1 Related Work

In Software Engineering, documentation on source code provides an important role in Software Development. Documentation help users understand the behavior of the program built by that source code. In the other hand, documentation can be provided as an input for developers to develop program. Documentation, which is described in the form of natural language, reflects the same intent of the behavior of the program in programming languages. Inferring natural language from programming language and vice versa, which are usually done manually by software developers, are effort consuming and error prone, since the inference require developers to have knowledge from writing both type of languages. To overcome this challenge, approaches for automatically deriving natural language and vice versa have been designed.

To support developers for better understanding source code by documentation in some restricted type of natural language, there are researches on generating pseudo code [1] and code comment [2]. [1] applied Statistical Machine Translation (SMT) approach to generate pseudo-code from Python source code. This work proposed an improvement for original Phrase-To-Phrase translation commonly used in Natural Language Processing (NLP), by providing a Tree-To-String translation approach. They built a parallel corpus at method levels, with the source language reflects Abstract-Syntax-Tree, and target language reflects pseudo code of the same method. This work limited the generation process statement-by-statement, means the pseudo-code was generated for each statement in the source code. This limitation caused insufficient for generated pseudo-code in large software project environments contains more than thousands line of code. [2] proposed an approach for extract the comments from Java Methods that summarize Java Methods' content. This work also generate at method level, by the extraction of s-unit, which is a set of important central lines of code, then generate the summary for lines in s-unit using a Software Word Usage Model constructed for each method.

The inference from Natural language to Programming language seems to be more tackle compare to the other direction of reference, since this problem needs to deal with Natural Language Grammar analysis. There are two trends of researches: analyzing Natural Language versus Programming Language and propose a solution for the inference in limited scope of specific purposes and specific programming paradigm.

For analyzing Natural Language and Programming Language grammar, one of the starting point was made in 1977 by [3]. This work proposed a project name Natural Language Programming,which analyzing natural language programs to have semantic representation of 1000 unique-words. [4] provided experiments to identifies which patterns in source code can lead to misunderstanding in software development. [5] analyzed the differences between programming language grammars, which were built by Context-Free-Grammar versus Natural Language grammar. The experiment shows that using Natural Language grammar can be used as an alternative for human to learn the fundamental of programming, which is more attractive and promising. [6] analyses characteristics of natural language by the mapping programming language. This work proposed the idea of naturalistic types, which contain 4 elements: concept, properties, quantities and conditions. In game programming development, [7] argued that entire natural language will not be a viable option for coding. Based on the experiment. [7] suggested some design rules for making use of natural language, two of them are constrain the programming language to minimize syntax error and the natural language used should be consistent with typical everyday use and avoid unnatural syntax or phrasing.

There are researches on approaches for synthesizing programming language in [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] and [18]. General, these work proposed solution that suited for specific types of programming and specific programming language. One of most well-known application in this area is WolframAlpha [8]. This work provided a Computation Engine that allow users to have the calculation output from well-known algorithms by their description in queries, commonly contain the name of algorithm and input arguments. [9] proposed a technique for generating assembly code from users' natural language description for industrial robot actions. This work applied state-of-the-art NLP technique for semantic parsing and syntactic parsing to get the set of predicate-argument which represent a sequence of tasks from the description, then mapping each predicate-argument to related action-object of the simulated word. The action-object is predefined and stored in a unified architecture. [10] designs a new language Quasi-Natural language that focus on knowledge representation, which contains data structures and suitable operations. Thís language provides support for natural language which are in certain lexical and syntactic rules. [11] provides a program synthesis approach for Domain Specific Language (DSL) by inferring a dictionary relation over pairs of English words and DSL terminal semi-automatically. [12] proposed another approach for program synthesis, which consider synonyms problem and provide modules to handle each type of statements in programming language. [13] focuses on the Control structure of the programming language and provide an approach for using type dependency in Natural language sentence to infer the its control structure. [14] proposes an heuristic approach to identify basic elements of Object-Oriented Programming, including class name and variables/ attributes. This work uses format new heuristics rules manual from a training data set, then apply and evaluation the correctness of these rules in another set of natural language description corpus. SWIM [15] uses data-driven approach to synthesize code snippet from users queries by 2 steps: extracting Ranked APIs set for each query and selected structured call sequence for each API. The Query to API model was learnt from Clickthrough data of Bing while Structured call sequence was learnt from Github Code corpus. Pegasus [17], which is a first step implementation of ideas of naturalistic types in [6] designs their approach by 3 processes: reading natural language, expressing natural language and expressing natural language. One of most recent work, [18] combines ideas of parsing technique from Natural language and type-directed synthesis and program repair from Programming language to generate SQL query from description. To be concluded, most of current works are focusing on generating code for a limited scope of programming tasks and languages.

The advantages of automatically inference leads to researches on how to fulfill the gap between natural language and programming language by an unified language structure. The most famous programming paradigm supported this idea is literate programming, which was invented by Donald Knuth [19]. Surprisingly, there were not many publications for the improvement from this programming paradigm, possibly due to the lack of techniques to implement this idea in the past. The most well-known of them are [20] [21] [22] [23] [24] [25] [26] [27]. [20] provided an application of literate programming in object-oriented code, to support develop computational Electromagnetics (CEM) library. In this work, literate programming environment FWEB for C++ language help the users by generating human-readable documentation in TEX format. [21] provides a chunk model which allow unlimited code and document types, and a theme model which provide multiple views of a given system based on different descriptions of a program aimed at different group. [22] extends the literate programming tool to support users, using a document processor LYX to represent literate documents by tree representation. [23] describes about advantages of literate programming

paradigm, and the this paradigm should be used in the design phase of software development, besides implementation and maintenance phases as before. VAMP [24] is a tool for support literate programming by auto generate modules. Compared to its prior work WEB, VAMP is not restricted to a single programming language. [25] proposed P-Coder, a tool for creating a better description of the program using literate notation. [26] provides a tool LP/ Lisp, which support literate programming for a language with incremental development and testing like LISP. Unlike previous LP tools which are suitable only for compiled languages that naturally have a step between writing and executing the code, LP/Lisp supports the Literate Programming in running portions of the code. [27] states about challenge of literate programming for new programmer, since it consists of 3 different languages: the implementation language, documentation language and literate programming glue language.This work propose Ginger, a language that considers literate programming language as core feature and propose G-expression transformation for represent naturally code as well as documentation and linear connections. In concluded, existing literate programming researches focuses on extending the original paradigm to support Object-Oriented programming and visualization tools. Nowadays, researches for improving literate programming with large-scale repositories and current Natural Language to Programming Language Inference techniques are needed.

References

1 Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. Learning to generate pseudo-code from source code using statistical machine translation (t). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '15, pages 574–584, Washington, DC, USA, 2015. IEEE Computer Society. ISBN 978-1-5090-0025-8. 10.1109/ASE.2015.36. URL `https://doi.org/10.1109/ASE.2015.36`.

2 Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K. Vijay-Shanker. Towards automatically generating summary comments for java methods. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, pages 43–52, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0116-9. 10.1145/1858996.1859006. URL `http://doi.acm.org/10.1145/1858996.1859006`.

3 Lance Miller. Natural language programming. *SIGART Bull.*, (61):32–32, February 1977. ISSN 0163-5719. 10.1145/1045283.1045299. URL `http://doi.acm.org/10.1145/1045283.1045299`.

4 Dan Gopstein, Jake Iannacone, Yu Yan, Lois DeLong, Yanyan Zhuang, Martin K.-C. Yeh, and Justin Cappos. Understanding misunderstandings in source code. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 129–139, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5105-8. 10.1145/3106237.3106264. URL `http://doi.acm.org/10.1145/3106237.3106264`.

5 Norton Trevisan Roman Osvaldo Luiz Oliveira, Ana Maria Monteiro. Can natural language be utilized in the learning of programming fundamentals? In *Frontiers in Education Conference, 2013 IEEE*. IEEE.

6 Roman Knöll, Vaidas Gasiunas, and Mira Mezini. Naturalistic types. In *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2011, pages 33–48, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0941-7. 10.1145/2048237.2048243. URL `http://doi.acm.org/10.1145/2048237.2048243`.

7 Judith Good ; Kate Howland. Natural language and programming: Designing effective environments for novices. In *Visual Languages and Human-Centric Computing (VL/HCC), 2015 IEEE Symposium*. IEEE.

8 Wolframalpha computational knowledge engine. http://www.wolframalpha.com.

9 Pierre Nugues Maj Stenmark. Natural language programming of industrial robots. In *Robotics (ISR), 2013 44th International Symposium*. IEEE.

10 Pu Yin. Natural language programming based on knowledge. In *Artificial Intelligence and Computational Intelligence (AICI), 2010 International Conference*. IEEE.

11 Aditya Desai ; Sumit Gulwani ; Vineet Hingorani ; Nidhi Jain ; Amey Karkare ; Mark Marron ; Sailesh R ; Subhajit Roy. Program synthesis using natural language. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference*. IEEE.

12 Tejas Karwa Sharvari Nadkarni, Parth Panchmatia and Swapnali Kurhade. Semi natural language algorithm to programming language interpreter. In *Advances in Human Machine Interaction (HMI), 2016 International Conference*. IEEE.

13 Mathias Landhauber and Ronny Hug. Text understanding for programming in natural language: Control structures. In *Proceedings of the Fourth International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, RAISE '15, pages 7–12, Piscataway, NJ, USA, 2015. IEEE Press. URL http://dl.acm.org/citation.cfm?id=2820668.2820671.

14 Nazlia Omar ; Nomariani A. Razik. Determining the basic elements of object oriented programming using natural language processing. In *Information Technology, 2008. IT-Sim 2008. International Symposium*. IEEE.

15 Mukund Raghothaman, Yi Wei, and Youssef Hamadi. Swim: Synthesizing what i mean: Code search and idiomatic snippet synthesis. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 357–367, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3900-1. 10.1145/2884781.2884808. URL http://doi.acm.org/10.1145/2884781.2884808.

16 Hugo Liu and Henry Lieberman. Metafor: Visualizing stories as code. In *Proceedings of the 10th International Conference on Intelligent User Interfaces*, IUI '05, pages 305–307, New York, NY, USA, 2005. ACM. ISBN 1-58113-894-6. 10.1145/1040830.1040908. URL http://doi.acm.org/10.1145/1040830.1040908.

17 Roman Knöll and Mira Mezini. Pegasus: First steps toward a naturalistic programming language. In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*, OOPSLA '06, pages 542–559, New York, NY, USA, 2006. ACM. ISBN 1-59593-491-X. 10.1145/1176617.1176628. URL http://doi.acm.org/10.1145/1176617.1176628.

18 Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. Sqlizer: Query synthesis from natural language. *Proc. ACM Program. Lang.*, 1(OOPSLA):63:1–63:26, October 2017. ISSN 2475-1421. 10.1145/3133887. URL http://doi.acm.org/10.1145/3133887.

19 Donald E. Knuth. Literate programming. *Comput. J.*, 27(2):97–111, May 1984. ISSN 0010-4620. 10.1093/comjnl/27.2.97. URL http://dx.doi.org/10.1093/comjnl/27.2.97.

20 P. Atlamazoglou D.G. Lymperopoulos, D. Logothetis. Using object-oriented and literate-programming techniques for the development of a computational electromagnetics library. In *IEEE Antennas and Propagation Magazine*. IEEE.

21 A. Kacofegitis ; N. Churcher. Theme-based literate programming. In *Software Engineering Conference, 2002. Ninth Asia-Pacific*. IEEE.

22 H. Anderson. Formalization and 'literate' programming. In *Software Engineering Conference, 2001. APSEC 2001. Eighth Asia-Pacific*. IEEE.

23 D. Cordes M. Brown. A literate programming design language. In *1990 IEEE International Conference on Computer Systems and Software Engineering*. IEEE.

24 M.R. Kramer E.W. van Ammers. Vamp: A tool for literate programming independent of programming language and formatter. In *CompEuro '92 . 'Computer Systems and Software Engineering',Proceedings.* IEEE.

25 Geoffrey G Roy. Designing and explaining programs with a literate pseudocode. *J. Educ. Resour. Comput.*, 6(1), March 2006. ISSN 1531-4278. 10.1145/1217862.1217863. URL `http://doi.acm.org/10.1145/1217862.1217863`.

26 Roy M. Turner. Lp/lisp: Literate programming for lisp. In *Proceedings of the 2010 International Conference on Lisp*, ILC '10, pages 21–28, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0470-2. 10.1145/1869643.1869647. URL `http://doi.acm.org/10.1145/1869643.1869647`.

27 James Dean Palmer and Eddie Hillenbrand. Reimagining literate programming. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, pages 1007–1014, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-768-4. 10.1145/1639950.1640072. URL `http://doi.acm.org/10.1145/1639950.1640072`.