explained below. We start from the most simple measurements to the more advanced techniques.

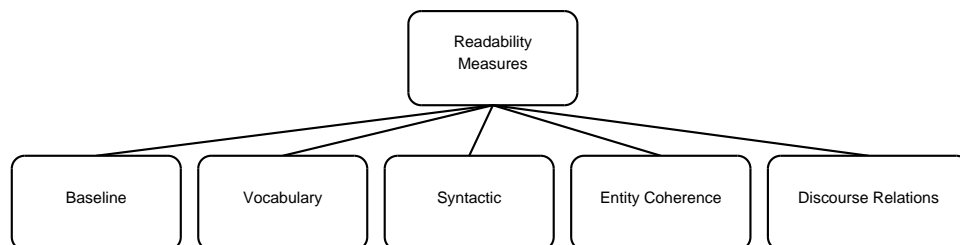


Figure 2.1: Methods for measuring readability

## Baseline

These were the first readability measures developed and were based on surface linguistic features<sup>2</sup> such as number of **words in the sentences** and the **number of letters or syllables per word**. Some measures were based on the observation that smaller words tend to be in the list of most common words of the language. The most popular of these are

- *Flesch Reading Ease* [Kin75]: Measures the number of words over the number of sentences and the number of syllables over the number of words. The formula is  $206.835 - 1.015 \times \frac{\text{totalwords}}{\text{totalsentences}} - 84.6 \times \frac{\text{totalsyllables}}{\text{totalwords}}$  and gives a measure from 0 to 100.
- *Flesch-Kincaid Grade Level* [SC01]: The grade level formula gives a number from 1 to 12 which indicates the grade (school age) that the student needs to be in order to find the text easy. The formula is  $0.39 \times \frac{\text{totalwords}}{\text{totalsentences}} + 11.80 \times \frac{\text{totalsyllables}}{\text{totalwords}} - 15.59$ .
- *Automated Readability Index - ARI*: This measure also gives an approximation of the US grade level (over which grade can this text be read easily) but uses different measures. The formula is  $4.71 \times \frac{\text{characters}}{\text{word}} + 0.5 \frac{\text{words}}{\text{sentence}} - 21.43$  [DuB04].

More recent research has shown that baseline measures don't approximate well the readability of a text and other methods that we will discuss below provide more accurate measures. Additionally, these measures are limited to the English language only. There are similar formulae for other popular languages, but they can not be generalised to any language.

## Vocabulary

Vocabulary based measures use the **word frequency** as their basic measure of text difficulty. They create a language model based on a background corpus<sup>3</sup>

---

<sup>2</sup>not getting deep into the structure or meaning of the text

<sup>3</sup>large amount of text in a specific language

and they predict the probability of knowing the meaning of the word, based on how often the word occurs in the corpus. See [PN08] for a detailed mathematical model.

The vocabulary measures are very powerful because they provide us with the ability to modify the language model and measure readability for different categories of people. For example, we can get some 6th grade books and create a language model over them and use it for predicting the readability for a 6th grade student!

## Syntactic

Syntactic measures take into consideration the **number of noun phrases**, **verb phrases**, **subordinate clauses per sentence** and **parse tree height**. These measures involve Natural Language Processing techniques such as *part of speech tagging* and *parsing*. Tagging is the task of labelling (or tagging) each word in a sentence with its appropriate part of speech. Parsing is the process of reconstructing the derivation(s) or phrase structure tree(s) that give rise to particular sequence of words [MS02].

For example, the sentence "the dog barked" (depicted in Figure 2.2) by part of speech tagging we would be able to recognise that "the" is an article (AT), "dog" is a singular noun (NN) and "barked" is a verb in the past tense (VBD). By parsing we would be able to recognise the noun phrase (NP) "the dog" and the verb phrase (VP) "barked" which form the whole sentence (S).

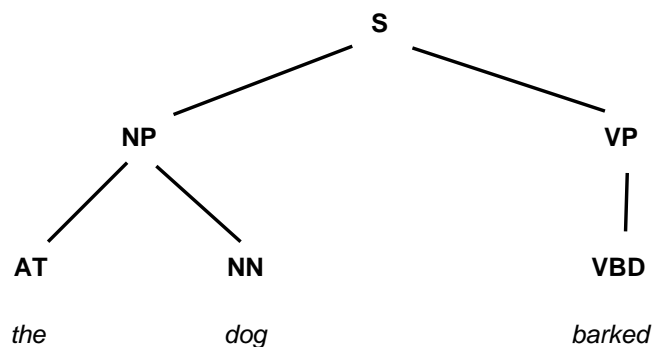


Figure 2.2: Parse tree of the sentence "the dog barked"

It has been shown that most of the syntactic measures are not particularly accurate in predicting readability [PN08] and they are often misleading. For example, the number of verb phrases in a sentence increases the text complexity but at the same time it makes it more appealing to adults which might prefer to have related clauses explicitly grouped together [PN08].

## Entity Coherence

Entity coherence has to do with identifying the *subject*, *object*, *other* and *not present* entities in the text and performing various measurements based on the transitions between these entity types. An experiment by [PN08] showed that such features of the text are not affecting significantly the text readability.

## Discourse Relations

Discourse relations have to do with relations/connections of words in the text. A word might be referring to a person with the use of anaphora (ex. they), to a place (ex. there), to a specific time (ex. before) or be explaining a cause (ex. so that). Readability measurements can be made on the number of such relations over the number of words [PN08] or by measuring conceptual overlap between sentences, paragraphs and the entire text [MDA]. Such measurements have been good estimators of text readability and they require advanced Natural Language Processing (NLP) techniques to find these types of relations in the text. Unfortunately, existing NLP tools are limited to the most commonly used languages: English, German, French and Spanish.

An ambitious research has been made at University of Memphis and a software tool (Coh-Metrix) has been developed that uses cohesion and coherence measures (discourse relations) along other measures to predict text readability. See [MDA] for more details on the methods being used in the software.

## 2.3 Content Extraction

News articles appear on the web in HTML form. The problem with this representation is that lots of advertisements, comments, images, links and forms are present and extracting the main body of the article is a quite challenging task. We need to extract the body in order to be able to measure correctly the text readability, or else many irrelevant words will be affecting our results.

Various algorithms have been invented for Content Extraction and they compare mainly on *speed* and *accuracy*. The most relevant (to extraction of news articles) techniques are listed below:

- *Body Text Extraction (BTE)* [FKS01]: It flattens the DOM tree<sup>4</sup> into a sequence that has two kinds of tokens, words and tags. This algorithm tries to find an area that contains as many words as possible and as few tags as possible.

---

<sup>4</sup>Document Object Model - Tree structure representation of HTML content

- *Content Code Blurring (CCB)* [Got08]: Similar concept with BTE but uses image blurring techniques to blur the final sequence and more successfully identify the body of the text.
- *CoreEx* [PP08]: DOM based algorithm that measures the number of links and words in each node and finds the node with the biggest amount of text and less amount of links. Note that this algorithm doesn't need to serialise the DOM tree.

## 2.4 Collaborative Filtering

*Collaborative Filtering* (CF) is a widely used algorithm for producing recommendations based solely on other user's ratings. "Through the ratings it finds users that are have similar tastes as you have, looks for other things they like and combines them to create a ranked list of suggestions" [Seg07]. CF involves a heavy computation that needs to be performed each time the rankings for the items change.

A straightforward implementation of this algorithm has computational complexity in order  $O(m^2n)$  where  $m$  is the number of users and  $n$  is the number of items on the system. Various sampling methods have been developed (randomly picking a subset of users and items for the CF computation) but they have poor accuracy.

To deal with this problem an Incremental Collaborative Filtering algorithm has been proposed by [DE05] which reduces the cost of computation to  $O(mn)$  and makes it very attractive for use in web-based applications. Google published their implementation of a distributed collaborative filtering approach that uses MinHash, Correlation and PLSI algorithms for recommending news articles to users. For more information about these algorithms can be found in their publication [DDGR07].



## Chapter 3

# Retrieving Articles

The core resource of Readabix are articles gathered from the daily news papers in different languages. Readabix periodically checks a list of RSS Feeds and gets the address of the newly posted articles. It then downloads each of those articles, stores them and processes them. It extracts the text content out of the HTML page, cleans it and breaks it into paragraphs. It identifies then the words of each paragraph (removes non-significant words) and stores them in a *reverse index*. When the text processing finishes, the system evaluates the difficulty of the page for each user.

### 3.1 RSS Feeds

RSS (*Really Simple Syndication*) is a Web content syndication format mainly used by news-like sites (news websites, blogs etc) to provide updates on the latest content posted on the site. We call RSS Feed the web page in RSS format that can be accessed by a URL. It is common for websites to provide a link to a RSS Feed on their HTML page in a similar way to the examples shown in figure 3.1.



Figure 3.1: Example links to RSS Feed

RSS-aware programs, called *News Aggregators*, check periodically the content of RSS Feeds (see example content in figure 3.2) and retrieve information about the latest content updates.

Readabix has a list of pre-specified feeds in each language. It uses them to get the title, description, publication date and URL link to the newly posted articles. Figure 3.2 highlights the data we collect from the RSS feed.

```
▼ <rss version="2.0">
  ▼ <channel>
    <generator>NFE/1.0</generator>
    <title>Health - Google News</title>
    <link>http://news.google.com?pz=1&ned=uk&hl=en</link>
    <language>en</language>
    <webMaster>news-feedback@google.com</webMaster>
    <copyright>&copy;2010 Google</copyright>
    <pubDate>Mon, 07 Jun 2010 13:42:40 GMT+00:00</pubDate>
    <lastBuildDate>Mon, 07 Jun 2010 13:42:40 GMT+00:00</lastBuildDate>
    <image>...</image>
    ▼ <item>
      <title>NHS 'should pay people' to lose weight - Telegraph.co.uk</title>
      <link>...</link>
      <guid isPermaLink="false">tag:news.google.com,2005:cluster=17593757996749</guid>
      <category>Health</category>
      <pubDate>Mon, 07 Jun 2010 06:32:03 GMT+00:00</pubDate>
      <description>...</description>
    </item>
    ▼ <item>
      <title>80 IVF fetuses are aborted a year, figures show - BBC News</title>
      <link>...</link>
      <guid isPermaLink="false">tag:news.google.com,2005:cluster=17593757443532</guid>
      <category>Health</category>
      <pubDate>Mon, 07 Jun 2010 10:49:04 GMT+00:00</pubDate>
      <description>...</description>
    </item>
    <item>...</item>
    <item>...</item>
    <item>...</item>
    <item>...</item>
    <item>...</item>
    <item>...</item>
    <item>...</item>
    <description>Google News</description>
  </channel>
</rss>
```

Figure 3.2: Example content of RSS Feed (taken from Google News). The highlighted text indicates the data Readabix stores

User can also specify feeds of his interest and put them under a category of his choice. Figure 3.3 shows the interface of the "Settings" page that allows the user to paste in a RSS Feed URL, choose one of the existing categories (or create a new one) and add it to the system. Then all the articles of this feed will appear under the selected category.



Figure 3.3: Interface that allows users to add their own feeds

We have to mention that a RSS feed contains at most  $N$  articles (the latest ones). That means that some of those articles might be already in our system, since they might still be in the list of the latest articles. It also means that if we don't check the RSS feed often enough, we might miss some articles.

Having this in mind, Readabix updates the feeds in a different process (cron job) than the rest of the processing. This allows the updating cron job to be quick. Each time the updating script is executed, the feeds that were least recently updated are checked first. When a new article is found, we store its title, link, description and publication date and we mark the article as "ready to be downloaded". From this point and on the Article processing begins.

## 3.2 Articles

The articles that we gather with the RSS feeds need to be processed before they become available to the user. Article processing is done in six steps: Download HTML, Extract Content, Tokenisation, Word Filtering, Count Word Frequency, Create Reverse-Index and Evaluate Difficulty. The rest of this chapter explains each of these steps, through an example. The example article we will retrieve is shown below in Figure 3.4.

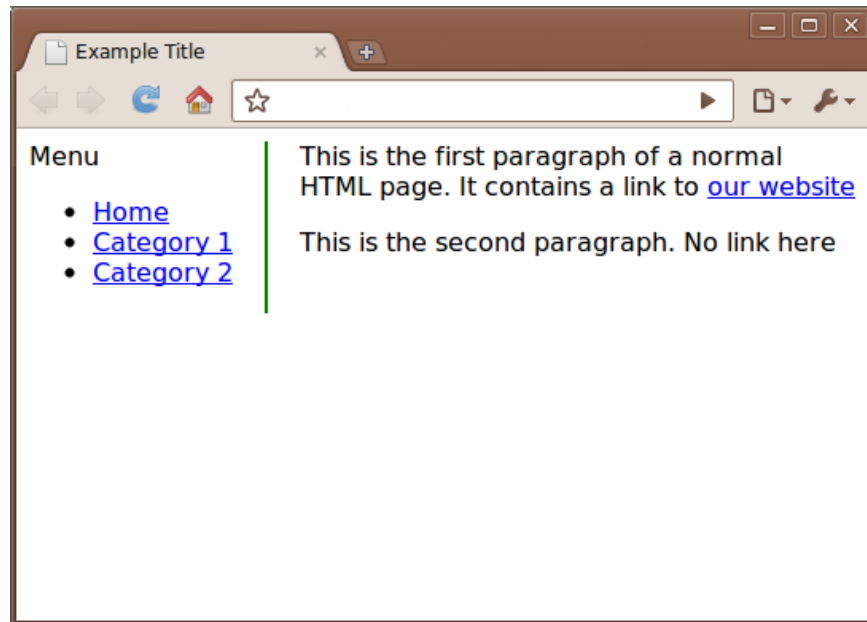


Figure 3.4: Example article we want to retrieve

## Download HTML

On this first step we download the HTML article and store it on the database. If the article is written in a different encoding than UTF-8 (detected by reading the `<meta charset = "encoding">` HTML tag), we convert it to UTF-8. This works in 95% of the cases.

Some of the articles (around 10%) can not be retrieved. This might happen because the article's URL might have changed (invalid URL) or the website might be unavailable at that time. Each article that fails to download is given other two chances. If it fails, it gets deleted from the database.

Figure 3.5 shows the HTML code of the example article we retrieved.

```
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8">
  <title>Example Title</title>
</head>
<body>
  <div style="float:left;border-right:2px green solid;
            margin-right:20px;padding-right:20px">
    Menu
    <ul>
      <li><a href="/homepage">Home</a></li>
      <li><a href="/category1">Category 1</a></li>
      <li><a href="/category2">Category 2</a></li>
    </ul>
  </div>
  <p>
    This is the first paragraph of a normal HTML page.
    It contains a link to
    <a href="http://example.com">our website</a>
  </p>
  <p>
    This is the second paragraph. No link here
  </p>
</body>
</html>
```

Figure 3.5: HTML code of the example article

## Extract Content

The second step is to extract the content out of the HTML page. To do that we use the weighted scoring, subset selection CoreEx algorithm described in [PP08]. We remove scripts, style and form tags and convert the HTML to XHTML using the `HtmlCleaner`<sup>1</sup> library prior to running the algorithm. According to [PP08], this implementation could give **precision** up to **0.9685** and **recall** up to **0.9868**.

In simple words, CoreEx measures how many words and links each node has under it and tries to find the set of nodes (at the same level in the tree structure) that maximises the amount of words over the number of links (words - links). Note that each link counts as 1 word (the words inside the link element are ignored). When we get that set of nodes, we traverse them and we transform each `<div>` and `<p>` elements into a paragraph in our extracted text. Each paragraph is separated by two end of line characters `"\n\n"`.

In our example, the `div` node has 1 word ("Menu") and 3 links (score=1-3=-2). The first paragraph node (`p`) has 15 words and 1 link (score=14). Second paragraph has 8 words and no links (score=8). The `body` node has score=-2+14+8=20. The algorithm will choose the set of nodes (at the same level)

---

<sup>1</sup>HtmlCleaner - <http://htmlcleaner.sourceforge.net/>

that maximise the score. In this case it will be the first and second paragraph since they score together 22 points. Listing 3.1 shows the text that CoreEx extracts out of the example page.

---

```
This is the first paragraph of a normal HTML page.  
It contains a link to
```

```
This is the second paragraph. No link&nbsp;here.
```

---

Listing 3.1: Extracted text of the example article

Our implementation of CoreEx performs quite well on the majority of the articles. The hardest case are articles with many comments from users, which in some cases are longer than the text itself. This makes CoreEx to extract the comment rather than the article. This is not absolutely bad in our case, since comments are written in simple language and therefore are easy to read. The only issue is that you don't know what the article is about when you read the comment and we can't inform the users that they are reading a comment rather than the article itself.

Another hard case are articles that contain tables. They appear often in Technology news where they list specifications for devices and in Business where they list some figures. Currently the implementation ignores tables, but they could be considered in the future.

After the text is extracted by CoreEx, we run some regular expressions to correct some common problems that occur frequently:

- Empty or single word paragraphs.
- Replace HTML tokens with their text value (ex "&nbsp;" with single space).
- Remove double spaces.
- Remove words consisted by more than one non alphanumeric characters (ex. "[/\_").

In our example, the whitespace at the beginning of each paragraphs is removed. The new line that was in the first paragraph is also removed similarly (whitespace). Finally the HTML token "&nbsp;" is replaced with the space character. Listing 3.2 below shows the cleaned up text.

---

```
This is the first paragraph of a normal HTML page. It con-  
tains a link to
```

```
This is the second paragraph. No link here.
```

---

Listing 3.2: Extracted and cleaned text of the example article

The cleaned up version of the extraction is then saved in the database and is used in the subsequent steps.

## Tokenisation

Tokenisation is the process of breaking up the text into meaningful words (tokens). Each language has different tokenisation rules. Some words might have no meaning by themselves and need to be combined with the previous/next word. Tokenisation might also vary across different applications.

For example, in an application that converts text into logical predicates, the token *"pre-defined"* is more useful if it is split into *"pre"* and *"defined"*. In our case, the token *"pre-defined"* should be considered as a single word, which will give more accurate translations to the reader while he reads it.

The tokenisation we use in Readabix is language dependent (different tokeniser for each language). We use the SimpleTokenizer provided by the OpenNLP<sup>2</sup> project as a base and we customise for each language. OpenNLP provides us trained tokenisers for English, German and Spanish, which could be used as a base for languages with similar punctuation symbols and structure.

A possible feature for tokenisation would be to convert each word into its root. This process, known as *stemming* or *lemmatisation* in NLP and *normalisation* in Information Retrieval. For Information Retrieval purposes, like searching, the word *"scared"* and *"scaring"* are the same. For us is not. A user might know *"scaring"* and not be able to recognise *"scared"* because he hasn't learned the past tense. English has simple word structure, but some other European languages have very different forms of the same word, which makes it "not the same" for the user.

To understand better the concept of tokenisation we show in the listing 3.3 below the tokenised version of the cleaned up text of listing 3.2. Each word of each paragraph is separated into a string (token) and is placed in a list of strings in the order they appear in the text.

---

```
[ " This", " is", " the", " first", " paragraph", " of", " a", " normal",  
"HTML", " page", ".", " It", " contains", " a", " link", " to", " This",  
" is", " the", " second", " paragraph", ".", " No", " link", " here", "." ]
```

---

Listing 3.3: Tokenisation of the example article

Obviously we don't want the "." or the "HTML" to appear as English words. Therefore we need filter out the unwanted words.

---

<sup>2</sup>Open Natural Language Processing Tools - <http://opennlp.sourceforge.net/>

## Word Filtering

Texts contain names, countries, organisations, locations, numbers etc. In our application, we want to keep track of user's knowledge of the language elements. So anything that isn't connected with the language we want to ignore it. Ideally we would want to ignore any word (or phrase) that doesn't belong to the language that the text is written into. That's quite hard to do though.

For our purposes we apply the simple rule that each word that begins with capital is considered a name, thus ignored, except of words that begin a sentence. Notice that a sentence might begin with a name and we have no way of detecting it. We could think of making a list of the names we found in previous texts, but the variety of names that could appear doesn't make it very reasonable.

Words that don't contain any characters of the alphabet and single character words are also filtered out. Note that all the capitalised words at the beginning of the sentence they get converted to lower case.

In our example the "HTML" token is removed because it starts with an upper case character and is not preceded by a ".". The words "This", "It" and "No" are converted to lower case and the dots are removed. Word "a" is removed because it's a single character word. After filtering we are left with the following set of words in listing 3.4.

---

```
[ " this ", " is ", " the ", " first ", " paragraph ", " of ", " normal ",  
" page ", " it ", " contains ", " link ", " to ", " this ", " is ",  
" the ", " second ", " paragraph ", " no ", " link ", " here " ]
```

---

Listing 3.4: Filtering of the example article

## Count Word Frequency

After filtering we get a list of the "meaningful" words in the text. We take then those words and we find out how many times each of them appears in the text. See example text word frequencies in listing 3.5 below. We then increase the occurrences of each word in a global occurrences "table" that keeps track of how many times a word appeared in all the texts. This will be used later on for recommending words (see chapter 8) and for evaluating texts (see 5.2).

---

```
{ " this ":2, " is ":2, " the ":2, " first ":1, " paragraph ":2,  
" of ":1, " normal ":1, " page ":1, " it ":1, " contains ":1,  
" link ":2, " to ":1, " second ":1, " no ":1, " here ":1 }
```

---

Listing 3.5: Words frequency map for example article



## Create Reverse-Index

In this step, we create a Reverse-Index between words and texts. That allows us to answer easily the query "which pages contain this word?". There are various methods for creating indexes (see [CDM08] for more details), but we will use a simple method that can be easily implemented with conventional databases. We create 2 records for each word in a page. One for the paragraph the word occurs and one for the text.

1.  $\langle \text{page\_id}, \text{paragraph\_no}, \text{word}, \text{occurrences\_in\_paragraph} \rangle$
2.  $\langle \text{page\_id}, \text{null}, \text{word}, \text{occurrences\_in\_text} \rangle$

To get the texts that a word appears in, we query with:

```
SELECT page_id WHERE word =  $\langle \text{word}_x \rangle$  AND paragraph_no = null
```

The first tuple can be also used to get the words in a paragraph of a text (note: this is not a reverse-index query):

```
SELECT word WHERE page_id =  $\langle \text{page\_id}_x \rangle$  AND paragraph_no =  $\langle \text{paragraph\_no}_x \rangle$ 
```

For optimisation, we define an index on word and paragraph\_no on our underlying database system. This is not the best way of creating a reverse-index because it uses a lot of space. Another method we considered, was to create one tuple per word and inside it have a list of texts. This is more space efficient but it makes queries and updates harder (ex. get occurrences of a word in a text, remove a text from the system).

## Evaluating Difficulty

The whole point of processing the articles is to recommend them to users for reading. This step makes it possible by evaluating the difficulty of the text for each user. For details on how difficulty is evaluated and the measures listed below see chapter 5.

For each user we then create a tuple that holds difficulty measures that can be queried and return results to the user. The tuple structure is:

```
 $\langle \text{user\_id}, \text{page\_id}, \text{language}, \text{category}, \text{measure1}, \dots, \text{measureN} \rangle$ 
```

[figure: use the example above to show the measures]

The current version of this tuple contains the following measures:

- **read\_percentage** : How much of the text has been read by the user.
- **known\_words** : How many words the user knows in this text.

- **unknown\_words** : How many words the user doesn't know (and we know he doesn't).
- **words** : How many words the text has.
- **unique\_words** : How many unique words the text has.
- **word\_frequency** : *Word Frequency* measure (section 5.2).
- **word\_repetition** : *Word Repetition* measure (section 5.2).
- **easy\_mode** : *Easy Mode* measure (section 5.3).
- **challenging\_mode** : *Challenging Mode* measure (section 5.3).

The reason we keep so many measures in a tuple is to speed up the updating of the difficulty measures (see section 5.5). When for example, when the user reads a paragraph, the *read\_percentage* of that page will change, along with the *known\_words* and *unknown\_words*. By fetching this tuple we can update all at the same time and re-calculate the *easy\_mode* and *challenging\_mode* measures which are used later on for recommending articles to the user.

## Chapter 4

# Reading Articles

The primary purpose of Readabix is to read! We find articles that users can read. How the article is presented and the reading experience our users get is very essential. In this chapter we discuss the requirements we set and our solution. Additionally, we explain what happens when users give feedback while reading the text and how we instantly update article recommendations.

### 4.1 Requirements

Before designing our solution we set the following requirements:

1. The text should be displayed in an easy to read font and text size.
2. The original text title should be displayed on the top.
3. We should provide a link to the original article.
4. User should be able to quickly get translations of words he doesn't know.
5. User should be able to easily mark words he knows or doesn't know.
6. User should be able to interrupt and continue reading a text at a later time (all words he marked and translated should be available next time he opens the text).
7. User should be able to give feedback on how easy/hard was this text for him.
8. User interaction with the text should be fast and informative (let the user know when the system is processing, or when a problem occurred).

## 4.2 Possible Solutions

We considered many possible solutions for each requirement. In this section we outline the pros and cons of some solutions. We focus on the three trickiest parts: *Translating, Marking, Saving*.

### Translating Words

We are looking for a solution that will be quickly giving translations to the user, so the user can read and understand the meaning of a sentence, even if it has a few unknown words.

**Option 1:** *Show a list of unknown words on the side of each paragraph.*

- **Pros:** User can see all the unknown words of this text in one place, go through them and refresh them in his mind.
- **Cons:** Hard to find the translation of a word while reading the text. Would have to find it on the list of unknown words.

**Option 2:** *Translation pops up when the user clicks on a word.*

- **Pros:** User can continue reading the sentence without too much distraction.
- **Cons:** If you show the translation by clicking, how will you mark words as known/unknown?

**Option 3:** *Translation pops up when the user places the mouse over a word*

- **Pros:** We avoid the problem of the previous option. User click can be used for marking words.
- **Cons:** Translating words by accident. User wouldn't be able to freely move his mouse around the screen. It would be too disturbing popping up translations for words you don't want to translate.

**Our Solution** *Translation pops up when the user places the mouse over an unknown word*

- **Pros:** User gets translations without too much distraction. Mouse click can be used for marking words. Translations pop up only for words the user doesn't know.
- **Cons:** User can't see a list of all the unknown words with their translations.

## Marking Words

Marking words is about the user giving feedback on words he knows and words he doesn't know. Marked words should be easily distinguished. Once a word is marked, it should stay marked for the next time the user opens the text.

**Option 1:** *All words are unmarked and user marks words as known or as unknown by clicking on them.*

Once the user clicks on the word, the word switches from unmarked to unknown, to known and back to unmarked.

- **Pros:** Straightforward for the user. All words are unmarked unless he clicks on them.
- **Cons:** A lot of clicking involved. Requires 3 different ways of showing a word so the user will be able to distinguish the unmarked/known/unknown. This can be too distracting while reading the text.

**Option 2:** *User marks the words he knows. The rest remain unknown.*

All words appear as unknown. If a user clicks on a word, the word is changed in to a known word (different colour). Note: when the text opens, all the known words will be automatically marked.

- **Pros:** Simpler distinction between words. Only two categories: known and unknown words.
- **Cons:** Still requires a lot of clicking, especially at the beginning when the system doesn't know much about the user's vocabulary (known words).

**Our Solution** *All words are considered as known unless the user clicks on a word to mark it as unknown*

All words appear as known. If a user clicks on a word, the word is changed in to an unknown word (underlined). User can cancel his marking by clicking on the word once again.

- **Pros:** Less clicking than other solutions. Less marking since suggested texts wont contain too many unknown words.
- **Cons:** User needs to save the text he read in order to mark the words he knows. No immediate way of marking known words.

## Saving User Progress

User shall be able to save his work (markings) and continue reading the text in the future. He should be able to see up to where he read, or which paragraphs he read (might have skipped some paragraphs that looked big and complicated).

**Option 1:** *Progress is saved each time the user clicks on a word.*

Each time a word is marked (known or unknown), a request is sent to the server with the word and the marking. The word is saved and is considered for future recommendations.

- **Pros:** Easy for the user. Avoid the case where user forgets to save his progress (since it's saved instantly).
- **Cons:** We can't get a clear view of how much of the text the user read. It also requires a lot of clicking, since each unmarked word needs to be clicked in order to be saved.

**Option 2:** *Marked words are saved instantly and progress is saved by clicking a "Save" button at the bottom of the page.*

- **Pros:** User markings are always saved. Progress can be saved by clicking once.
- **Cons:** User must read the whole text in order to save correctly his progress. That's not always the case, since the text might become boring halfway.

**Our Solution:** *Marked words are saved instantly and progress is saved by clicking a "Save" button at the end of each paragraph.*

Note: this solution keeps the save button at the end of the text, in case the user read the whole text and wants to save it by clicking once.

- **Pros:** User markings are always saved. You can read any paragraph you want and save it without reading all the text.
- **Cons:** You must read a whole paragraph before saving it. There are cases where the beginning of a long paragraph is comprehensible and the rest isn't and user stops halfway.

### 4.3 Our Solution

Having all the requirements in mind and possible solutions, we came up with the following interface:

## NHS to face readmission penalties - Financial Times

[View Original Article](#) [Report bad page](#) Recommend to other users: ★★★★★

Hospitals will no longer be paid in full when patients are readmitted within 30 days of discharge, Andrew Lansley, the health secretary, is to announce on Tuesday. Research from the department of health, however, suggests such a penalty will not be easy to apply fairly. ○

Emergency readmissions in England have been rising for years – up from just over 350,000 in 1998 to almost 550,000 in 2007/08, or from about 7 per cent of admitted patients to around 10 per cent. ○

Save & Close

Close

Figure 4.1: Reading interface with an example text

The example (figure 4.1) is taken from the "Health" category and we show the first two paragraphs for simplicity. The first line shows the **title** of the article, taken from the feed (highlighted in figure 3.2). The second line is the *header menu* which contains a **link to the original article**, a button to inform the administrator that this page is not extracted correctly (**report bad page**) and the **stars** for rating the difficulty of the text.

The **text paragraphs** are listed under the header menu, one by one, with some space between each paragraph. At the end of each paragraph we find the button (**white circle**) that saves the user progress. After the last paragraph, the *footer menu* appears and has two buttons: "**Save & Close**" and "**Close**".

### Interactivity

Figure 4.1 shows the text as it would appear to the user when he would open it for the first time. The user then starts reading the text (whichever paragraph he wants) and clicks on the words he doesn't know. The marked words are then underlined as shown in figure 4.2.

Hospitals will no longer be paid in full when patients are readmitted within 30 days of discharge, Andrew Lansley, the health secretary, is to announce on Tuesday. Research from the department of health, however, suggests such a penalty will not be easy to apply fairly. ○

Figure 4.2: First paragraph with 5 marked words

When the user puts the mouse over an unknown word, the word translation appears under the word.

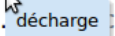
within 30 days of discharge, Andrew Lansley, the health secretary, is to announce on Tuesday.  h from the department of health,

Figure 4.3: Translation appears right under the word as the user places the mouse over the word

If the user clicks on an unknown word, the line under it disappears and the translation no longer appears when the mouse goes over it.

within 30 days of discharge, Andrew Lansley, the health secretary, is to announce on Tuesday. Research from the department of health,

Figure 4.4: Marked the unknown word as known

When a user finishes reading a paragraph, he clicks on the white circle shown in figure 4.5 below.

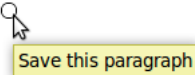
Research from the department of health,  
penalty will not be easy to apply fairly. ○  


Figure 4.5: White circle at the end of paragraph with the mouse over it

The white circle then turns into a waiting animation until the saving is finished (see figure 4.6 below).



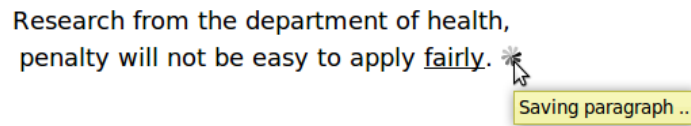


Figure 4.6: Waiting animation while saving user's progress

After the words and progress of the user has been saved, the circle turns yellow, which means that the paragraph has been saved. Figure 4.7 shows the saved paragraph. Note that if the user wants to save the paragraph again (if he made any changes to his markings) then he can still do it by clicking the yellow circle.

Hospitals will no longer be paid in full when patients are readmitted within 30 days of discharge, Andrew Lansley, the health secretary, is to announce on Tuesday. Research from the department of health, however, suggests such a penalty will not be easy to apply fairly. ●

Figure 4.7: Paragraph has been saved and white circle turned in to yellow.

Now, if the user has read the whole text, rather than marking one by one each paragraph he can use the "Save & Close" button at the bottom. When the save button is clicked, a progress bar animation appears and the user can't modify anything, until the saving process finishes. When it finishes, the window/tab of the text closes. Figure 4.8 shows a snapshot of the saving pop-up window.

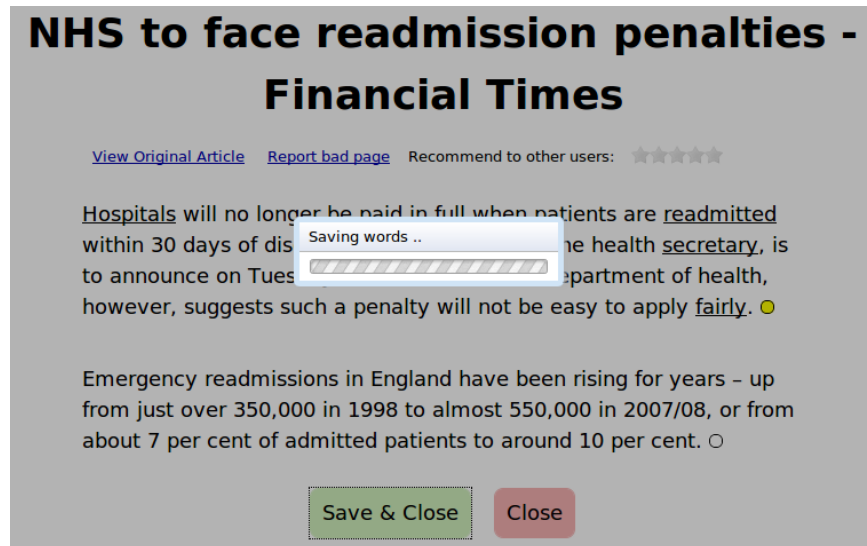


Figure 4.8: User waits for the text to be saved

If the user opens again the same text (after saving) he will still be able to see the words he marked, make changes and save the paragraphs again. Note: all the paragraphs have been saved, thus all the circles will be white (as shown below in figure 4.9).

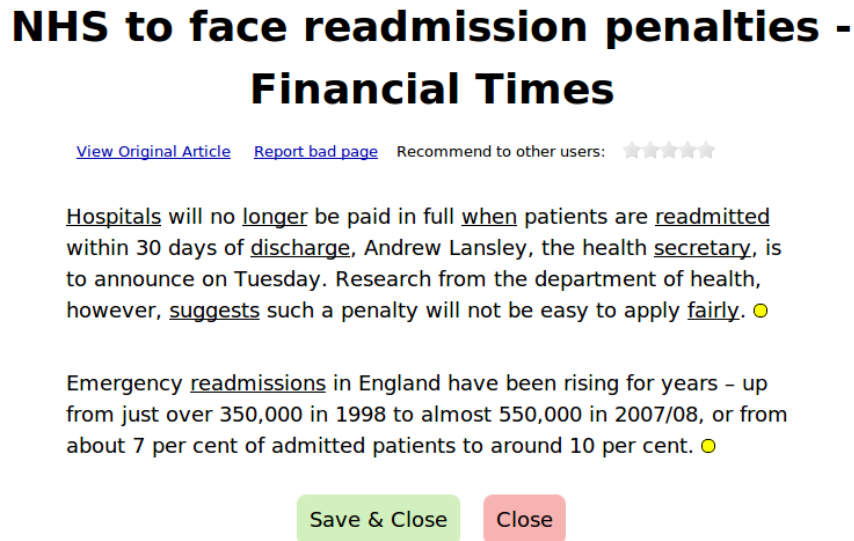


Figure 4.9: User opens again the saved text

## Translation

For translation we use Google Translate<sup>1</sup> tool, through the Google AJAX Language API<sup>2</sup>. Google translate currently offers translation between 51 language pairs. When the user requests a translation we do one of the two:

- If we translated the same word in the text, we show the translation, which we stored locally on the browser.
- If not, we make a request for translation to the URL below and once we get it we store it locally (to minimise the number of hits to the server).

The URL we use for translation is:

```
http://ajax.googleapis.com/ajax/services/language/translate?v=1.0&q=<word>&langpair=<from\_lang>%7C<to\_lang>
```

where

⟨ **word** ⟩ is the word we want to translate.

⟨ **from\_lang** ⟩ is language you want to translate from.

⟨ **to\_lang** ⟩ is language you want to translate to.

Note that the Google Translate API can translate whole sentences and paragraphs. Translating a whole sentence gives more accurate word translation since it knows the context where it appears (called *word disambiguation* in Natural Language Processing).

## Saving Marked Words

Each time a user clicks on a word, we update an entry on the database, marking the state of this word (known/unknown) in the current text. The tuple looks like this:

⟨ **user\_id**, **page\_id**, **language**, **word**, **known**, **marked\_date** ⟩

**known** Is the word known or unknown? Note that this value can be *null*, which is the case we don't know anything about this word.

**marked\_date** Date and time the word was marked.

**Note:** These entries are used only for keeping track of which words the user marked in this text. They don't affect article recommendations or anything else.

---

<sup>1</sup><http://translate.google.com/>

<sup>2</sup><http://code.google.com/apis/ajaxlanguage/>

## Saving Paragraph or Text

When the user requests to save a paragraph or the whole text, we do the following:

1. Retrieve (from database) **all words** of the paragraph/text we want to save.
2. Retrieve the **marked words** (known/unknown) in this paragraph/text.
3. Mark all **unmark words** in the paragraph/text as **known** (update database).
4. Mark all the **known** words in this text as **known** for the user.
5. Mark all the **unknown** words in this text as **unknown** for the user.

Note: This is a simplified version of what really happens. We will see more details about how does this knowledge affects article recommendations (see chapter 5).

We mark words as known for the user in the same tuple structure but with the *page\_id* field as *null*.

$\langle \text{user\_id}, \text{null}, \text{language}, \text{word}, \text{known}, \text{marked\_date} \rangle$

This structure represents the knowledge of each user, in other words, the words he knows and words he doesn't. This simple structure is the basis of the user profile and is used for recommending articles to other users at the same user's level (see chapter 7). Its also used for keeping track of user's progress (see chapter 6).

## 4.4 Performance

When a user waits for a computation to be completed, it better be quick! Users don't like waiting and they will stop using a system if it is not highly responsive. Having this in mind, we analyse the performance of our operations:

Table 4.1: Performance analysis of reading article operations

Operation	Time	Time [Cold Start]
Load article	500-700ms	2-8s
Mark a word	300-500ms	900ms-6s
Translate a word	200-400ms	700ms-6s

By this table we observe that the only "worrying" waiting times are the *Cold Start* waiting times. Cold start means that the process responsible for serving the request is inactive/sleeping and it takes some time to load and start serving requests. All the subsequent requests (if not too much time passed by) will be served at the normal service times shown in the middle column.

The performance of the *saving operation* (paragraph/text) is cleared out in the Software Engineering chapter 8, where we take into consideration batch operations that can speed up the retrieval of the articles which need to be updated.

It's important to mention here that the user can continue reading, marking and translating words *while a paragraph is being saved*. That means that the saving operation doesn't affect directly the user's experience, since it's done in the background. Also, more than one paragraph can be saved concurrently (no need to wait for the previous paragraph to finish saving).

## Chapter 5

# Recommending Articles

Readabix helps users find articles they can read. When a user joins the website for the first time, he gets recommendations of texts in various languages. The system knows nothing about the user, so it recommends articles that could possibly be easy to read. These recommendations are based on *User Independent* measures. As the user starts reading and marking words, the system improves the recommendations. If the texts that the user read were too hard, the system will recommend articles that are easier to read (less unknown words). If the texts were too easy, user has the option to get more challenging texts (more unknown words). The easy and challenging modes are based on *User Dependent* measures.

This chapter presents the *User Independent* and *User Dependent* measures we used to create the *Combined* measures that we use for recommendations. At the end of the chapter we discuss performance issues and how we tackle them.

### 5.1 Interface

Before diving into maths and measures, let's see how we recommend the articles to the user. When the user enters the website, a list of articles, grouped into categories appear in the default language. The page looks like the figure 5.1 below.

Readabix

[Read Fill up the survey](#) [Evaluation Instructions](#) [Logout](#)

Spanish ▾

**All**  
Read  
Business  
Spain  
Entertainment  
World  
Sports  
Technology

**Business**

- [El Ibx prolonga su rebote y suma la segunda mayor subida anual - La Vanguardia](#)
- bolsa continuó hoy el que inició en la sesión de ayer. , y subió el 3,95 por ciento, la mayor del año , animada por el avance de la banca. Así, el índice de referencia del mercado es...
- [Así queda el mapa de las cajas de ahorro - Finanzas.com](#)
- [Barajas gana un 5% de pasajeros en mayo a pesar de la nube de cenizas - ABC.es](#)
- [Los precios se mantuvieron estables en Extremadura en mayo - ABC.es](#)
- [El Sabadell descarta pedir ayudas para adquirir el Guipuzcoano - El País.com \(España\)](#)
- [more »](#)

**Entertainment**

- [450 años de Estrella por Sevilla - Arte Sacro](#)
- 450 años de Estrella por Sevilla . La Hermandad de la Estrella llegó ayer, sábado 12 de junio, al punto culminante de los actos del 450 aniversario fundacional. A las 9,30 horas en la parroquia de...
- [La Rioja ha logrado un Manojó de Plata en tintos reserva - ABC.es](#)
- [Castilla y León y Castilla La Mancha empatan con 9 galardones en... - ABC.es](#)
- [La Virgen de la Estrella recibirá hoy su mejor regalo de cumpleaños - La Pasión Digital](#)
- [La bandera multicolor conmemora el Día del Orgullo Gay en el ... - Diario de Sevilla](#)
- [more »](#)

**Spain**

- [El Consistorio, distinguido otra vez con la "Escoba de Platino" - El Diario Montañés](#)
- El Ayuntamiento de Laredo ha vuelto a ser distinguido con la "Escoba de Platino", la más alta distinción que concede la Asociación Técnica para la Gestión de Residuos y Medio Ambiente -Miembro Naciona...
- [La depresión del PSOE - ABC.es](#)
- [Tarazona recibe el premio nacional 'Escoba de plata 2010' por su ... - Qué.es](#)
- [3.000 militares contra los incendios - El País.com \(España\)](#)
- [El PsdeG no consentirá "ni una" al PP ni le permitirá que "entre ... - Qué.es](#)
- [more »](#)

**World**

- [Intermón señala que la ayuda a los palestinos no mejora su situación - elmundo.es](#)
- La pobreza aumentó entre 1998 y 2007 de un 20% a un 57% de la población La ayuda española a los territorios palestinos aumentó un 109% entre 2007 y 2008, llegando a los 111 millones, pese a ello ...
- [Moratinos pedirá a la UE que exija a Israel el fin del bloqueo - ABC.es](#)
- [Jornada más violenta de México desde el 2006 - Prensa Latina](#)
- [Presidente de Ecuador destaca libertad de expresión ciudadana - Prensa Latina](#)
- [El Papa pide perdón a las víctimas de los abusos y promete que ... - 20 minutos](#)
- [more »](#)

**Learned Words (0)**  
none

**Unknown Words (0)**  
none

**Known Words (0)**  
none

Figure 5.1: User interface for recommending articles

On the left of the articles there is a list of categories. When the user clicks on one of them, a list of articles from that category appears. On the top left of the page (under the logo), user can select the language of his choice from the drop-down menu. The biggest part of the page is the article view which shows the list of articles. Each article's title is listed as a link that by clicking it the user can start reading the article's text. A small summary (first 200 words) of the first text of each category appears right under the title. At the left of the title, the white circle shows how much of the text has been read by our user. The circle becomes more yellow as bigger percentage of the text is being read. On the right of the page we can track our progress in learning words and get word recommendations (see chapter 6 for more details).

## 5.2 User Independent Measures

As we mentioned earlier, User Independent measures are useful for recommending articles when we know nothing about the user. They are based on simple statistics. The aim of these measures, is to recommend texts that are *possibly* easy to read. That us not guaranteed, since these measures are not very accurate. The two measures we use are *Word Frequency* and *Word Repetition*.

Before we present the measures, let's define some symbols that will be used in the rest of the chapter. First is the occurrence of a word which equals to:

$occurrence(w, t) = \text{how many times the word } w \text{ appears in the text } t$

By knowing the occurrence of each word in each text we define the word frequency as the sum of all occurrences of that word in all texts we retrieved so far.

$$frequency(w) = \sum_{t \in texts} occurrence(w,t) \quad (5.1)$$

Maximum frequency will be the frequency of the most frequent word (ex. word "the" in English).

$$frequency_{max} = \max_{w \in words} frequency(w) \quad (5.2)$$

The maximum frequency is used to give a normalised measure of each word's frequency. The normalised measure range is [0..1]. Apparently the most frequent words will have measure closer to 1 and less frequent words will have measure closer to 0.

$$frequency_{normalised}(w) = \frac{frequency(w)}{frequency_{max}} \quad (5.3)$$

Before we move on, let's define word\_count as the number of words in a text:

$$word\_count(t) = \sum_{w \in words(t)} occurrence(w,t) \quad (5.4)$$

## Word Frequency

*Word Frequency* measure is based on the assumption that

**The more frequent a word is, the more probable is that the user knows that word.**

By frequent, we mean the *amount of times a word appears in all the articles* we retrieved so far. This works with words that appear often in news articles, but there are many other words that don't appear and are well known by many users. In English, the word "**president**" appears to be very frequent, where the word "**baby**" is not.

We define the Word Frequency measure as the average of the normalised frequency of each word in the text. Note that we need to multiply the normalised



frequency by the occurrence of the word, since  $words(t)$  is a set (contains each word once).

$$word\_frequency\_measure(t) = \frac{\sum_{w \in words(t)} occurrence(w, t) * frequency_{normalised}(w)}{word\_count(t)} \quad (5.5)$$

The range of this measure is  $[0..1]$ , since each frequency is normalised from  $[0..1]$ . Texts with high *word frequency measure* are considered easier than texts with low measure.

## Word Repetition

*Word Repetition* measure is based on the assumption that:

**Texts with high word repetition are easier to read.**

By word repetition we mean the amount of words that appear more than once in the text. Intuitively, when a word appears many times in the same text, even if we don't know that word, by translating it once, we can remember the meaning the next time we see it. It helps also with learning the word by repeatedly seeing it and remembering the meaning.

The formula for calculating word repetition is simple: we divide the amount of words in the text by the number of unique words. This gives us on average of how many times each word appears in the text.

$$word\_repetition(t) = \frac{word\_count(t)}{|words(t)|} \quad (5.6)$$

Using the *word\_repetition* we define the Word Repetition measure as

$$word\_repetition\_measure(t) = 1 - \frac{1}{word\_repetition(t)} \quad (5.7)$$

The measure range is  $[0..1)$ . A text with no word repetition will have *word repetition measure* = 0. The higher the repetition in a text, the higher *word repetition measure* will be (closer to 1).

There is an issue with this measure though. It is *biased* towards larger texts. The larger the text is, the more probable it is that words will repeat. This can be improved by measuring the word repetition of each paragraph and taking the average of it.

Thus, we use this formula for calculating word repetition.

$$word\_repetition_{unbiased}(t) = \frac{\sum_{p \in paragraphs(t)} \frac{word\_count(p)}{|words(p)|}}{|paragraphs(t)|} \quad (5.8)$$

### 5.3 User Dependent Measures

User Dependent measures are based on the user's feedback on which words he knows and which ones not. Based on this feedback, each text can be split into three groups of words as shown in figure 5.2.

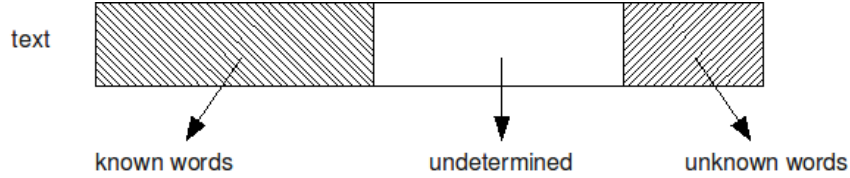


Figure 5.2: Division of a text into known, unknown and undetermined words

*Undetermined words* are the ones we don't know yet their classification. Some of them might be known to the user and some not. These words haven't been seen by the user in any previous text. Apparently, the amount of undetermined words will be decreasing as user reads more texts. There is always though the possibility that a word appears only in one text and therefore it will remain undetermined until the user reads that text.

In the following sections we present the measures we use, based on this classification of words. Before we do that, we define  $known\_words\_count(u, t)$  as the amount of words the user  $u$  knows in text  $t$ .

$$known\_words(u, t) = \sum_{w \in known\_words(u)} occurrence(w, t) \quad (5.9)$$

Similarly, the amount of unknown words in text  $t$  is defined by

$$unknown\_words(u, t) = \sum_{w \in unknown\_words(u)} occurrence(w, t) \quad (5.10)$$

## Known Words

Known words measure is based on the assumption that:

**The more known words, the easiest will be to read.**

This measure works fine if the user specified all the words he knows. But often (if not always) this is not the case. Users learn new words that the system isn't aware of them.

$$known\_words\_measure(u, t) = \frac{known\_words(u, t)}{word\_count(t)} \quad (5.11)$$

This measure ranges from  $[0..1]$ , where the higher the measure, the easier the text.

## Unknown Words

The Unknown Words measure comes in to improve the Known Words measure by including the unknown words. This measure is based on the assumption that:

**Some of the unknown words are known to the user (user learned them).**

When a user marks an unknown word, he sees the translation and tries to learn the word. Then that word will be easier for the user, since he has seen it before.

$$unknown\_words\_measure(u, t) = \frac{known\_words(u, t) + unknown\_words(u, t)}{word\_count(t)} \quad (5.12)$$

Unknown words measure has a drawback. A text with  $K$  known words and  $0$  unknown words is evaluated the same with a text with  $0$  known words and  $K$  unknown words. This drawback leads us to the Easy Mode measure (see next).

## Weighted Mode

The Weighted Mode measure comes in to improve the Unknown Words measure by adding a parameter  $A$  that controls the significance of known and unknown words.

$$\textit{weighted\_mode}(u, t, A) = \frac{A * \textit{known\_words}(u, t) + (1 - A) * \textit{unknown\_words}(u, t)}{\textit{word\_count}(t)} \quad (5.13)$$

This parameter gives us the flexibility to control the recommendations. By increasing parameter  $A$ , we get texts with more known words (probably easier). By decreasing it we get texts with less known words, more unknown words, therefore more challenging.

### Easy & Challenging Mode

Easy and Challenging modes are simply the Weighted Mode measure with different parameter  $A$ .

$$\textit{easy\_mode\_measure}(u, t) = \textit{weighted\_mode}(u, t, \mathbf{0.6}) \quad (5.14)$$

$$\textit{challenging\_mode\_measure}(u, t) = \textit{weighted\_mode}(u, t, \mathbf{0.3}) \quad (5.15)$$

In the current version of Readabix, Easy Mode uses  $\mathbf{A=0.6}$  and Challenging Mode  $\mathbf{A=0.3}$ . Note that Challenging Mode aims to recommend articles that user will learn more words by reading them. These articles will contain many words that the user has translated before, therefore he gets another chance to learn them. If we would want to get hard texts, we would define a *hard mode measure* = 1 - *easy mode measure*.

## 5.4 Combined Measures

User Independent measures are good for recommending articles before the user has given any feedback. User Dependent measures work only after user gives feedback. The measures we present below combine these two.

The first measure gives us the recommendations for the Easy Mode and the second for the Challenging Mode.

$$\begin{aligned} \textit{easy\_combined\_measure}(u, t) = & 0.1 * \textit{word\_frequency\_measure} + \\ & 0.3 * \textit{word\_repetition\_measure} + \\ & 0.6 * \mathbf{easy\_mode}(u, t) \end{aligned} \quad (5.16)$$

$$\begin{aligned} \text{challenging\_combined\_measure}(u, t) = & 0.1 * \text{word\_frequency\_measure} + \\ & 0.3 * \text{word\_repetition\_measure} + \\ & 0.6 * \text{challenging\_mode}(u, t) \end{aligned} \tag{5.17}$$

The measures are combined with significance proportional to their accuracy. Word frequency measure gets significance of 10%, since its the least accurate. Word repetition gets significance of 30%, since its more sound an accurate. User dependent measures (easy & challenging mode) get significance of 60% and they start affecting recommendations after the first user feedback.

Note that these two measures are the only ones used for recommendations, since they combine all the measures we defined.

## 5.5 Incremental Counting

User Independent measures are computed once for each article; when it enters system. However, User Dependent measures change each time the user saves a paragraph or text. When he does that, new known and unknown words enter his profile, known words become unknown and the opposite. In order to get the updated recommendations, the combined measures need to be re-evaluated for each text.

Re-evaluating the measures is a very expensive computation. For each text we need to fetch all its words and compare them with the user's profile to determine if they are known or not. The system might contain thousands of texts and re-evaluating them would take a long time.

Having this difficulty in mind, we created an incremental algorithm that exploits the advantages of the Reverse-Index we created while retrieving the articles (see section 3.2 for more details). The Reverse-Index gives us the set of texts each word appears into.

$$\text{texts}(w) = \{t | w \in \text{words}(t)\} \tag{5.18}$$

Now, when the user saves a paragraph or text, we iterate through the saved words and we update the measures of the affected texts.

```
for w in marked_words:
    for t in texts(w):
        change_affected_text(w, t)
```

The changes we make depend on the marking and the status of the word in user's profile. Table 5.1 below lists the changes that need to be made in each case.

Table 5.1: Changes in measures of an affected text

Marking	User Profile	Change in affected texts
known	known	none
known	null	increase known words
known	unknown	increase known words, decrease unknown words
unknown	unknown	none
unknown	null	increase unknown words
unknown	known	increase unknown words, decrease known words

## Chapter 6

# Recommending Words

Apart from articles, Readabix has the ability to recommend words. Word recommendations, give another chance for the user to learn new words. If not learn, he will at least recognise them the next time he sees them. Repetition is essential for language learning. Through word recommendations we also get more feedback on the user's known words.

The chapter begins with presenting the User Interface we use for recommending words and the basic interaction it provides. We then move into defining the measures we used for recommendations and the recommendations algorithm. At the end, we talk about keeping track of user's progress with language learning and the way we detect newly learned words.

### 6.1 User Interface

Readabix provides a simple user interface for recommending words. It appears on the right of article recommendations (main page), and looks like in figure 6.1 below.

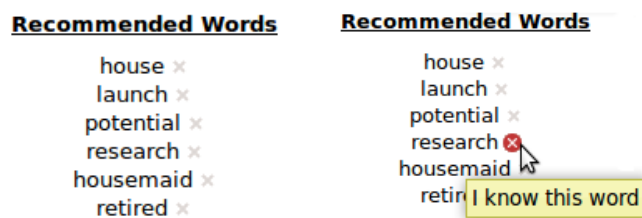


Figure 6.1: Recommended words interface

A user can remove a word from the recommendations list by clicking the "x" button. At the same time, the word is marked as known.

As expected, a new word appears at the place of the removed word. Note that the list of recommended words can change at any time, since it depends on the retrieved articles and users' profile. Both can be changed at any time.

## 6.2 Measures

Before we present our recommendation algorithm, we need to define some relevant measures that we will use. The three statistical measures we use are the following.

1. **Word Frequency** : How many times this word appears in all the texts.
2. **Text Popularity** : How many texts contain this word.
3. **User Popularity** : How many users know this word.

Intuitively, a word that appears very often (high word frequency) and in many texts (text popularity), is worth to be learned.

We use the same *Word Frequency* formula (formula 5.1) as in Article Recommendations. We normalise it in the same way (formula 5.3).

We define *Text Popularity* as the amount of texts this word appears in (similar to the Reverse-Index formula 5.18).

$$text\_popularity(w) = |\{t | w \in words(t)\}| \quad (6.1)$$

We normalise popularity by dividing each text\_popularity measure with the maximum text\_popularity measure.

$$text\_popularity_{max} = \max_{w \in words} text\_popularity(w) \quad (6.2)$$

$$text\_popularity_{normalised}(w) = \frac{text\_popularity(w)}{text\_popularity_{max}} \quad (6.3)$$

The normalised text popularity takes values from [0..1], where higher values refer to higher text popularity.

*User popularity* is defined as the number of users that know a word.

$$user\_popularity(w) = |\{u | w \in known\_words(u)\}| \quad (6.4)$$



In a similar way with text popularity we normalise the user popularity (by dividing with the maximum user popularity).

$$user\_popularity_{max} = \max_{w \in words} user\_popularity(w) \quad (6.5)$$

$$user\_popularity_{normalised}(w) = \frac{user\_popularity(w)}{user\_popularity_{max}} \quad (6.6)$$

### 6.3 Recommendations

As mentioned in the introduction the purpose of word recommendations is to

1. expose user to new words that will improve his reading skills by learning them.
2. discover words he already knows to improve article recommendations.

Having this in mind, we present two approaches we experimented with and the combined approach which we use in the current version of Readabix.

Our first approach is based on the assumption that:

*If the user learns words that appear many times (high word frequency) in many texts (high text popularity), he will eventually be able to read more texts in the system.*

This approach exposes user to new words but fails to discover words he might already know. It is also affected by the variety of texts we retrieve. In a sense, if there are more financial articles than health articles the word "commodity" will be recommended instead the word "body".

Our second approach is based on the assumption that:

*If many users know a word (high user popularity), it is more probable that our user will do the same.*

This approach, works well for discovering words user already knows and exposing user to new words. The main drawback of this approach is that it requires many users in order to distinguish the words to recommend. In other words, most words have the same user popularity and recommending ends up choosing randomly between them.

Taking into consideration the pros and cons of these two approaches, Readabix combines them into one measure, which is currently used.

$$combined\_measure(w) = 0.1 * frequency_{normalised}(w) +$$

$$0.3 * \text{text\_popularity}_{\text{normalised}}(w) + \\ 0.6 * \text{user\_popularity}_{\text{normalised}}(w) \quad (6.7)$$

Word frequency counts the least, since we want to avoid words that appear many times in few texts. User popularity is our "best" measure, so it counts the most. Text popularity will have a big impact at the early stages of Readabix. As the user base increases, user popularity will overtake text popularity.

As all the measures we use, *combined\_measure* takes values from (0..1]. In order to recommend words, we iterate through the words with the highest combined measure and we select the ones that are unknown to the user.

## 6.4 User Progress

Readabix wants to keep users motivated so they continue reading and improving their skills. One way of doing that is by showing them their progress on words they learned. But how do we detect learned words?

It's simple. Each time a user marks a word as known, if that word was previously unknown, we mark it as learned. In other words, if a user marked a word as unknown in a text and later on when he read another text he remembered the meaning of the word and marked it as known, we consider that he learned the word.

User progress is shown at the right of the article recommendations (main page), along with word recommendations. Figure 6.2 below shows the learned, unknown and known words lists.

<b><u>Learned Words (10)</u></b>	<b><u>Unknown Words (20)</u></b>	<b><u>Known Words (129)</u></b>
dwarf ×	flagging ×	boost ×
penalty ×	stumbled ×	flood ×
robust ×	traits ×	he ×
contracted ×	devastating ×	hopes ×
wisdom ×	saplings ×	land ×
verdict ×	specimen ×	new ×

Figure 6.2: User progress interface

The lists appear one under another in the website. They are listed in one row only for the purposes of the report. Note that use can give feedback in a similar way by clicking the "x" buttons.

# Chapter 7

## User Feedback

We have seen so far how Readabix improves article recommendations by gathering feedback on known/unknown words. The methods we have seen are limited to improving a single user's recommendations. Why not using this feedback for improving other user's articles? Do we need additional feedback to make it possible?

This chapter answers such questions by introducing another method of receiving feedback and how to improve recommendations of other users.

### 7.1 Motivation

We used feedback on known/unknown words for improving article recommendations. The issue with this approach is that is based on the assumption that:

*A text with many known words will be easy to read.*

But isn't there a case where the user might know many (if not all the) words and still not be able to understand the text? Authors "can make the simple things complicated and the complicated things simple". Text difficulty depends also on the subject itself. Some texts talk about complicated concepts, situations, topics etc. Other texts might be using many undefined references, many local names, local situations etc that make them harder for users from other countries.

A possible approach to this would be to use Natural Language Processing (NLP) for further analysis of the texts to find undefined references etc. This approach could improve our recommendations but its very hard to implement and extend it to many languages (lack of freely available NLP tools). Additionally, to discover local references we would need to use a database with locations, names etc of each country.

Readabix approaches this issue with the *collaborative approach*. Users decide if the text is easy or hard and let others know about it by rating it. We introduce the 5 stars rating scheme shown in Figure 7.1.

[View Original Article](#) [Report bad page](#) Recommend to other users: ★★★★★

Figure 7.1: Menu on top of each article page

The star values correspond to the following interpretations:

- 1 star** Not recommended - Hard to read
- 2 stars** Not recommended - Quite hard to read
- 3 stars** Recommended - Quite hard to read
- 4 stars** Recommended - Quite easy to read
- 5 stars** Highly recommended - Easy to read

Using this additional feedback we can get a estimate for each rated article how easy or hard it is. We can assume that:

*An article with many "easy" votes has higher probability to be easy for users that haven't read it.*

But there is an issue with this approach. What if a text is rated as "easy" by users with advanced language skills; should we recommend it to everyone else? In other words, an "easy" text for an advanced user doesn't mean it will be easy for a beginner!

A better approach would be to take into consideration ratings from users at a similar level with you. In order to do that we need a way to compare users' level.

## 7.2 Profile Matching

We present in this section the way Readabix matches profiles of users (compares users' level). By profile we mean the information we know about the user:

- known words
- unknown words
- ratings

Our first approach is based on the simple assumption that:

*Users are at a similar level if they know a similar amount of words*

A user that knows 1000 words is more probable to be at a similar level with a user that knows 1200 words rather than a user that knows 300 words. Using the above we define the comparison as:

$$\text{similarity}(userA, userB) = \frac{1}{||known\_words(userA)| - |known\_words(userB)|| + 1} \quad (7.1)$$

This is fine, but what if the words they know are different? Especially for beginners which have limited vocabulary. Its possible that some users know more words in categories in which others don't.

To face this issue we compare the amount of words they both know (common words) and we define a new similarity function ( $\text{similarity}_{words}$ ) based on it.

$$\begin{aligned} \text{common\_known\_words}(userA, userB) &= \text{known\_words}(userA) \cap \text{known\_words}(userB) \quad (7.2) \\ \text{similarity}_{words}(userA, userB) &= \frac{|\text{common\_known\_words}(userA, userB)|}{\max(|\text{known\_words}(userA)|, |\text{known\_words}(userB)|) + 1} \quad (7.3) \end{aligned}$$

This similarity function has the same properties with the first one. If the amount of words they know is very different, the measure will be lower. Additionally, if two users know the same amount of words, the measure will be higher if they share many known words.

This function is the one used by Readabix. It could be expanded to include unknown words and ratings in a similar way. We refrained doing that in order to keep the computation simple and quick. See Incremental Counting section 7.4 below for details on how we quickly compare users' profiles.

### 7.3 Recommendation Algorithm

As stated in the Motivation section, we want to give emphasis on ratings of users at the same level. One approach would be to define a threshold  $\tau$  that defines if two users are at the same level or not.

$$\text{same}(userA, userB) = \text{if } \text{similarity}(userA, userB) > \tau \text{ then } \top \text{ else } \perp \quad (7.4)$$

This would require us to discover the right threshold and creates the problem of not finding any users at similar level. In case no user at similar level exists, with this approach we would get no help from other users.

A better approach would be to use all user's ratings, but each rating will have a different weight, depending on how similar the two users are. This way, if no users exist at the same level, we would get recommendations from users from other levels (lower and higher). The closer another user's level is, the more his vote counts. Therefore, for each rated article, we define its *weighted\_rating(u, t)* by:

$$rated\_by(text) = \{u \in users | rating(u, text) > 0\} \quad (7.5)$$

$$weighted\_rating(user, text) = \frac{\sum_{u \in users} similarity_{word}(user, u) * rating(u, t)}{|rated\_by(text)|} \quad (7.6)$$

The weighted rating measure returns values from [0..5]. The higher the value the better the recommendation will be for the user. The algorithm calculates the weighted rating of all the pages and returns the K best pages.

```
getRecommendations(user ,K):
    recommendations = []
    for t in rated_texts():
        S = 0
        for u in rated_by(t):
            S += similarity(user ,u) * rating(u,t)
        R = S / len(rated_by(t))
        recommendations.append( (R,t) )
    sort(recommendations)
    return recommendations [0:K]
```

The algorithm is in order  $O(T*U)$ , where T is the number of rated texts and U the number of users. Thus, complexity increases by the number of ratings in the system. This algorithm is currently used by Readabix, since the users base and amount of ratings is small.

For better performance an incremental algorithm could be considered that updates the weighted average of pages when a user's profile is changing or a page is being rated.

## 7.4 Incremental Counting

Evaluating similarity of users is a computationally intensive operation since it needs to compare the known words of all the users. Users might know hundreds or thousands of words and comparing them would take long time. To handle this, we use a similar method to the Incremental Counting (see section 5.5) to update the user similarity measures.

Each time a new word is coming into the user's known list, we find all the users that know that word and we increment their common known words. Similarly, when a known word gets marked as unknown we decrement the common known words of the other users. This way, we need to only update the user's common known words count.

Note that similarly to Incremental Counting we need to have a reverse index from known words to users. This speeds up the process of finding the users that know each word.

## Chapter 8

# Software Engineering

We wanted Readabix to scale-up easily in the number of users, articles and languages. We had this in mind from the beginning when we were choosing our tools and kept it though the whole development process. We split up the processing into small manageable units that can run concurrently on the cloud. We refactored the code and kept coupling low using popular design patterns in order to easily scale up the code. We unit tested all the algorithms and database operations. We created a modular user interface using the latest tools and patterns.

This chapter presents the Software Engineering aspects of Readabix. We start by presenting an overview of the system and the different modules. We then move to talk about deploying the service on Google's cloud and the way it affected our design. We conclude by presenting our code structure on server and client side.

### 8.1 System Overview

Readabix collects periodically news articles from the web, stores them, processes them, recommends them to users for reading and receives feedback. Figure 8.1 below shows this interaction of Readabix and the environment.



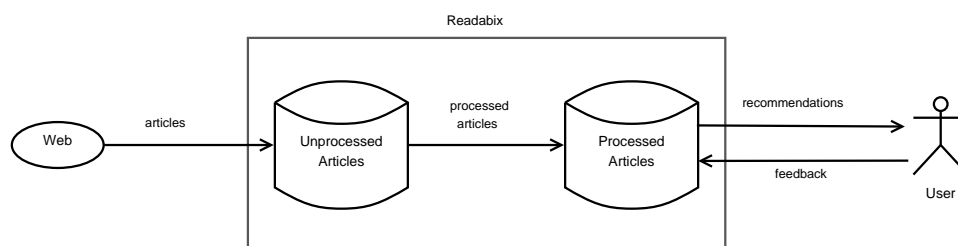


Figure 8.1: Readabix interaction with the environment

As you can see in the figure above, articles are split into two conceptual categories: *unprocessed* and *processed* articles. Unprocessed articles represent the articles that have been downloaded, extracted and stored on the database. These articles can not be recommended yet because their difficulty hasn't been evaluated. When their difficulty has been evaluated for every user in the system and a reverse index is created for them, they become *processed*. Processed articles are recommended to the users and the system receives feedback for improving the recommendations. Feedback consists of the known/unknown words and the rating we discussed in the previous chapters.

This system interaction with the environment leads us to structure conceptually the system into three main modules: *Retrieval*, *Processing*, *Feedback*.

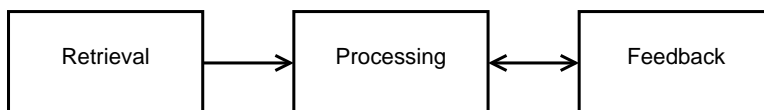


Figure 8.2: Readabix main modules

The responsibilities of these modules are:

- **Retrieval** - Find new articles, download them and extract their text.
- **Processing** - Create reverse index, count word frequencies and evaluate difficulty.
- **Feedback** - Update user's profile, recommendations and profile comparison.

Now, by knowing the role of each module we can find out where it applies in the whole system interaction. A more complete view of the system is shown below (figure 8.3).

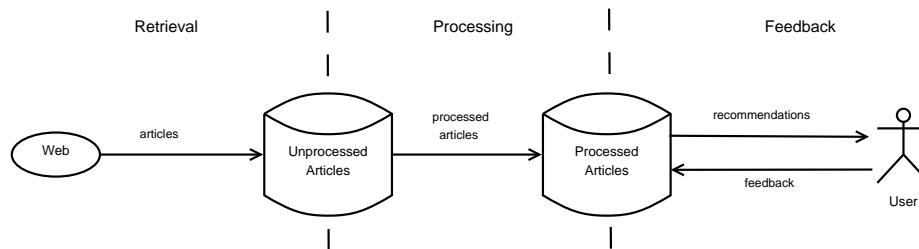


Figure 8.3: Readabix modules in the environment

We continue by breaking down each module into smaller, independent units.

## Retrieval Module

Retrieval module is consisted of three main processes:

- **Feeds Updater** - Updates feeds and discovers new articles
- **Page Loader** - Downloads articles
- **Page Extractor** - Extracts content out of HTML

It is important to mention that these processes run independently (don't depend on each other). Feeds Updater gets new articles and stores them on database. Page Loader queries periodically the database and downloads the new articles. Page extractor finds downloaded articles in the db and extracts them. Figure 8.4 below shows this process independence.

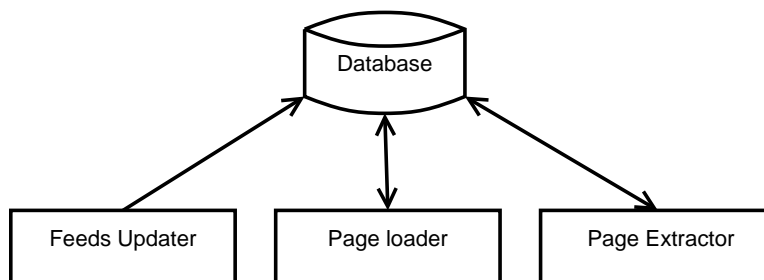


Figure 8.4: Retrieval processes

This design (independent execution) gives us the following advantages:

- We can run each of them at **different rate** (ex. Feeds Updater needs to be more often)

- We can run them **concurrently** (system can be downloading, updating feeds and extracting text at the same time).
- **Easier to make changes** to the processes (ex. If we want to use a different extraction algorithm for the articles we don't have to download them again).

The main disadvantage of this approach is that it requires **more database accesses**. For processing a single page we require 5 accesses (2 reads and 3 writes) where if they were all in one process they would require only 1 access (1 write). In order to reduce database accesses we use **batch read and write** operations (see Datastore in section 8.3) and we process many articles at each run (not only one).

## Processing Module

This module aims to process the text content, measure each text's difficulty and make it available to users through article recommendations. This module consists of the following three processes.

- **Reverse Index** - Creates database entries that allow us to easily find which texts a word appears in.
- **Word Frequencies** - Increases frequency of each word in the text.
- **Difficulty Evaluation** - Evaluate difficulty of a text for each user, using user's profile.

The processes run independently for the same reasons as the Retrieval Module's processes. Figure 8.5 below shows these three processes independent execution.

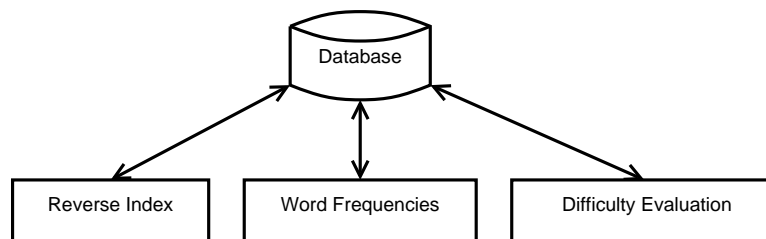


Figure 8.5: Processing Module processes

## Feedback Module

Feedback module is responsible for updating the measures that change when the user gives some feedback. It implements the incremental approaches of the

algorithms used in the Processing module. The Processing module initialises the measures that the incremental algorithms update in the Feedback module.

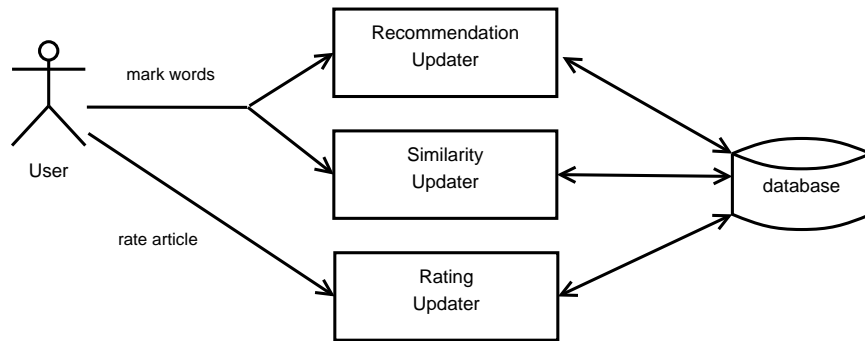


Figure 8.6: Feedback processes

As shown on figure 8.6 above, when a user marks some words as known and unknown, article recommendation and similarity measures need to be updated. When a user rates an article, the rating of this article changes for other users. The Rating Updater process performs this calculation.

## 8.2 De-Coupling Database Operations

Each process we presented above accesses the database. However, none of them has direct access to it. All accesses are done through a set of *Database Managers* (commonly known as DAO<sup>1</sup>) which keep the database **de-coupled** from the processes (see figure 8.7 below). This gives us the following advantages:

- We can change the underlying database (or better, the persistence layer) without modifying our processes.
- We can optimise database access by performing batch read and write operations more easily.

<sup>1</sup>Direct Access Object:<http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>

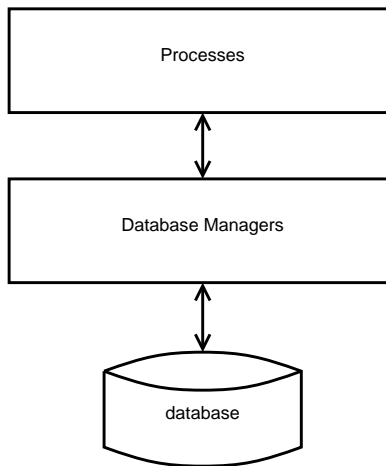


Figure 8.7: Database Managers between Processes and Database

Each Database Manager handles the operations (creation, query, modification, removal) of one kind of entities. Here is the list of entities we use:

- **User**: Holds user specific data like date of join, language preferences etc.
- **Feed**: Contains feed's URL, language, category and last time we updated it.
- **Page**: Holds an article's URL, HTML, text, category and language.
- **Word**: Information we hold about a word (frequency, language etc).
- **WordPages** : Reverse index. Contains the pages each word appears into.
- **UserWord** : A word that a user knows.
- **UserReadParagraph** : Holds information about which paragraphs have been read from each text.
- **UserPageMeasure** : Holds measures of how easy/hard is an article for a user.
- **Recommendation** : Holds user ratings (who, which page and what rating).

Each entity is represented by a Java Data Object (JDO)<sup>2</sup> and contains a primary key that uniquely identifies it. As we will see later on we use these keys for efficient retrieval.

---

<sup>2</sup>Java Data Objects : <http://java.sun.com/jdo/>

## 8.3 Running on the cloud

Readabix is hosted on Google AppEngine<sup>3</sup> which allows you to run your application on Google's infrastructure (cloud). Google AppEngine supports applications written in Java<sup>4</sup> or Python. We chose Java for static typing, easier refactoring and accurate code completion.

Running Readabix on Google's cloud has the following advantages:

- **Performance** - The application runs on highly optimised infrastructure (BigTable, GFS).
- **Easy to scale** - The increase on the number of users doesn't affect service performance. In other words, we can have as many users as we want.
- **Cost efficiency** - Price depends on resources usage. No monthly fee etc. A remarkable amount of resources is available for free<sup>5</sup>.
- **Multiple Versions** - We can run different versions of our website at the same time. This is very useful for trying out a new version before releasing it to the users.

It has also a number of disadvantages:

- **30s limit on each request** - This causes some trouble with time consuming operations. However, it can be addressed by using Task Queues (see section 8.3 for more details).
- **Lack of SQL** - You can't use the features of SQL for querying the database (aggregation, joins etc). You have to manually code the ones you need.
- **Database query overhead** - A database query on the cloud has a lot higher overhead than a local query, which makes it harder to estimate the actual performance during development. This can be addressed by batch read/write operations, caching and *Query Cursors*<sup>6</sup>.

### Datastore

*Datastore* is Google's storage service available for AppEngine applications. Datastore is a schema-less object database. There are no tables or schemas. Each object (known as *entity*) is stored serialised and is identified by a unique key. Datastore supports the following operations:

---

<sup>3</sup>Google AppEngine Website: <http://code.google.com/appengine/>

<sup>4</sup>It can host any language that runs on JVM (Java Virtual Machine)

<sup>5</sup>Google AppEngine Quotas: <http://code.google.com/appengine/docs/quotas.html>

<sup>6</sup>Query Cursors : [http://code.google.com/appengine/docs/java/datastore/queriesandindexes.html#Query\\_Cursors](http://code.google.com/appengine/docs/java/datastore/queriesandindexes.html#Query_Cursors)

- **Put** - Writes one or more (up to 500) objects concurrently in the database (max total size: 1 MB).
- **Get** - Gets one or more objects concurrently from the database (max total size 1 MB). Note that each object is retrieved with all its properties.
- **Delete** - Removes one or more (up to 500) objects concurrently from the database (max. total size: 1 MB).
- **Query** - Query a single entity by its properties. Each query can be served only if an index has been created for it.

The nature and limitations of Datastore affected our design in the following ways:

- **Design of the Word model** - Initially we wanted to keep all the information we know about a word in the same object. That would include the reverse index (list of texts it appears in), word frequency and other measures. With this approach, we would be limited on the amount of words we can retrieve in one Get operation (max 1MB). For example, if we wanted the frequency of 800 words, we would have to do it in many Get operations than in one. So, our final approach was to split the model into two.
- **Design of Incremental Counting** - When a user saves a paragraph, we need to update the texts that contain the affected words. Normally, we would iterate over the words, retrieve the texts each word appears in and modify them one by one. Instead, we use a single Get operation to retrieve all the affected texts at once, we update their them and we use a single Get operation to write the changes. This speeds up incredibly our operations and allows us to scale easier on the number of articles.
- **Design of the Reverse-Index** - Here we had two choices. Either to create an entity with *text* and *word* as properties and store multiple pairs for each word in each text or store a *list of texts* for each *word*. The first approach wouldn't allow us to get texts of more than one words at once and the second approach makes it slower to remove a text from the system. The second approach limits also the amount of texts a word can appear in (around  $1\text{ MB} / 8\text{ bytes} \approx 100\ 000$  texts). We implemented both approaches and we chose the second approach which leads to a lot faster saving of paragraphs.

## Task Queues

*Task Queues* is the service that Google AppEngine offers for running a computation in the background. When you want to perform an operation without forcing the user to wait for it's completion, you can put a task in the Task Queue and run it on the background. You can put more than one task at once

(max 100 at once) and they will be executed concurrently. In case a task fails to complete, it will be re-executed.

There is a limit of starting 50 tasks per second and each task can run up to 30 secs. That means that you can have up to  $50 * 30 = 1500$  active tasks at the same time. Note that all these executions don't affect the speed of the rest of the system. In other words, it doesn't slow down the website. This is because we are running it on the cloud and the tasks can be executed on different machines. With a single server web hosting we wouldn't be able to do so many concurrent computations without a major performance drop.

We use Task Queues in Readabix for all the processes. Retrieval and Processing processes are scheduled by cron jobs and Feedback processes by users. Table 8.1 below shows the task scheduling.

Table 8.1: Task Scheduling

Module	Process (Task)	Schedule
Retrieval	Feeds Updater	every 1 minute
Retrieval	Page Loader	every 5 minutes
Retrieval	Page Extractor	every 5 minutes
Processing	Reverse Index	every 10 minutes
Processing	Word Frequencies	every 10 minutes
Processing	Difficulty Evaluation	every 10 minutes
Feedback	Recommendation Updater	when user saves text/paragraph
Feedback	Similarity Updater	when user saves text/paragraph
Feedback	Rating Updater	when user rates a text
<i>none</i>	Front Page Updater	when user saves text/paragraph or rates a text

Table 8.1 contains the *Front Page Updater* process we didn't mention earlier. This process is responsible for loading the front page into cache, so it will load quickly when the user goes back to get more recommendations.

## 8.4 User Interface

User Interface was done with Google Web Toolkit (GWT)<sup>7</sup> which translates Java code into Javascript. GWT's main advantages are:

- The generated Javascript code is optimised and cross-browser compatible.
- Easy to debug your code. You can do it in Eclipse.
- Can easily call server side code using RPC calls (which are translated to AJAX calls in Javascript).
- Higher productivity than with Javascript

---

<sup>7</sup>Google Web Toolkit : <http://code.google.com/webtoolkit/>



- You can call native Javascript code from Java. This allows you to use any Javascript library available online.

GUI code can get quite complicated and hard to scale if it's not properly structured. In order to allow our application to scale-up on client side we used from the beginning the Model-View-Presenter (MVP) design pattern as suggested in "Large scale application development and MVP" [Ram10].

We split the User Interface into 5 main views: Global Menu, Language, Feeds, Pages and Embedded Words (see figure 8.8). Each of them can be modified, exchanged with a different view without any changes on other views. To lower coupling we used Event Bus for inter-views communication. For example, when language changes, Languages View sends a message to the Event Bus which in turn informs Feeds, Pages and Embedded Words that the event happened and they should switch to the new language.

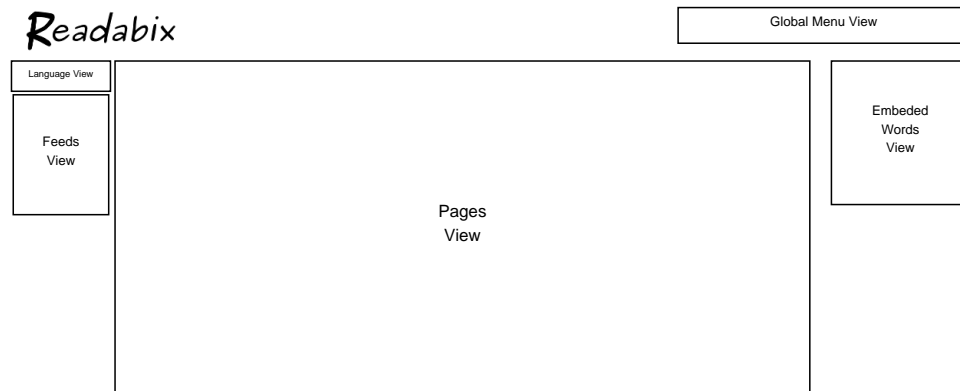


Figure 8.8: GUI Views

Administration Panel contains other 5 additional views for inspecting and modifying feeds, pages, words etc.

## 8.5 Code Base

The code base consists of 212 Java classes: 79 for client-side code, 23 for server-side servlets, 28 for database operation and 82 classes of application logic. Additionally there are 59 JUnit tests: 16 for database and 43 for application logic.

We used Dependency Injection<sup>8</sup> thoroughly and one Factory (Language Tools).

---

<sup>8</sup>Dependency Injection - [http://code.google.com/p/google-guice/wiki/Motivation?tm=6#Dependency\\_Injection](http://code.google.com/p/google-guice/wiki/Motivation?tm=6#Dependency_Injection)

The Language Tools factory is responsible for creating language dependent objects like Tokeniser, Word Manager, Known Words Evaluator etc. The Language Tools factory keeps references of the created objects and returns the same ones when requested. This way we keep garbage collection low when we process a batch of pages from many languages.

## Chapter 9

# Evaluation

We worked hard to design, implement, debug and optimise Readabix. We set a goal to create the best reading experience for language learning. Did we do that? Did we miss something? There is a way to find out: Evaluation.

In order to evaluate Readabix we asked from users to use it for two days. We friezed the articles on the system (no new articles added) and we added special evaluation instructions to appear when the site loads. We then tracked user's interaction and we got their opinions through an online survey.

In this chapter we present the objectives of the evaluation, the evaluation environment and the results we got from tracking the users and their opinions. We critically evaluate the results and we point out our mistakes and achievements. At the end, we do feature evaluation with other existing systems.

### 9.1 Objectives

In order to know what information to track and what questions to ask the users we had to put down a set of objectives. We placed the following objectives to find out

- if Readabix suggests easy articles (initially and after users' feedback).
- if users liked our page for reading articles (marking, translating, saving).
- which features users liked and which ones they didn't.
- if people find Readabix useful for language learning.
- if there is any issue with the user interface we didn't notice.
- which additional features would users like to use in Readabix.

## 9.2 Evaluation Environment

As we mentioned earlier we used a fixed set of articles for the evaluation. We offered articles in 4 languages: *Spanish*, *French*, *Polish* and *Greek*. We intentionally excluded *English* because our users were all very proficient in English and the evaluation environment didn't include the challenging mode. Unfortunately we didn't have enough articles in French due to HTML encoding issues. There was no time to address those issues because French was added on the last moment upon users' request. Users were informed about the issue before using the system. Table 9.1 below shows the number of articles in each language and the average size of the articles (in words).

Table 9.1: Languages used for evaluation

Language	Number of articles	Average number of words per text
Spanish	111	246.01
French	32	193.84
Polish	222	148.62
Greek	109	181.37

Our feeds source was Google News. We got texts from *Business*, *Technology*, *Entertainment*, *Sports* and *World* categories for all the languages. Additionally for each language, we added the country's local news: *Spain*, *France*, *Poland* and *Greece* (also taken from Google News).

Each user was starting with a *clean profile* (no words, no ratings). We used *Easy Mode* measure for evaluation (*Word Repetition* and *Weighted Mode* measures).

The features that were de-activated during evaluation are:

- Ability to interact with the lists of learned/known/unknown words.
- Recommend article to other users (the stars were used for rating difficulty of the article).
- Settings page wasn't accessible to users because we didn't want new articles in the system.
- Word recommendations

## 9.3 Tracking Users

We tracked user's interaction with the system by adding log entries to the database about:

- Which articles were displayed for each user.
- Which articles were opened by each user.

These log entries, would give us an accurate measure of how many **distinct** articles were displayed to each user.

The rest of the the data we collected them by querying the database after the end of the evaluation phase. The table 9.2 below shows the data we collected:

Table 9.2: Data collected at the end of evaluation phase

Measure	Value
Number of users	18
Number of displayed articles	2183
Number of opened articles	178
Number of saved articles (partial or full)	62
<b>Number of rated articles</b>	<b>12</b>
<b>Number of articles rated as easy</b> ( $rating \geq 3$ )	<b>8</b>
<b>Number of articles rated as hard</b> ( $rating \leq 3$ )	<b>4</b>
<b>Number of users that rated articles</b>	<b>6</b>
Average number of articles read per user	3.44
Average number of read (saved) paragraphs	6.84
Average number of paragraphs	11.42

By processing the data above we get the following percentages (displayed in table 9.3).

Table 9.3: Analysis of tracked data

Measure	Value
Opened articles out of displayed articles	8%
Saved articles out of opened articles	34%
<b>Rated articles out of saved articles</b>	<b>19%</b>
<b>Users that rated any article out of all users</b>	<b>30%</b>

We observe that the number of rated articles is very low (19%). This seems to be because only 30% of the users rated any article. By a closer look in the database we noticed that the users that rated any article have rated only 54% of the articles they read. The possible reasons are considered:

- Users **didn't want** to rate the articles they read.
- Users **forgot** to rate the articles they read.





The second is more probable to be the real reason. Users started reading the text, marking words, saving paragraphs and scrolling down to the next of the paragraphs. When they finished they clicked the "Save & Close" button, the article closed and they **forgot** to rate the article.

This discovery gives us a **good feedback** about the **mistake** we did with the User Interface. Users should be prompted to rate the article when they finished reading it. When the user would close the window/tab by either clicking the buttons on the bottom or the browser "x" button, a pop-up message should appear asking the user to rate the text before he closes it.

## 9.4 Survey



Users were instructed to fill up the survey at the end of the evaluation. We provided a link (for the online survey) in red colour at the top-right menu. Survey participation was 50% (9 users). Users were asked to answer 9 questions. We list the questions along with the responses we received. For each question we make some comments.

### Q1. Which languages did you use?

		Response Percent	Response Count
English		0.0%	0
French		77.8%	7
Spanish		44.4%	4
Polish		33.3%	3
Greek		22.2%	2
		<b>answered question</b>	<b>9</b>
		<b>skipped question</b>	<b>0</b>







English has been disabled for evaluation, so no users used it. The most popular language was French and second most popular was Spanish. Note that French had the smaller number of articles (due to encoding errors) in the system. That didn't affect users' response because only 2 out of 7 users used only French. The other 5 tried the system with Spanish, Polish or Greek.

### Q2. Did you like the page that displays the article? Marking and translating words, saving paragraphs etc.

		Response Percent	Response Count
Very much		66.7%	6
A bit		33.3%	3
Not much		0.0%	0
Not at all		0.0%	0
		<b>answered question</b>	<b>9</b>
		<b>skipped question</b>	<b>0</b>

The majority of the users liked a lot the interface for reading the article. The minority that liked it less, left comments on misformatted pages and more user friendly page (ability to change text size and show pictures from the original article). We consider the reading interface as successful with space for improvement.

### Q3. Which of the following features did you use?







		Response Percent	Response Count
Marking unknown words		88.9%	8
Saving paragraphs		77.8%	7
Translating words		100.0%	9
List of learned/known/unknown words (on the right side)		55.6%	5
Division of articles into categories		77.8%	7
Ability to keep track of the articles you have read		33.3%	3
		answered question	9
		skipped question	0

We notice that almost all the users used the marking, translating and saving features. We notice that 1 user responded that he didn't mark any unknown word. This is impossible, since to get word translations you must mark the words as unknown. It's highly probable then that the user didn't realise that he marked the word as unknown. This can be improved by showing a hint ("*click to translate and mark this word as unknown*") when the user puts the mouse over a word.

We notice that users didn't use much the lists of learned/known/unknown words. This might be because interaction between the user and the words was disabled for evaluation.

Users didn't use that much the ability to keep track of articles you read. They might have not noticed it, if they read only a few paragraphs.

## Q4. From the features you used, which ones did you like?

		Response Percent	Response Count
Marking unknown words		88.9%	8
Saving paragraphs		33.3%	3
Translating words		77.8%	7
List of learned/known/unknown words (on the right side)		33.3%	3
Division of articles into categories		66.7%	6
Ability to keep track of the articles you have read		44.4%	4
		<b>answered question</b>	<b>9</b>
		<b>skipped question</b>	<b>0</b>

The fact that most people liked the marking of unknown words, proves that they liked giving feedback about their knowledge. We were concerned if users would like clicking/interacting with the text, since they are not used to or it could be too distracting while reading.


Translations were also appreciated. We believe we missed that one vote from inaccurate translations.

We noticed by this answer that not many users liked the "saving paragraphs" feature. The reasons need to be investigated and actions must be taken for improvement. The same is with the list of words (on the right). Few users liked them. We must consider either improving them (making them more useful) or removing them completely. We need to investigate before we do that because the list of words wasn't updating automatically and users had to refresh the page. This might have been the reason of so low votes on it.

Ability of keeping track of articles you have read seems to have been liked by the people that used it. This indicates that it's desired but not noticed enough.



**Q5. Which feature(s) didn't you like?**




	Response Percent	Response Count
Marking unknown words	0.0%	0
Saving paragraphs 	100.0%	1
Translating words	0.0%	0
List of learned/known/unknown words (on the right side)	0.0%	0
Division of articles into categories	0.0%	0
Ability to keep track of the articles you have read	0.0%	0
<b>answered question</b>		<b>1</b>
<b>skipped question</b>		<b>8</b>

One person didn't like saving paragraphs. Possible reasons:

- The saving paragraphs process was inactive and took too long to load and mark the words
- Might have been a server error
- Too much clicking



As we mentioned earlier, the saving paragraphs needs improvement since users are not satisfied with it.

**Q6. When you used Readabix for the first time, how easy were the articles?**

	Response Percent	Response Count
Very Easy	0.0%	0
Easy 	33.3%	3
Average Difficulty 	55.6%	5
Quite Hard 	11.1%	1
Hard	0.0%	0
<b>answered question</b>		<b>9</b>
<b>skipped question</b>		<b>0</b>

The system doesn't know anything about the user at the beginning and recommends articles based on Word Repetition. This response indicates that Word Repetition might not be enough. Different measure can be considered in the future but its not urgent. It would be urgent if the texts were hard!


**Q7. After using the system for a while (marking words and saving them), did you get any articles that were easy to read?**

		Response Percent	Response Count
<b>Yes</b>		66.7%	6
<b>No</b>		0.0%	0
<b>Didn't notice any difference</b>		33.3%	3
		<b>answered question</b>	<b>9</b>
		<b>skipped question</b>	<b>0</b>

This is the most surprising response. Having in mind that each user read on average 3.44 articles, we were not expecting them to notice any difference. Surprising also was that 1 of the 2 users that used only French reported that he/she got easier articles.







This answer proves (even though it needs to be confirmed with more users) that the User-Dependent measures we used work well and need small amount of feedback to start working.

**Q8. Do you find Readabix useful for improving your language skills?**

		Response Percent	Response Count
<b>Very useful</b>		100.0%	9
<b>A bit useful</b>		0.0%	0
<b>Not useful</b>		0.0%	0
		<b>answered question</b>	<b>9</b>
		<b>skipped question</b>	<b>0</b>

All users found Readabix very useful. This surprised us, since we were expecting some votes for "a bit useful" but we got none.

### Q9. Which of the following features would you like to be included in Readabix?

		Response Percent	Response Count
Vocabulary exercises with your unknown words		77.8%	7
Suggestions of new words to learn		33.3%	3
Listen to music and read lyrics		88.9%	8
Read books		55.6%	5
Ability to upload your own texts		33.3%	3
Ability to share reading material with other users		44.4%	4
		answered question	9
		skipped question	0

We have many things in mind to add to Readabix. But which one would be the most wanted? This response gives us a hint that we should go towards the lyrics feature and vocabulary exercises. Lyrics can be the next feature, since its the easiest to support with the current system structure.

This response also tells us that the Word Recommendations feature we implemented (but disabled for evaluation) wasn't really needed. Wish we had asked earlier. It would save us some time. Uploading and sharing texts also doesn't seem to be very popular either. It will be postponed then for the time being.

## 9.5 Existing Tools

We looked for tools that recommend you articles to read in foreign languages. We found websites with users sharing texts and labelling manually the text difficulty. We looked for tools that help you to read in a foreign language and we found translators and browser plugins for translation. We looked for a service that can keep track of your known and unknown words and we found vocabulary trainers and language learning programs.

We strongly believe that Readabix is unique in its kind, at least for English users learning other languages. There might be a system like Readabix entirely written in a foreign language for learning English (or German) and we couldn't find it.

Since we didn't find any similar service with Readabix, we will compare it with popular language learning software (Babbel), with news aggregators (Google News) and with vocabulary trainers (Parley-KDE4).

We do a feature comparison, showing which features of Readabix are supported by which tools and which features is Readabix missing. Table 9.4 shows this comparison.

Table 9.4: Feature Comparison Table with Existing Tools

Feature	Readabix	Babbel	Google News	Parley-KDE4
Online Service	✓	✓	✓	-
News aggregation	✓	-	✓	-
Interactive reading environment	✓	-	-	-
Keeping track of known and unknown words	✓	✓	-	✓
Dynamic vocabulary	✓	-	-	✓
Vocabulary exercises	-	✓	-	✓
Supports many languages	✓	-	✓	✓
Keeping track of learning progress	✓	✓	-	✓
Provides word translation	✓	✓	-	✓
Adjusts to the user's level	✓	-	-	✓
Suggests words	✓	✓	-	-
Supports images and audio	-	✓	✓	✓
Social interaction	✓	✓	✓	-

By comparing to these systems we can see Readabix's potential. The features Readabix has in common with them might not be as advanced as the theirs' (yet) but there are certainly features that give Readabix a big advantage.

Comparing Readabix with Babbel, Readabix has the ability to meet up user's level and recommend him something challenging. Babbel has a predefined set of lessons and leaves it up to the user to go through the lessons and find out which one is the right one. And that doesn't guarantee that all the previous lessons will contain vocabulary that the user knows. Babbel has nice content with images, videos and text but lacks the dynamic of Readabix to adjust to the user.

Comparison of Readabix with Google News can be seen only for the article reading experience. Google indexes thousands of articles on the web and displays them in groups to the users. Readabix's capability is much much lower than that. We can't compare to Google News for aggregation.

Comparing Readabix with Parley-KDE4 is interesting. Parley helps users improve their vocabulary but they need to type in manually all the words they want to exercise. This makes it hard for users to keep on using it. Readabix, can get to Parley's level by providing similar vocabulary exercises. But Readabix's approach will be different. Vocabulary exercises will be created automatically for each user, which we believe gives a very strong advantage over Parley-KDE4.

# Conclusions

Evaluation gave us a big motivation to continue working on Readabix. Our aim, to prove that Readabix is a useful language tool that can help users improve their language skills has been achieved. Finding easy articles for users works and requires small amount of feedback. However, further evaluation is needed with more users, more articles and more languages in order to be completely confident. Some mistakes we did with the user interface need to also be addressed and focus on creating a more user-friendly website.

Our choice to run Readabix on the cloud turned out to be a good one. The performance advantage it gives and the ease of scaling up is incomparable to any other conventional hosting service. It's also very cost efficient and gives us the benefit of having Readabix available all the time. However we need to address some copyright issues with the news articles before we open Readabix to the public. This will probably require us to develop a browser plugin for reading the articles.

Readabix has many options for expansion. First would be the songs and lyrics which would allow the practise of listening and reading skills of a user with songs that are easy to understand. Second would be automatically created vocabulary exercises (missing words, word recall etc) that would challenge every user. Books would be another option, through project Gutenberg which offers thousands of free ebooks. I'm sure iPad users would love it.

Working on Readabix has been a great personal pleasure, challenge and reward. There were moments of desperation with many technical difficulties and uncertainties but we got over them. It needed a lot of effort to do that but we knew from the beginning that *"τ' αγαθα κοποις κτωνται"* (*good things are gained with effort*).

# Bibliography

- [Bro00] H. Douglas Brown. *Principles of Language Learning and Teaching*. Pearson ESL, 10 Bank Street, White Plains, NY 10606, 4th edition, March 2000.
- [CDM08] Hinrich Schutze Christopher D. Manning, Prabhakar Raghavan. *Introduction to Information Retrieval*, pages 61–77. Cambridge University Press, New York, NY, 2008.
- [Che71] Arnold B. Cheyney. *Teaching Reading Skills through the Newspaper*, page vi. Reading Aids Series. International Reading Association, 6 Tyre Avenue, Newark, Del. 19711, 1971.
- [DDGR07] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 271–280, New York, NY, USA, 2007. ACM.
- [DE05] Papagelis M. Rousidis I. Plexousakis D. and Theoharopoulos E. Incremental collaborative filtering for highly-scalable recommendation algorithms. In *Foundations of Intelligent Systems*, volume 3488/2005 of *Lecture Notes in Computer Science*, pages 553–561, Berlin / Heidelberg, 2005. Springer.
- [DuB04] W.H. DuBay. The principles of readability. *Impact Information*, pages 1–76, 2004.
- [EDH03] Jr. E. D. Hirsch. Reading comprehension requires knowledge of words and the world. *American Educator*, 27(1):10–13,16–22,28–29,48, 2003. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.115.721&rep=rep1&type=pdf>.
- [FKS01] A. Finn, N. Kushmerick, and B. Smyth. Fact or fiction: Content classification for digital libraries. In *Joint DELOS-NSF Workshop on Personalisation and Recommender Systems in Digital Libraries (Dublin)*, 2001.

## BIBLIOGRAPHY

---

- [Got08] Thomas Gottron. Content code blurring: A new approach to content extraction. In *DEXA '08: Proceedings of the 2008 19th International Conference on Database and Expert Systems Application*, pages 29–33, Washington, DC, USA, 2008. IEEE Computer Society.
- [GS97] William Grabe and Fredricka L. Stoller. Reading and vocabulary development in a second language. In *Second language vocabulary acquisition: a rationale for pedagogy*, The Cambridge applied linguistics series, chapter 6, pages 98–122. Cambridge University Press, Cambridge, UK, 1997.
- [Kin75] JP Kincaid. Derivation of New Readability Formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy Enlisted Personnel. 1975.
- [Lee09] Sang-Ki Lee. Topic congruence and topic interest: How do they affect second language reading comprehension? *Reading in a Foreign Language*, 21(2):159–178, October 2009. <http://www.nflrc.hawaii.edu/rfl/October2009/articles/lee.pdf>.
- [MDA] Louwrese M.M. McNamara D.S. and Graesser A.C. Coh-metrix: Automated cohesion and coherence scores to predict text readability and facilitate comprehension. <http://cohmetrix.memphis.edu/cohmetrixpr/publications.html>.
- [MS02] C.D. Manning and H. Schütze. *Foundations of statistical natural language processing*. MIT Press, 2002.
- [PN08] Emily Pitler and Ani Nenkova. Revisiting readability: a unified framework for predicting text quality. In *EMNLP '08: Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 186–195, Morristown, NJ, USA, 2008. Association for Computational Linguistics.
- [PP08] Jyotika Prasad and Andreas Paepcke. Coreex: content extraction from online news articles. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 1391–1392, New York, NY, USA, 2008. ACM.
- [Ram10] Chris Ramsdale. Large scale application development and mvp. <http://code.google.com/webtoolkit/articles/mvp-architecture.html>, 2010.
- [SC01] L. Si and J. Callan. A statistical model for scientific readability. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 574–576. ACM New York, NY, USA, 2001.
- [Seg07] Toby Segaran. *Programming Collective Intelligence*, page 8. O'Reilly Media Inc., Sebastopol, CA, 2007.

## BIBLIOGRAPHY

---

- [Wak03] Richard Wakely. Good practice in teaching and learning vocabulary. In *Good Practice Guide*. Subject Centre for Languages, Linguistics and Area Studies, Southampton, UK, February 2003. <http://www.llas.ac.uk/resources/gpg/1421>.
- [Wal03] Catherine Walter. Reading in a second language. In *Good Practice Guide*. Subject Centre for Languages, Linguistics and Area Studies, Southampton, UK, January 2003. <http://www.llas.ac.uk/resources/gpg/1420>.