QuickDefine: generating concise definitions of computer science concepts for novice autodidact programmers.

Jack Ragless
School of Computer Science
The University of Adelaide
SA 5005 Australia
jack.ragless@student.adelaide.edu.au

Dr Christoph Treude
School of Computer Science
The University of Adelaide
SA 5005 Australia
christoph.treude@adelaide.edu.au

Dr Sebastian Baltes
School of Computer Science
The University of Adelaide
SA 5005 Australia
sebastian.baltes@adelaide.edu.au

1 ABSTRACT

This paper describes QuickDefine, a tool which aims to automatically generate concise and accessible definitions of Computer Science concepts for novice programmers. It is a response to a Stack Overflow study which found users frequently embedded Wikipedia links in lieu of formal definitions, despite its numerous pedagogical deficiencies.

QuickDefine works by scraping the top results for a given concept keyword from Google Search. It then selects preferred websites based on predefined heuristics. Lastly, potential definition sentences are found via extractive summarisation based on models including TextRank and constituency parsing.

In designing this software, valuable data was generated regarding the syntactic structure of definition sentences. Of the methods tested a two-level constituency parsing tree was ultimately the most effective at retrieving the definition of a given keyword.

2 GITHUB REPOSITORY

https://github.com/jackragless/QUICKDEFINE

3 INTRODUCTION

Due to manifold factors programming is a difficult subject to teach through traditional means. Students face a lack of personal instruction, errors specific to their code implementation and constantly updated languages / libraries. Hence autodidactic learning is a necessity among computer science (CS) majors [2]. Vincent et al previously examined this phenomenon in the context of Stack Overflow answers, where users provided Wikipedia links to related concepts in lieu of formal definitions [5]. Treude, Robillard then studied the pedagogical quality of these Wikipedia articles. From survey data they concluded Wikipedia was ill-suited to autodidactic learning due to esoteric terminology, lack of examples, irrelevant peripheral information and unfamiliar notation [1].

QuickDefine is intended to overcome these shortcomings via a two-stage application of Natural Language Processing. In Stage One, top websites are scraped from Google Search results and eliminated according to a list of predefined heuristics. Then in Stage Two, extractive summarisation is applied to the remaining sites to form a candidate pool of sentences. From this pool the optimal definition is determined.

The initial motivation for QuickDefine was to create a central source which provides accessible definitions for novice programmers. Ideally this could be integrated into sites like Stack Overflow to generate output when Wikipedia links are provided.
In researching and designing QuickDefine, however, factors which determine the pedagogical quality of a learning resource have been explored. As has syntactic structure of definition sentences.

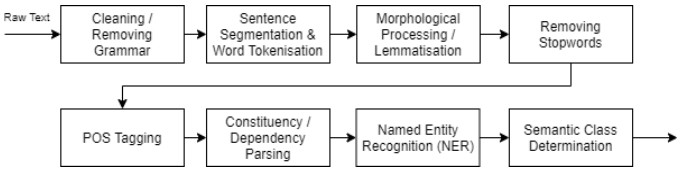
4 LITERATURE REVIEW

4.1. Opportunistic Programming:

A term frequently used by Brandt et al, referring to a programming process where speed and ease of development take precedent over functionality and maintainability [4]. Another Brandt study describes subtypes of opportunistic programming: learning, clarification and reminding [3]. QuickDefine is intended as a clarification tool for Stack Overflow users, as it should explain high-level concepts behind a block of code. It could also be used to remind users of concepts, but this is typically reserved for syntax and function concepts. Its focus on concision prevents it from being a full learning resource.

4.2. Natural Language Processing (NLP):

In abstract, Natural Language Processing refers to systems which attempt to understand human language and manipulate it whist retaining meaning [6]. It is difficult to provide a universal technical explanation of the processes which constitute NLP, as its implementation is highly dependent upon its application. A common pipeline includes the following modules:



4.2.1. Cleaning raw text: removing grammar, converting characters to lowercase, et cetera.

4.2.2. *Sentence segmentation and tokenisation*: splitting text --> sentences --> words. [9]

4.2.3. *Morphological processing*; lemmatization; stemming: similar processes where words are simplified to their base form (lemma) whilst maintaining meaning.
eg. running, ran, runs ---> run [8]

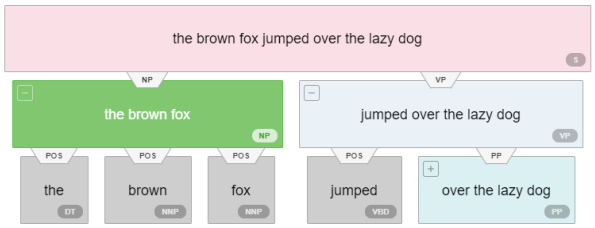
4.2.4. *Removing stopwords*: eliminating frequently occurring terms which tend to have little sematic value. Existing SW lexicons like NLTK's are frequently used for this process. [17]

4.2.5. *Part-of-speech (POS) tagging*: indicates syntactic role of each word in sentence; noun, adverb, adjective etc.

4.2.6. *Constituency parsing*: chunks POS tags into a tree structure. A vital intermediary stage in understanding a sentence's semantics.

Example [19]:
"the brown fox" == noun phrase (NP);

"jumped over the lazy dog" == verb phrase (VP).



Alternatively, *dependency parsing* represents a sentence as a graph structure, where each word is a node and the edges describe the dependencies between words. [18].

4.2.7 *Named entity recognition (NER):* classifying nouns as person names, organisations, locations etc.

4.2.8. *Semantic role labelling (SRL):* "lexical semantics" is the most rudimentary form of SRL, concerned with determining the hypernym of each word (hyponym). For instance, the word colour is the hypernym of blue, yellow, red etc. More sophisticated sematic analysis considers the semantics of phrases within a sentence. [7] This technique is beyond the scope of the QuickDefine project.

4.2.9. *Coreference Resolution (CR):* an extension of the NLP pipeline. CR is the process of finding two or more expressions which refer to the same real-life entity. [9]

Example:

(1) Donald Trump built the wall.

(2) It is the defining achievemenst of his presidency.

Anaphor:		Antecadent:
It	-->	"the wall"
His	-->	"Donald Trump('s)"

4.3. Sentiment analysis:

There is growing literature regarding the application of sentiment analysis to the software engineering domain. It is apparent that out-of-the-box models are insufficient within the SE domain as they were originally trained on social media datasets. [12] Hence, incorrect interpretation of domain-specific terminology substantially degrades their accuracy. [13] Lin et al attempted to retrain existing models with a Stack Overflow dataset, including the *Sentistrength* lexical scoring algorithm and *Standford CoreNLP*, a recursive neural network (RNN) model. No model tested could accurately determine positive/negative polarity from the SO dataset. All models, however, were effective at predicting neutral sentiment. The range of the neutral precision among all models was 0.815 for the general NLTK model to 0.884 for the RNN model. [11]

4.4. TextRank:

TextRank is an algorithm which can perform extractive summarisation by determining the similarity (ie. common tokens) of a sentence to all others in a given text. The higher the similarity score, the greater its importance.
It is a graph-based ranking algorithm designed on Google's PageRank [15]. A sentence which shares words with another is essentially "recommending" its counterpart. Recommendations are represented as edges between

vertices (words). The number of recommendations a vertex receives determines its weighting, which then affects the value of its own recommendations. Hence TextRank's weighting computations must be continually iterated until they stabilise below a given threshold. The formal definition for TextRank is provided below:

$$S(V_i) = (1-d) + d * \sum_{j \in In(V_i)} \frac{1}{|Out(V_j)|} S(V_j)$$

Out(Vi): the no of vertices vertex Vi points to.
In(Vi): the no of vertices pointing to Vi.
D: a damping factor, usually set to 0.85 after Brin and Page

In their paper Mihalcea et al compared its summartive performance and found it ranked among the top five systems as of DUC 2002. [14] Since publication, others have proposed models with additional dimensions. Ferreira et al modelled four dimensions (similarity, semantic similarity, coreference, discourse information) yielding improved performance [16].

5 METHODOLOGY & EXPERIMENTAL SETUP

5.1. Developing website scraper:

The intent of the website scraper is to retrieve the textual content of the top results from a preexisting search engines for a given keyword. For superior runtime and lower probability of connection errors, APIs from platforms like DuckDuckGo or Google were initially considered. Unfortunately, however, these services require payment above a certain query threshold and often omit search results that would otherwise appear in the browser version [20].

Hence, the Python Requests library was used to perform HTTP requests. Python Requests does not natively allow simultaneous requests, but was selected for its concision and community support. [21] BeautifulSoup4 was selected to parse and filter the retrieved HTML content, again for its popularity.

The scraping methodology consists of two steps. First, a Google Search (GS) query is made for a given concept keyphrase, with a specified number of results. From the HTML of this GS page the result URLs are extracted. Sub-requests are then made to scrape the HTML from each of these websites.

Lastly, a page's "raw text" is extracted from its <p> or <li> tags. Note these tags were selected through a trial and error process. Other tags such as <pre> could be added.

All of this data is combined into a website object and stored in a nested array:

```
site_obj =
{
'keyword' : 'undefined behaviour',
'link' :
'https://en.cppreference.com/w/cpp/language/ub',
'soup' : '<!doctype html> <html lang="en"><head>...',
'raw_text' : 'Undefined Behaviour defined any
problem…'
}
             Keyword 1 array                kw2 arr
               ^                             ^
nest_arr = [ [{site_obj_1}, site_obj_2 …],  […] ]
```

To approximate the efficacy of the scraper I scraped 800 websites over the University of Adelaide WIFI network and measured the connection error frequency, error type and request speed.

5.2. Synthesising heuristics & detection methods:

This stage was a direct extension of Treude, Robillard's paper into the unsuitability of Wikipedia as a learning resource for opportunistic learning. This paper attributed Wikipedia's failure to a number of key heuristics that are expanded upon here. [1] Treude's paper also provided a dataset of ~40,000 computer science concept keywords mined from Stack Overflow (SO) forums in order of prevalence.

The experimental process was a rather subjective one. Five concepts from the SO dataset were searched in Google verbatim and for each, the first five search results were ranked by the perceived intuitiveness of their textual content.

Once the pages were ordered, the heuristics influencing the results were considered. It was noted that a website's aesthetic design tended to bias its rank.

Methods of measuring / detecting these heuristics were then synthesised. Existing Python libraries and custom functions were used; see *Results*.

5.3. Implementing heuristic elimination:

To combine heuristics, a score-based system was developed. Each website starts with a perfect score == 1, which is then decreased by a series of weighted heuristic factors:

$$siteScore = 1 - (wd_1 + \cdots wd_n) - (wf_1 + \cdots wf_n)$$

Scores are assigned a weight w according to perceived importance and validity.
wf_i (fixed) heuristic factors are those applied when a simple Boolean condition is met. For example, if a website object's URL contains a certain substring it may trigger a fixed heuristic.
wd_i heuristic factors are measured. For instance, the sentiment score of a site object. To avoid applying arbitrary heuristic thresholds, these values are compared to a normal distribution of other websites within the same keyword. If the heuristic exceeds +/- one standard deviation from the mean, $d = 1$. If it exceeds two standard deviations $d = 2$.
The site objects of a keyword are ordered by score and the top 50% proceed to the extractive summarisation phase. Currently sixteen websites are initially scraped per keyword, with eight proceeding to summarisation. The website number and elimination proportion, however, have simply been set to obtain a reasonable number of definitions at the end of the QuickDefine pipeline. Outside this they are unjustified and therefore subject to change.

5.4. Syntactic structure of definition sentences:
Given the time constraints of this project, a purely syntactic approach to extractive summarisation was utilised. To achieve this, the structure of definition sentences needed to be explored. Existing literature mentioned basic characteristics, such as starting with a noun and containing the verb "be" [22]. More in-depth information could not be found, however, so this syntactic data needed to be self-generated.
A PDF of Oxford's *A Dictionary of Computer Science* [23] served as the corpus for this experiment.
First, a keyword array was extracted from the dictionary's glossary, as were the definition sentences from its body. These were cleaned to remove unwanted grammar, uppercase characters, as well as two-part definitions or abbreviations of others within the book.
One major issue remained with this corpus. The definitions had the following structure:

CONCATENATION: The operation of joining two strings to form a longer string.

This posed an issue as the keyword itself was not part of the sentence. To solve this, only sentences beginning with the Determiner (DET) POS Tag were used. This allowed "KEYWORD"(NOUN) + "IS"(VERB) to be prepended to these sentences. After filtering, 3200 valid definitions remained.

These definitions were assigned POS tags using the Natural Language Toolkit (NLTK) library and the highest frequency sequences were retained.
The dictionary also underwent a more sophisticated constituency parsing algorithm from the AllenNLP library. Levels one and two of highest constituency trees were kept. Levels beneath this were excluded as their high level of specificity would filter out all potential candidate sentences.

5.5. Extractive summarisation
Extractive sumnmarisation first required the raw text within the remaining site objects to be thoroughly cleaned:

- Convert all characters to lowercase
- Remove all content in brackets
- Remove punctuation symbols
- OPTIONAL: lemmatisation and stopword removal
- Remove foreign characters

The paragraphs were then segmented into sentences and those not containing their respective keyword were filtered out.
These so-called "candidate sentences" were then used to test various extraction models, singular and combined. These model permutations included:

Model	**Python library:**
Random	*Random*
1-level constituency parse matching	*AllenNLP*
Heuristics + 1-level constituency parse matching	*AllenNLP*
2-level constituency parse matching	*AllenNLP*
Heuristics + 2-level constituency parse matching	*AllenNLP*
TextRank	*Gensim*
Heuristics + TextRank	*Gensim*

From these models a "definition candidate pool" was formed, consisting of remaining sentences from all sites for a given keyword that passed the filter. The mean candidate pool success rate (CPSR) for each model was determined for 20 keywords (each containing 16 site objects). CPSR was determined manually by looking through the sentences

in all pools and determining the proportion of viable definitions in each. Note, definitions sentences need not meet a qualitative measure, but MUST be intended to act as a definition. Definitions do not include sentences where one can infer the meaning of the concept.

'Random' was used as a baseline to outperform. It combines all sentences from all scraped websites for a given keyword in one array and selects 10 at random to form its definition candidate pool.
Constituency parsing compares the tree structure of each candidate sentence to those obtained from the dictionary in 5.4.
The TextRank model uses a modified PageRank algorithm (as described in *Literature Review*) at a 0.1 ratio to extract 10% of the most "important" sentences.

6 RESULTS:

6.1. Website scraper:
There is little to analyse regarding the webscraper besides its reliability and performance. When tested on a sample size of 800 websites, it only failed 0.5% of requests. Note these failures do not affect the final number of results per keyword, as the scraper is designed to mine additional sites when failure occurs to reach its target number. (target == 16 in this instance).

Error Type	Frequency	Pct (of 800 requests)
ConnectionError	3	0.37%
TimoutError	1	0.13%
ToomanyredirectError	0	0.00%
Total errors	4	0.50%

Runtime is a more dubious metric, as it is highly dependent upon network speed. These following results were obtained over UofA WIFI.

Process	Avg Runtime (s)
Google Search Query	2.41
Single Page Query	1.72
Avg per kw (16 requests)	29.95

The focus of QuickDefine was not performance in its first iteration. If later versions are created, however, libraries like Octopus capable of multithreaded requests should significantly decrease runtime.

6.2. Heuristics & detection methods
The following website elimination heuristics were synthesized, as were methods for measuring/detecting them:

Negative heuristic:	**Solution:**
Long avg sentence length **Poor grammar** **Verbose**	Flesch Reading Ease Score (Textstat lib)
Explanation: rates sentences 0 (very difficult) --> 100 (very easy) based on function: $FRES = 206.835 - 1.105(\frac{total\ word}{total\ sent} - 84.6(\frac{total\ syllab}{total\ word})$	
Esoteric terminology	Custom function
Explanation: finds probability of each word in a site's raw text being a keyword from Treude's SO dataset. Function: $ESO = \frac{total\ kw\ instances}{total\ words}$	
Opinionated language	Subjectivity score (TextBlob lib)
Explanation: calculates score 0 (objective) ---> 1 (subjective) for a given string. This score is an average of the string's constituent words when assigned a weighting from TextBlob's lexicon.	
Commercial	Custom function
Explanation: commercial heuristic detected if websites header contains the key phrase 'Solutions' or 'Products'	
Academic	Custom function
Explanation: eliminates site entirely (ie. score = 0) if '.edu' or '.pdf' detected in site object's URL. Also manually removes other unwanted sources like Stack Overflow.	
No examples / pseudocode	Not addressed
Unfamiliar notation	Not addressed
Historical (and other peripheral) information)	Not addressed

The inclusion of the Flesch Reading Ease Score was initially motivated by small sample tests conducted personally, as well as use by many existing institutions such as the US Department of Defence [25]. Subsequent research has since revealed its inaccuracy [24] and it should be substituted or removed altogether in future iterations.

Using an out-of-the-box sentiment analysis package like Textblob is justified by findings discussed in the *Literature Review* that domain specific models consistently failed to outperform SE ones [11].
Academic sources, primarily university sites, were omitted as they tend to explain concepts in their entirety (including mathematical proofs etc), rather than simplifying them for novice programmers.
A commercial filter was deemed necessary to safeguard the trustworthiness of definitions. This said, anecdotal evidence showed almost all business sites provided accurate explanations of concepts (such as SQL Injection [26]). Note, all concepts can be weighted according to their perceived validity.

6.3. Syntactic structure of definition sentences:
The POS structure of *The Dictionary of Computer Science* followed a rigid 'NOUN VERB DET' structure due to the prepending process previously described.
If applied, these sequences would only detect definition sentences with similar syntax. Furthermore, this model would rely on matching identifier and target sentences of the same word count if applied. One cannot match a 5 POS sequence to the beginning of a 20 length POS sequence and expect accuracy.

Freq	5 word POS	6 word POS	7 word POS	8 word POS	9 word POS
1st	NOUN VERB DET NOUN ADP	NOUN VERB DET NOUN ADP NOUN	NOUN VERB DET NOUN ADP DET NOUN	NOUN VERB DET NOUN ADP DET NOUN ADP	NOUN VERB DET NOUN ADP DET NOUN ADP DET
2nd	NOUN VERB DET ADJ NOUN	NOUN VERB DET NOUN ADP DET	NOUN VERB DET ADJ NOUN ADP DET	NOUN VERB DET NOUN ADP DET ADJ NOUN	NOUN VERB DET NOUN ADP DET ADJ NOUN ADP
3rd	NOUN VERB DET NOUN NOUN	NOUN VERB DET ADJ NOUN ADP	NOUN VERB DET NOUN ADP ADJ NOUN	NOUN VERB DET NOUN ADP NOUN ADP DET	NOUN VERB DET NOUN ADP VERB DET NOUN ADP
4th	NOUN VERB DET NOUN DET	NOUN VERB DET NOUN DET VERB	NOUN VERB DET NOUN ADP NOUN NOUN	NOUN VERB DET ADJ NOUN ADP DET NOUN	NOUN VERB DET NOUN ADP DET NOUN DET VERB

Constituency parse matching alleviates this issue. Here we see the highest frequency first level constituency trees from our corpus. The most common second level sequences for each have also been provided. The 'NP VP' is overwhelmingly the most common definition sentence level-1 constituency.

First level Sequence	Freq	Pct (of 3200)	2nd lvl	1st	2nd	3rd
NP VP	2902	90.69%	--->	NNP VBZ NP (477 || 16%)	NN NN VBZ NP (374 || 13%)	JJ NN VBZ NP (374 || 13%)
NP VBZ NP	99	3.09%	--->	NN NN ? NP PP (8 || 8%)	NNP ? NP SBAR (5 || 5%)	NN NN ? NP SBAR (4 || 4%)
S VP	35	1.09%	--->	VP VBZ NP (33 || 94%)	VP VP NP (1 || 3%)	JJ NN VBZ NP (1 || 3%)
NP HYPH NP VP	33	1.03%	--->	NN ? NN NN VBZ NP (8 || 24%)	NN ? JJ NN VBZ NP (4 || 12%)	NN ? NN VBZ NP (3 || 9%)
LST HYPH NP VP	6	0.19%				
S HYPH NP VP	6	0.19%				
VP VP	6	0.19%				
VP VBZ NP	5	0.16%				
ADJP VP	5	0.16%				
S CC S	5	0.16%				

6.4. Extractive summarisation:
All models developed significantly outperformed 'Random', indicating some form of efficacy.

Model	Avg Candidate Pool Size	Candidate Pool Success Rate (CPSR)
random	10.00	4.44%
1lvl_const_parse	18.89	24.65%
1lvl_const_parse + heuristics	11.33	24.53%
2lvl_const_parse	2.44	38.19%
(alt)	-	52.88%
2lvl_const_parse + heuristics	1.72	29.63%
(alt)	-	53.33%
textrank	9.44	17.45%
textrank + heuristics	5.50	20.10%

By far the best performing singular model was the 2-level constituency parse matcher. The non-alt version includes results where the model returned an empty candidate pool (hence CPSR for that pool equals zero). The alt version excludes these results. This highlights the 2-level

constituency parser's limitation; its tendency to over-filter candidates to nil.
The second-best performer was the 1-level constituency model, followed by TextRank at third.
The heuristic filter yielded mixed results when combined with these models. It improved the results of some (ie. Textrank) and was detrimental to others (ie. lvl2_const_parse).
Note, this data has limited accuracy due to its small sample size (n = 20 keywords) and inconsistent candidate pool sizes. One could argue a 50% CPSR from a pool size == 20 is far more significant than a 50% CPSR for a pool size == 2.

7 CONCLUSION

Of all the systems developed for QuickDefine, the two-level constituency parsing shows the most promise. Though it sometimes eliminates all candidate definitions, it could be extended to apply other models (eg. one-level constituency parsing, TextRank, other unexplored models) when this occurs. Combining models together is likely the key to producing consistent, reasonable candidate pool sizes, and increasing the success rate of definition identification.
Conversely, there is no evidence to suggest the heuristic filters produced a higher success rate in extracting definitions. This is expected, however, considering this system filters based on quality of a website's textual data, not its probability of containing definitions. In this regard, heuristics may still prove useful when QuickDefine's definitions begin to be qualitatively assessed.
Outside of these models, QuickDefine yielded valuable definition sentence syntactic data.

8 IMPROVING & EXTENDING QUICKDEFINE:

Moving forward, there are several enhancements QuickDefine requires. Performance could be improved during scraping using a multithread-enabled library like Octopus and constituency parsing could apply a rule-based system rather than relying on AllenNLP.

For improved accuracy combined models need to be formulated and tested, starting with TextRank + constituency parsing. New models, including semantic analysis, should then be considered. If the extracted sentences are sufficiently accurate they could even form a training dataset for an unsupervised machine learning approach if needed.

Once extractive summarisation generates just one viable definition sentence per keyword, a blind qualitative survey could be conducted. If respondents consistently rank QuickDefine's definitions above Wikipedia's, the system would be considered effective.

At this point a user interface could be designed to return a QuickDefine definition for a given keyword string.

9 REFERENCES

[1] Martin P. Robillard and Christoph Treude. 2020. Understanding Wikipedia as a Resource for Opportunistic Learning of Computing Concepts. In The 51st ACM Technical Symposium on Computer Science Education (SIGCSE'20), March 11–14, 2020, Portland, OR, USA. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3328778.3366832

[2] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A Study of the Difficulties of Novice Programmers. In Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education. 14–18.

[3] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. 2009. Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. 1589–1598.

[4] Joel Brandt, Philip J. Guo, Joel Lewenstein, and Scott R. Klemmer. 2008. Opportunistic programming: how rapid ideation and prototyping occur in practice. In Proceedings of the 4th international workshop on End-user software engineering (WEUSE '08). Association for Computing Machinery, New York, NY, USA, 1–5.

[5] Nicholas Vincent, Isaac Johnson, and Brent Hecht. 2018. Examining Wikipedia With a Broader Lens: Quantifying the Value of Wikipedia's Relationships with Other Large-Scale Online Communities. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. 566:1–566:13.

[6] James F. Allen. 2003. Natural Language Processing. In Encyclopedia of Computer Science. 1218-1222

[7] Collobert R., Weston J., Bottou L., Karlen M., Kavukcuoglu K., Kuksa P. 2011. Natural language processing (almost) from scratch. In Journal of Machine Learning Research. 2493-2537.

[8] P. J. Hancox. N/A Morphological Analysis.. https://www.cs.bham.ac.uk/~pjh/sem1a5/pt2/pt2_intro_morphology.htmlhttps://www.cs.bham.ac.uk/~pjh/sem1a5/pt2/pt2_intro_morphology.html

[9] Wee Meng Soon, Hwee Tou Ng, Daniel Chung Yong Lim. 1999. A Machine Learning Approach to Coreference Resolution of Noun Phrases. In Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methodsin Natural Language Processing

[10] Fabio Calefato, Filippo Lanubile, Federico Maiorano, and Nicole Novielli. 2018. Sentiment polarity detection for software development. In Proceedings of the 40th International Conference on Software Engineering (ICSE '18). Association for Computing Machinery, New York, NY, USA, 128. DOI:https://doi-org.proxy.library.adelaide.edu.au/10.1145/3180155.3182519

[11] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, Michele Lanza, and Rocco Oliveto. 2018. Sentiment Analysis for Software Engineering: How Far Can We Go?. In ICSE '18: ICSE '18: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3180155.3180195

[12] Robbert Jongeling, Proshanta Sarkar, Subhajit Datta, and Alexander Serebrenik. 2017. On negative results when using sentiment analysis tools for software engineering research. Empirical Software Engineering (2017), 1–42.

[13] Md Rakibul Islam and Minhaz F. Zibran. 2017. Leveraging automated sentiment analysis in software engineering. In Proceedings of the 14th International Conference on Mining Software Repositories (MSR '17). IEEE Press, 203–214.

[14] Rada Mihalcea. 2004. Graph-based ranking algorithms for sentence extraction, applied to text summarization. In Proceedings of the ACL 2004 on Interactive poster and demonstration sessions (ACLdemo '04). Association for Computational Linguistics, USA, 20–es.

[15] Sergey Brin and Larry Page. 1198. The PageRank Citation Ranking: Bringing Order to the Web

[16] Rafael Ferreira, Frederico Freitas, Luciano de Souza Cabral, Rafael Dueire Lins, Rinaldo Lima, Gabriel França, Steven J. Simskez, and Luciano Favaro. 2013. A Four Dimension Graph Model for Automatic Text Summarization. In Proceedings of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) - Volume 01 (WI-IAT '13). IEEE Computer Society, USA, 389–396.

[17] Manning, C. and Raghavan, P., 2008. Introduction To Information Retrieval. 1st ed. Cambridge: Cambridge University.

[18] Jurafsky, D. and Martin, J., 2000. Speech And Language Processing. 3rd ed.

[19] AllenNLP. 2020. Constituency Parsing Demo. [online] Available at: <https://demo.allennlp.org/constituency-parsing> [Accessed 6 November 2020].

[20] DuckDuckGo. 2020. Duckduckgo Instant Answer API. [online] Available at: <https://duckduckgo.com/api> [Accessed 6 November 2020].

[21] Speir, M., 2016. HTTP Requests In Python 3. [online] Twilio. Available at: <https://www.twilio.com/blog/2016/12/http-requests-in-python-3.html> [Accessed 6 November 2020].

[22] Christoph Treude and Martin P. Robillard. 2016. Augmenting API documentation with insights from stack overflow. In Proceedings of the 38th International Conference on Software Engineering (ICSE '16). Association for Computing Machinery, New York, NY, USA, 392–403.

[23] Butterfield, A., Kerr, A. and Ngondi, G., 2016. A Dictionary Of Computer Science. 7th ed. Oxford: Oxford University Press.

[24] Tapas Kanungo and David Orr. 2009. Predicting the readability of short web summaries. In Proceedings of the Second ACM International Conference on Web Search and Data Mining (WSDM '09). Association for Computing Machinery, New York, NY, USA, 202–211.

[25] Luo Si and Jamie Callan. 2001. A statistical model for scientific readability. In Proceedings of the tenth international conference on Information and knowledge management (CIKM '01). Association for Computing Machinery, New York, NY, USA, 574–576.

[26] Acunetix. 2020. What Is SQL Injection (Sqli) And How To Prevent It. [online] Available at: <https://www.acunetix.com/websitesecurity/sql-injection/> [Accessed 6 November 2020].