Flexibility of Indonesian text pre-processing library

Dian Sa'adilah Maylawati[1], Hilmi Aulawi[2], Muhammad Ali Ramdhani[3]
[1]Department of Informatics, Sekolah Tinggi Tekologi Garut, Indonesia
[2]Department of Industrial Engineering, Sekolah Tinggi Tekologi Garut, Indonesia
[1,3]Department of Informatics, UIN Sunan Gunung Djati Bandung, Indonesia

Article Info

Article history:

Received Jul 27, 2018
Revised Aug 21, 2018
Accepted Nov 18, 2018

Keywords:

Flexibility
Software library
Software quality
Text pre-processing

ABSTRACT

This study aimed to achieve and measure flexibility as a software quality factor of text pre-processing libraries with Indonesian text from social media. Library was built based on a review of some text mining applications that did not yet have a special pre-process for Indonesian text. Text pre-processing libraries were designed and built using an object-oriented approach that was modular to achieve flexibility. Flexibility was measured by the Mc Call Cyclomatic Complexity (CC) metric. Flexibility of library was tested by implementing the library into text mining applications. The results of experiment showed that text pre-processing libraries could be flexible and easy to use without much configuration in text mining applications. It was proved by the value of CC of 2.51 which meant the library or software was not too complex, simple enough, and also flexible to use.

Corresponding Author:

Dian Sa'adilah Maylawati,
Department of Informatics,
Sekolah Tinggi Tekologi Garut, Indonesia.
Email: dsaadillah@sttgarut.ac.id

1. INTRODUCTION

Software quality was one of the important parts that needed to be accomplished in software development. Software quality factors to be achieved in accordance with the needs and objectives of software built [1-2]. There were various models of software quality factors, such as McCall which was the beginning of the development of software quality models [3], Boehm software quality model [4], FURPS [5-6], International Organization for Standardization (ISO) [7], CMM (Capability Maturity Model) [8-9], and other models were widely used as the goal of software quality to be achieved [10-12]. The software quality model evolved according to the needs of the software, one of which was a library or component-based software quality model [13-15].

Text mining technique was a technique to find important values previously unknown automatically by extracting text data so that obtained useful knowledge [16-21]. One of the problems in text mining was to represent text that was unstructured data into a structured data representation before the mining process. The process of preparing the data into a structured representation was called the pre-processing stage until the data had ready for the mining process [22-23].

Seeing so rapidly the needs of technology [24], especially on text management with text mining techniques appeared a variety of Text Mining applications such as Weka, Kea, Mallet, LingPipe, GATE, Carrot2, MinorThird, Simmetrics, and so forth. Building libraries that pack preprocessing for text mining could be useful in streamlining and simplifying the mining process. Where libraries or software libraries contained modules that had classes and functions [25], collection of functions on functional programming, or class definitions on object-oriented programming. Library was usually used for code reuse with specific functions that exist in a library reused to streamline program code, so it did not repeat the program code with the same algorithm. The result of the pre-processing library was a structured text representation that could be used for mining processes performed by text mining applications.

To meet the needs of text mining applications that vary, in this study pre-processing libraries built must certainly be able to meet the quality factor of flexibility. Flexibility was achieved by providing abstract classes that could be implemented as needed for text mining applications and measured by McCabe Cyclomatic Complexity (CC) Metrics. Where the McCabe CC metrics could measure software complexity, the lower the CC software value the simpler, easier, and more flexible to use, modify, and maintain [26-27]. In addition, library flexibility was also tested by implementing libraries in text mining applications.

2. RESEARCH METHOD

This research used Research and Development method (R&D) [28]. We studied the literatures about text pre-processing as research phase. We analyzed the needs of text pre-processing, the characteristics of Indonesian language, the needs of library, and built the library as development phase. We used Waterfall Software Development Life Cycle (SDLC) method for built the library. Waterfall SDLC was used because it was simple SDLC [10], systematic, and requirements of text pre-processing library had defined clearly, although today many software development methodology that develop, such as Agile methodology [29]. Then, for modeling the analysis and design, we used Unified Modeling Language (UML), because of UML was suitable for object oriented analysis and design that support modularity to reach flexibility also [30-31].

3. RESULTS AND DISCUSSION

3.1. Flexibility of Software Library

Flexibility was a software quality factor that was part of the category of product revision factor according to McCall, required to support software maintenance activities [3], [10], [13]. Likewise, in software component model, flexibility was sub quality or sub characteristic of maintainability [13]. Components had the ability to be modified in maintenance activities. Flexible components or software were easier to maintain [27]. Flexibility was different from adaptability that was a sub quality of portability [13], [15]. Adaptability was the ability of components to adapt in different platforms or environments without much modification [13]. Both flexibility and adaptability needed to had ease of use in a different environment. However, flexibility required components to be easily modified in the maintenance process, whereas adaptability required the ease of components when mounting in different environments.

Library as it was known as software reuse approach [32], so it needed to be flexible to change, in order to reduce the cost when maintained. Flexibility had sub quality including modularity and simplicity. Flexible software could be achieved by applying the principles of modularity, because the modular software reduced the complexity in the software. Modularity could be measured by evaluating cohesion and coupling software module design [10], [15], [33]. While simplicity would be measured by complexity metrics. One of the metrics for measuring flexibility and maintainability was the McCabe Cyclomatic Complexity Metrics [27]. By measuring the complexity of the library, it would also be measured libraries were easy to maintain or not.

McCabe cyclomatic complexity measured software complexity based on graph theory it was cyclomatic number, $V(g)=e-n+p$ [26]. Where, $V(g)$ is a cyclomatic complexity graph, e is the sum of edge, n is the sum of node, and p is saparated component or graph. The McCabe cyclomatic complexity used to calculate component complexity with $V(g)=e-n+2$ formulation. Where 2 was the addition of an additional edge from the exit node to the entry node on each module component. McCabe value was right if always that mean the value of component was 1. The limit of complexity value of cyclomatic complexity according to Software Engineering Institute among others 1-10 was a simple module without many risks, 11-20 mean more complex modules with multiple risks, 21-50 was a complex module with high risk, and more than 50 including programs which was untestable with a very high risk.

3.2. Indonesian Text Pre-Processing and Feature Extracting for Structured Text Representation

Doing text mining techniques was highly dependent on the language. This was because each language had unique characteristics, rules, spelling, and grammar. Especially for texts derived from social media, many natural languages, abbreviations, even slang languages were used so needed special handling. Therefore, it was necessary to understand well the grammar, in this study the Indonesian language, so that the result of the pre-process of the text was a good representation as well. Based on previous research, the text pre-processing produced structured text representation. Structured text representation data was ready for use in the mining process. Structured representation of a text in general there were two types, namely the form of single word or better known as the bag of words and the form of multiple words (n-gram). Bag of word was a structured representation of the text by collecting all the words in a text document without seeing the interrelationship between words [23], while the representation of multiple word was a text representation that collect words in text document by considering the interrelationship between words so that semantic meaning in the text document could be better preserved, because it could capture the relationship between words / phrases, even clauses and sentences [34-35]. Sequential pattern was one form of representation of multiple words, so that compared with single words, the meaning or knowledge of text documents can be maintained better [19].

Sequential pattern used in the text was called Sequence of Word (SOW) with a sequence of words that consider the order of occurrence of words. There were 3 structured text representations in SOW form that could be used [33-34], among others, Frequent Word Sequence (FWS) which was the set of FWS with respect to the order of occurrence of each word. Then, the set of Frequent Word Sequence (Set of FWS) which was a set of FWS set that not only pay attention to the sequence of word appearance but also pay attention to sequence of occurrence of sentences. The last, Frequent Word Itemsets (FWI) which was a FWI set that did not consider the order of occurrence of words. There was also FWI association (Set of FWI) as the development of SOW which had been proven able to maintain the meaning of Indonesian slang language well [36-38]. The pre-processing library in this study used Set of FWI as a structured text representation. Flowchart of Indonesian text pre-processing library shown nig Figure 1.

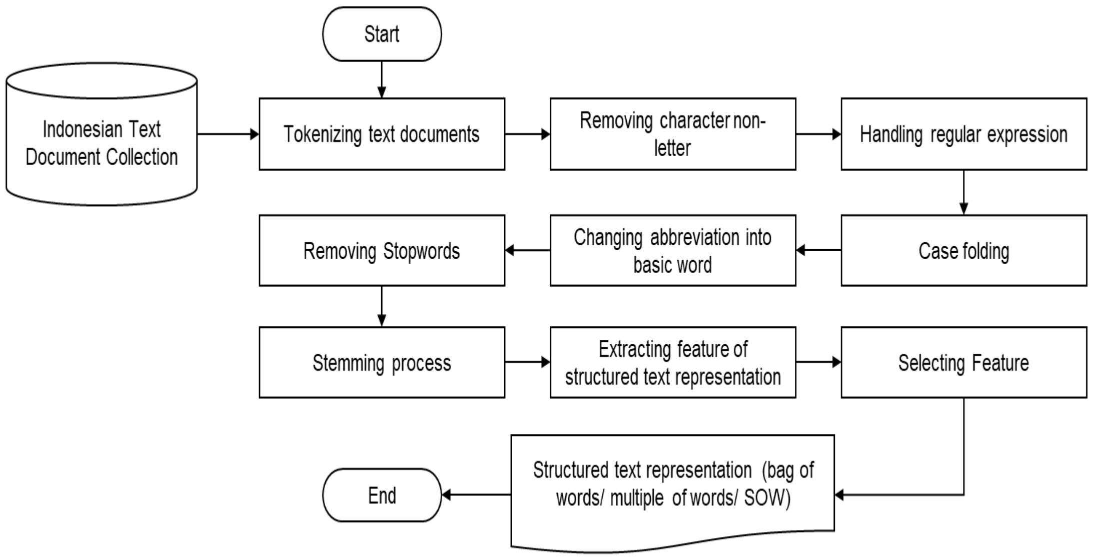

Start
Indonesian Text Document Collection
Tokenizing text documents
Removing character non-letter
Handling regular expression
Removing Stopwords
Changing abbreviation into basic word
Case folding
Stemming process
Extracting feature of structured text representation
Selecting Feature
End
Structured text representation (bag of words/ multiple of words/ SOW)

Figure 1. Flowchart of Indonesian Text Pre-processing Library

3.3. Modul Design of Indonesian Text Pre-Processing Library

Based on the needs of Indonesian text pre-processing described in the third section, we built a text pre-processing library with the architectural design shown in Figure 2. There were three main modules, including the Text Preprocessing module, the Feature Extraction module, and the module Featured Selection. Each module had an abstract classes as a prototype that could be implemented flexibly. The text pre-processing module was a collection of classes that performed tokenizing processes, removing non-letter characters, manipulating regular expression, case folding, converting abbreviations to their original form, removing stopwords, and stemming processes. The Feature Extraction module is a collection of classes that function to extract features in the form of structured text representation, while the Feature Selection module contains a set of classes that serve to select the extracted features because a feature may have sub-features that can be removed. Because in this research, Indonesian text is taken from social media so it is more appropriate to use Set of FWI as representation of structured text, so algorithm used is frequent pattern algorithm. One of the frequent pattern algorithms is the FP-Growth algorithm that does not generate feature candidates, but builds tree structures, making it more efficient [39-40].

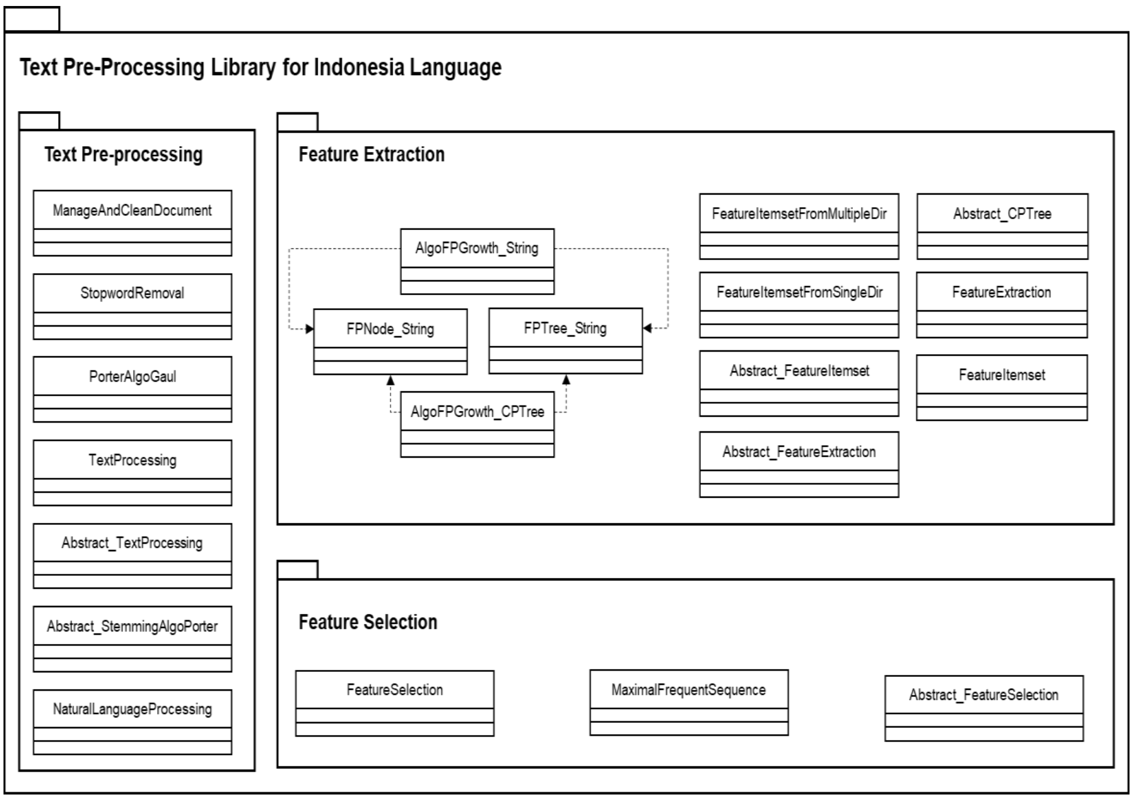
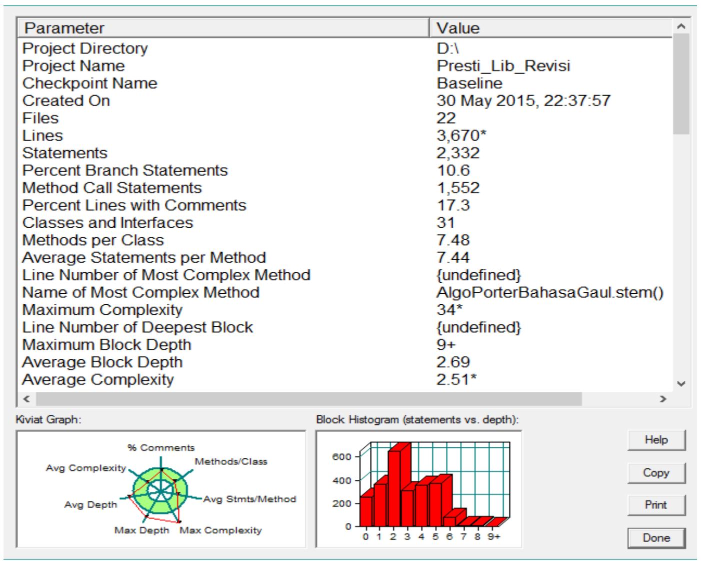


Figure 2. Modul design of Indonesian text pre-processing library

3.3.1 The Result of Library Measured by McCabe Cyclomatic Complecity Metrics

Evaluation of the libraries of preprocessing libraries using the McCabe cyclomatic complexity metrics calculated the complexity of the modules in the library. The result of the evaluation of the flexibility obtained by the average value of cyclomatic complexity 2.51 as in Figure 3. The value indicated the library was not too complex, quite simple and flexible (indicator value was gotten from Software Engineering Institute). However, there were some classes that had a cyclomatic complexity value greater than 10, even the PorterAlgoGaul.java class had a value of more than 50 which means it was very complex and inflexible.



Parameter	Value
Project Directory	D:\
Project Name	Presti_Lib_Revisi
Checkpoint Name	Baseline
Created On	30 May 2015, 22:37:57
Files	22
Lines	3,670*
Statements	2,332
Percent Branch Statements	10.6
Method Call Statements	1,552
Percent Lines with Comments	17.3
Classes and Interfaces	31
Methods per Class	7.48
Average Statements per Method	7.44
Line Number of Most Complex Method	{undefined}
Name of Most Complex Method	AlgoPorterBahasaGaul.stem()
Maximum Complexity	34*
Line Number of Deepest Block	{undefined}
Maximum Block Depth	9+
Average Block Depth	2.69
Average Complexity	2.51*

Figure 3. Result of Compexity Metrics of Indonesian Text Pre-processing Library

3.3.2 The Result of Testing on LingPipe 4.1.0

LingPipe was an application for text processing by using linguistic computing that could perform functions such as searching for names of people, organizations, or locations in the news, automatically classifying search results in categories, and correcting the correct spelling of a statement. LingPipe 4.1.0 could accept input data in plain text and xml. In the LingPipe app, the function that was invoked from the preprocessing library was readDir() from the class FrequentItemsetFromMultipleDir to generate the getFwi() feature of the FrequentItemset class to used features in the model development process, as well as PreprocessingTestFileToString() from the TextPreprocessing class to perform preprocesses on the data for testing. LingPipe received input data .txt, so library preprocessing could be directly used without changing the input type first. Examples of library tests for classification with DynamicLM algorithms were available in the LingPipe application. We used data from Twitter and Blog with three scenarios, among others: From 170 testing data, 136 data was classified correctly, among others 115 data in ”kampanye” class, and 21 data in ”tahunbaruislam” class; from 242 testing data, 87 data was classified correctly, among others 9 data in ”agama” class, 15 data in bisnis class, and 63 data in ”komputer” class; and total test data from blog was 48 with only 4 data correctly classified.

3.3.3 The Result of Testing Kea 5.0

Kea was a text-processing application that could perform tokenizers by extracting phrases, performing stopword removal and stemming, and modeling the process of document training and model testing. Models were built using the Naïve Bayes algorithm. Kea contracted the phrase of a document and the phrase was used as a keyword to train document data and as a label for classification. Kea had no algorithm for clustering process. The input for the data train process was a .txt file containing text for the train data and a key file with the same name as the .txt file. This .key file contained keyphrases for each .txt file. To prepare a phrase.key file containing phrases it could use the readDir() function of the FrequentItemsetFromMultipleDir class to generate features that could be used as keyphrase, getFwi() from the FrequentItemset class to use features in the model development process, as well as PreprocessingTestFileToString() from the TextPreprocessing class to perform preproces on training or test data. So that the test and training data had been done tokenize process, stopwords removal, and stemming. Because Kea did not provide stopword and stemming Indonesian language. Kea was very dependent on the language used, so if setVocabulary had a "none" value the resulting model could not be tested. Kea accepted only English, Spanish and French input. This was why the model with Indonesian produced could not be tested, but the model was successfully established.

3.3.4 The Result of Testing Mallet 2.0.6

Mallet (MAchine Learning for LanguagE Toolkit) was a Java-based package for the processing of natural language statistics, document classification, clustering, topic modeling, information extraction, and other machine learning applications for text. Mallet included powerful tools for document classification: efficient routines for converting text to "features", various algorithms (including Naïve Bayes, Maximum Entropy, and Decision Tree), and code for evaluating classifier performance using some commonly used metrics. As with LingPipe and Kea, preprocessing libraries could be added to Mallet. Same as the LingPipe application, the function was called from the preprocessing library was readDir() from the FrequentItemsetFromMultipleDir class to generate the getFwi() feature of the FrequentItemset class to used the features in the model development process, as well as the PreprocessingTestFileToString() of the TextPreprocessing class to perform preprocesses on the data for testing. Mallet also received input data .txt, so the preprocessing library could be directly used without changing the input type first. There were several examples of library testing with the same train and test data as in the LingPipe library test, for the process of classification with the Naïve Bayes algorithm. The classification results had an average accuracy of about 96.5%, this result was certainly influenced by good text data pre-process results.

4. CONCLUSION

Software needed to achieve quality in order to continue to be used, and easy to maintain and develop. There was a quality factor software that could be used as a measure of what quality to be achieved. Component-based software, including libraries, which were collections of modules, classes, or functions, have quality factors that needed to be achieved. One of them was the quality factor of flexibility. The pre-processing stage in text mining was an important stage that could affect the mining results. However, no many text mining applications had a complete pre-processing phase, especially for Indonesian text. Because, in the text mining needs and characteristics of each language on text data to be processed differently.

This research built and measured the flexibility of text pre-processing libraries for Indonesian text, ranging from tokenizing stages to forming structured text representations. The flexibility of library measurement using McCabe cyclomatic complexity metrics showed that text pre-processing libraries for Indonesian texts were flexible enough, simple, and easy to modify. This was proved by implementing library in 3 text mining applications, including LingPipe 4.1.0, Kea 5.0, and Mallet 2.0.6. Each application utilized the pre-processing library as a pre-process stage for document classification techniques. In addition, the library provided some abstract modules that could be implemented according to the needs of text mining applications. However, of course the programming language used affects the use of libraries, because the library could not be used in applications with different programming languages. Therefore, libraries must achieve with other quality factors, such as portability, adaptability, or interoperability that could be implemented in different environments (both languages and operating systems).

REFERENCES

[1] H. Aulawi, M. A. Ramdhani, C. Slamet, H. Ainissyifa, and W. Darmalaksana, “Functional Need Analysis of Knowledge Portal Design in Higher Education Institution,” *Int. Soft Comput.*, vol. 12, no. 2, pp. 132–141, 2017.
[2] T. Wahyuningrum and K. Mustofa, “A Systematic Mapping Review of Software Quality Measurement: Research Trends, Model, and Method,” *Int. J. Electr. Comput. Eng.*, vol. 7, no. 5, p. 2847, 2017.
[3] J. A. McCall, P. K. Richards, and G. F. Walters, “Factors in Software Quality - Volume 1 - Concept and Definitions of Software Quality,” *Def. Tech. Inf. Cent.*, 1977.
[4] B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. McLeod, and M. Merrittt, *Characteristics of Software Quality*. North Holland, 1978.
[5] S. Tripathi, “A Survey on Quality Perspective and Software Quality Models,” *IOSR J. Comput. Eng.*, 2014.
[6] R. Al-Qutaish, “Quality models in software engineering literature: an analytical and comparative study,” *J. Am. Sci.*, 2010.
[7] ISO/IEC, “Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models,” *ISO/IEC Fdis 250102011*, 2011.
[8] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber, “Capability maturity model, version 1.1,” *IEEE Softw.*, 1993.
[9] Y. Fang, B. Han, and W. Zhou, “Research and Analysis of CMMI Process Improvement Based on SQCS System,” *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 10, no. 7, pp. 1849–1854, 2012.
[10] R. S. Pressman, *Software Engineering: A Practitioner’s Approach*, 7th ed. New York: McGraw-Hill, 2011.
[11] I. Sommerville, *Software Engineering*. 2010.
[12] M. A. Kabir, M. U. Rehman, and S. I. Majumdar, “An analytical and comparative study of software usability quality factors,” in *7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2016.
[13] A. Sharma, R. Kumar, and P. S. Grover, “Estimation of Quality for Software Component - an Emporical Approach,” *SIGSOFT Softw. Eng. Notes*, vol. 33, no. 6, 2008.
[14] Y. e. a. Choi, “Practical S/W Component Quality Evaluation Model,” in *ICACT*, 2008.
[15] A. Alvaro, E. Santana de Almeida, and S. Romero de Lemos Meira, “A software component quality framework,” *ACM SIGSOFT Softw. Eng. Notes*, 2010.
[16] C. J. Torre, M. J. Martin-Bautista, D. Sanchez, and I. Blanco, “Text Knowledge Mining: And Approach To Text Mining,” *ESTYLF08*, vol. 17–19, 2008.
[17] V. Gupta and G. S. Lehal, “A survey of text mining techniques and applications,” *Journal of Emerging Technologies in Web Intelligence*, vol. 1, no. 1. pp. 60–76, 2009.
[18] H. Jiawei, M. Kamber, J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*. 2006.
[19] A. Hoonlor, “Sequential Patterns and Temporal Patterns for Text Mining,” New York, 2011.
[20] M. R. Islam, I. F. Al-Shaikhli, R. B. M. Nor, and V. Varadarajan, “Technical approach in text mining for stock market prediction: A systematic review,” *Indones. J. Electr. Eng. Comput. Sci.*, vol. 10, no. 2, pp. 770–777, 2018.
[21] R. Gunawan and K. Mustofa, “Finding knowledge from Indonesian traditional medicine using semantic web rule language,” *Int. J. Electr. Comput. Eng.*, vol. 7, no. 6, pp. 3674–3682, 2017.
[22] H. Mahgoub, D. Rösner, N. Ismail, and F. Torkey, “A Text Mining Technique Using Association Rules Extraction,” *Int. J. Comput. Intell.*, vol. 4, no. 1, pp. 21–28, 2008.
[23] Y. E. Zohar, “Introduction to Text Mining,” *Automated Learning Group, University of Illinois*, 2002. [Online]. Available: http://www.docstoc.com/docs/25443990/Introduction-to-TextMining.
[24] M. A. Ramdhani, H. Aulawi, A. Ikhwana, and Y. Mauluddin, “Model of green technology adaptation in small and medium-sized tannery industry,” *J. Eng. Appl. Sci.*, 2017.
[25] C. e. a. Szyperski, *Component Software, Beyond Object-Oriented Programming*, 2nd ed. Great Britain: Addison-Wesley, Rearson Education Limited, 2002.
[26] T. J. McCabe, “A Complexity Measure,” *IEEE Transactions on Software Engineering*. 1976.
[27] F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert, and J. F. Girard, “An activity-based quality model for maintainability,” in *IEEE International Conference on Software Maintenance, ICSM*, 2007.
[28] D. Mahdjoubi, “The Linear Model of Technological Innovation: Background and Taxonomy,” *The Atlas of Innovation*. 1997.
[29] N. Sharma and M. Wadhwa, *eXSRUP: Hybrid Software Development Model Integrating Extreme Programing, Scrum & Rational Unified Process*. 2015.
[30] M. A. Ramdhani, D. S. Maylawati, A. S. Amin, and H. Aulawi, “Requirements Elicitation in Software Engineering,” *Int. J. Eng. Technol.*, vol. 7, no. 2.29, pp. 772–775, 2018.

[31] A. E. H. Soumiya and B. Mohamed, "Converting UML Class Diagrams into Temporal Object Relational DataBase," *Int. J. Electr. Comput. Eng.*, vol. 7, no. 5, p. 2823, 2017.

[32] B. Jalender, a. Govardhan, and P. Premchand, "Designing code level reusable software components," *Int. J. Softw. Eng. Appl.*, 2012.

[33] B. Meyer and P. America, "Object-oriented software construction 2nd Ed.," *Sci. Comput. Program.*, 1989.

[34] A. Doucet and H. Ahonen-Myka, "Non-contiguous word sequences for information retrieval," *MWE '04 Proc. Work. Multiword Expressions*, vol. 26, no. July, pp. 88–95, 2004.

[35] A. Doucet and H. Ahonen-Myka, "An efficient any language approach for the integration of phrases in document retrieval," *Lang. Resour. Eval.*, vol. 44, no. 1–2, pp. 159–180, 2010.

[36] D. S. Maylawati, "Pembangunan Library Pre-Processing untuk Text Mining dengan Representasi Himpunan Frequent Word Itemset (HFWI) Studi Kasus: Bahasa Gaul Indonesia," Bandung, 2015.

[37] D. S. A. Maylawati, M. A. Ramdhani, A. Rahman, and W. Darmalaksana, "Incremental technique with set of frequent word item sets for mining large Indonesian text data," in *2017 5th International Conference on Cyber and IT Service Management, CITSM 2017*, 2017.

[38] D. Sa'Adillah Maylawati and G. A. Putri Saptawati, "Set of Frequent Word Item sets as Feature Representation for Text with Indonesian Slang," in *Journal of Physics: Conference Series*, 2017.

[39] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: Current status and future directions," *Data Min. Knowl. Discov.*, vol. 15, no. 1, pp. 55–86, 2007.

[40] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Min. Knowl. Discov.*, vol. 8, no. 1, pp. 53–87, 2004.