International Journal of Trend in Scientific Research and Development (IJTSRD)
Volume: 3 | Issue: 3 | Mar-Apr 2019 Available Online: www.ijtsrd.com e-ISSN: 2456 - 6470

Cohesive Software Design

Janani Tharmaseelan

Assistant Lecturer, Sri Lanka Institute of Information Technology, Colombo, Sri Lanka

How to cite this paper: Janani Tharmaseelan "Cohesive Software Design" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-3 | Issue-3 , April 2019, pp.955-957, URL: https://www.ijtsrd.com/papers/ijtsrd22900.pdf

IJTSRD22900

Copyright © 2019 by author(s) and International Journal of Trend in Scientific Research and Development Journal. This is an Open Access article distributed under the terms of the Creative Commons Attribution License (CC BY 4.0) (http://creativecommons.org/licenses/by/4.0)

ABSTRACT

This paper presents a natural language processing based automated system called DrawPlus for generating UML diagrams, user scenarios and test cases after analyzing the given business requirement specification which is written in natural language. The DrawPlus is presented for analyzing the natural languages and extracting the relative and required information from the given business requirement Specification by the user. Basically user writes the requirements specifications in simple English and the designed system has conspicuous ability to analyze the given requirement specification by using some of the core natural language processing techniques with our own well defined algorithms. After compound analysis and extraction of associated information, the DrawPlus system draws use case diagram, User scenarios and system level high level test case description. The DrawPlus provides the more convenient and reliable way of generating use case, user scenarios and test cases in a way reducing the time and cost of software development process while accelerating the 70% of works in Software design and Testing phase

KEYWORDS: *Natural Language Processing; NLP; UML automation; test case generation; Open NLP; Grammar Algorithm; User Scenario Automation; NLP Co-reference; Design phase Acceleration; raw requirement analyze; Actor identification; function identification*

I. INTRODUCTION

Software design and software testing is known as some of the key critical areas in the software development life cycle which takes considerable amount of time and cost to a software project. Basically in the software design phase system analysis are design the software by using various diagrams (UML) or charts bases on requirement specification given by the business. In the current process of designing a software almost every design is done by manually by utilizing large amount of human effort and can have some reliability issues with the design.

When it's come to the testing phase it is the last phase of SDLC before software is delivered and in this phase software is test against the requirements. Here generating test cases is one of the main task and testers face many problems when generating test cases such as ambiguity in the requirements, huge time takes to read and understand the requirement specification etc. By considering the all of those problems we came up with an integrated software design tool which is capable of generating use case, user scenarios and Test cases for the given requirement

Specification in a way accelerate the Design and Test phase of the SDLC. So that this paper proposes an approach to generate UML diagrams, user scenarios test case from software requirements expressed in natural language using natural language processing techniques. Basically, system takes and preprocess the requirements specification as the input, then use NLP core techniques such as sentence detector, tokenizer, pos (part-of-speech) tagger and de-tokenizer with our own algorithms to capture and filter only necessary parts of the given requirement specification. Finally system produce outputs in three forms such and UML diagrams, user scenarios and test cases.

II. METHODOLOGY

For the domain of this research has carried out there are always limited ways of stating a requirement. Hence it's a matter of catching each possible ways of stating a requirement. But increase in the Grammar array may reduce performance. For each grammar we have defined, there is a corresponding rule to be applied in order to perform the actor and function extraction. There for there comes the Rule set. As a result of this research project, the research team was able to introduce few new algorithms to be used with Open NLP libraries. The algorithm is applicable when the concept of grammar is involved. Grammar is composed of multiple number of tags with the expected sequence of the tags.

The algorithm processes one sentence at a time. Initially the grammar matching algorithm receives a sentence with all the words are tagged [5] by the POS Tagger [1]. The entire function is running on top of a nested loop consist of two nested loops. The algorithm then has three loops inside, each is nested to the other. Altogether it makes 5 loops nested to each other and containing more than 13 conditions in between. The Grammars and Rules are defined and store in a two dementional array. For each grammar there we have defined, thers is a coresponding Rule. The rule contains

information and instructions to follow when a particular grammar was hit. The correct rule for a particular grammar can always be refered by the index number of the grammar. The *Figure 1* shows how the above mentioned loops are nested and the connection and the purpose of each loop.

Figure 1 - Loop structure

Considering the 3 loops inside the algorithm, the first loop concerns about the grammar number which is now being compared, and the second loop concerns about the tokenized tagged sentence. The tokenized sentence is prepared by tokenizing the tagged sentence using the Open NLP Tokenizer [3]. The third and the last loop concerns about multi valued tags if defined any in the selected grammar. Multi valued tags are cases where the grammar indicates, in a particular tag position whether there can be any alternative tags. This technique is lately added to improve the efficiency of the algorithm. With the multi-valued check technique, we were able to reduce more than half of the grammars and number of rules defined in the method. This technique also improves the overall efficiency of the entire process.

Since the first loop concerns about the Grammar rules number, increment in a single grammar would cause a massive number of comparisons inside the next two loops. As an average value for a description with 12 sentences, the algorithm makes 710 comparisons for 22 grammars defined. Therefore reduction in the number of grammars effect all three inside loops directly. The multi-valued tags are identified by the "/" sign in between two tags in a single tag position. The efficiency rate is shown below in the Figure 1 and Figure 2. The tests were carried out by testing giving random sentences as the input for the program. Test 1 contains 12 sentences, test 2 contains 10 sentences and test 3 contains 6 sentences. The test result show that the technique has an effective and direct impact for the performance of the overall process. As the decrease in the grammar resulted in a decrease in number of comparisons, the total time taken to process the data were also affected and resulted a decreased. The test results shown in Figure 2 and Figure 3 shows direct and effective outcomes of the application of the multivalued tag check into the grammar.

Figure 2 - Comparison efficiency

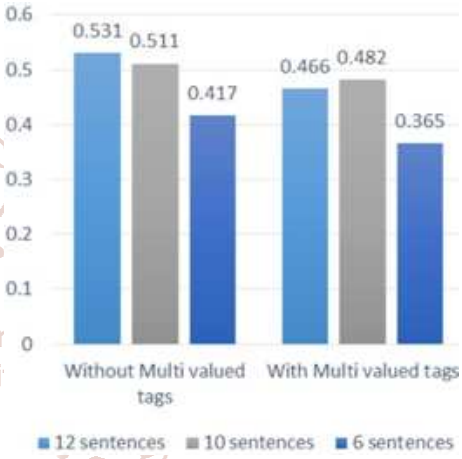

Figure 3 – Duration efficiency

There can be cases where this techniques may not show any performance, cases where any sentences doesn't match with a grammar that does have a multi-valued tag. When finding a matching grammar, whenever small segments in the grammar is matched with the segments in the sentence, the indexes of the matched tokens are saved in a string array to be used when locating the actions and actors. The Grammars are stored in a special order. First comes the longest in length to avoid conflict where one grammar can be consisted inside another.

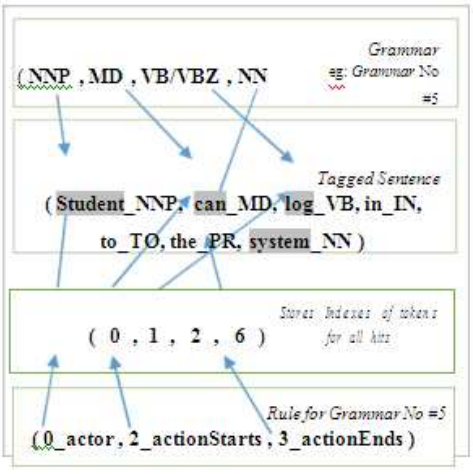

Figure 4 - Grammar matching algorithm

The Figure 4 shows the wireframe of the Grammar matching algorithm. The third block of the figure 4 refers to an array used for a special purpose in this algorithm. The array simply bridges the sentence and the Rule. Then indicates where the grammar findings lie inside the original sentence. The content inside the Rule (last block in figure 4) show where to look for. The numbers at the first index in each element refers to the special array in section three. Where ultimately it is pointed to the indexes of the original sentence. When a sentence completely matches a grammar, the matched grammar number in the grammar set is passed into two algorithms named getActor() and getAction(). The both algorithms take grammar number as the input and apply the rules defined specifically for the matched grammar number which will produce the ultimate result, the actor and the actions executed by the same actor. The algorithm can manage advanced relationships between the data.

The algorithms is capable of identifying multiple actors with their actions mentioned in the same sentence and multiple actions mentioned to be executed by a particular actor. The algorithm is further developed to support co-reference in an advanced manner. When a grammar was hit the mentioned actor is stored in a variable carries forward till the next actor gets a hit. Meanwhile if an action is caught without a proper actor and if the sentence matches the right grammar requirements, the previous actor that was carried forward will be patched as the actor. The algorithm is also maintaining two variables to get statistics of hit counts and miss counts when matching a grammar. The two count variables will be resetted each time it is switching between grammars. This count can be used to prompt the users as an indication of the confidence level of each segment of the result.

Since the output of these algorithms can complex as the data is related to one another, these data cannot be stored in regular variables available in JAVA [4]. Therefore we have created a data structure to maintain data, special methods to retrieve the data or insert new data.

When it comes to the the UML design phase main input is an XML file which is generated form NLP core. For that a rich algorithm was implemented. Initial location of diagram is provided in the first step. The algorithm is executed in this flow. First actor of the XML is identified and get number of functions intended for that actor. According to that X, Y location 1st actor is drown in left side. Then get that actor X axis value, changing Y axis value and draw the 1st function from the function set. Define a constant value to put a space in between two functions. The remaining functions are drown related to the selected actor by changing X axis value. Y axis value needs to be changed for all the actor up until left side allocated space is over. 2nd actor Y alliance is taken by using this formula .

Whole functions area= number of function count*(function high + difference)

Next actor location= whole functions area + difference

Another condition is to be checked before draw the actor. End point for last function for the next actor <= window size. If this condition is false then that should be draw in left side else it should be move to opposite side.

To draw arrow in between actors is needed two parameters. Actor location and the function location are the both location and actor location is taken by dividing the height of the actor and function location is taken by getting value of X axis. When the actor move to opposite side these two parameters are changed according to that location.

III. CONCLUSION

In this paper, we have presented a structure of modeling use case diagram, use case scenario and high level test case description extracted from the given description. We have introduced an intelligent algorithm to match the grammar of many different form. The algorithm is smart enough to notify the user the certainty of the particular piece of findings. From this algorithm we can identify the relationship of actors and their functions.

The results shows evidence that improving the algorithm is much more effective than increasing the number of grammar to be checked. Which elevated the total performances of the system. The accuracy can be increased by adding more and more grammar rules into the application. Therefore we figured out that each step we take trying to improve the accuracy the performance of the software reduces. But the fact is no user is willing to get the chance of having a possible faulty results over an extra few seconds that they have to spend more.

Acknowledgment

It gives us immense pleasure and satisfaction in submitting this research paper. In the endeavor of preparing this paper many people gave us a helping hand. So it becomes our duty and pleasure to express our deep regard to them.

References

[1] NLP with JAVA [Online] http://stackoverflow.com/questions/5836148/how-to-use-opennlp-with-java

[2] Part-Of-Speech tags [Online] https://www.ling.upenn.edu/courses/Fall_2003/ling001/pen n_treebank_pos.html

[3] Stuart J. Russell and Peter Norving "Artificial Intelligence A Modern Approach" 1995

[4] Stack overflow [Online] http://stackoverflow.com/

[5] Rob Callan "Artificial Intelligence" 2003

[6] Software Development Life Cycle [Online] Available http://www.tutorialspoint.com/software_engineering/softwa re_development_life_cycle.htm

[7] Progtamcreek [Online] Available http://www.programcreek.com/2012/05/opennlp-tutorial/

[8] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE, 1989, pages 257-286

[9] The canonical paper on LDA: David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. Journal of Machine Learning Research, 3:993–1022, 2003

[10] Parsing: Dan Klein and Christopher D. Manning. Accurate un lexicalized parsing. In ACL, pages 423–430, 2003.