Software Analysis using Natural Language Queries

Pooja Rani
Software Composition Group
University of Bern, Switzerland
http://scg.unibe.ch/staff/Pooja-Rani

Abstract

Understanding a software system consumes a substantial portion of a developer's effort. To support software comprehension and evolution, reverse engineering aims at creating a high-level representation of an existing software system. With state-of-the-art technology, abstract models of software systems are created by reverse engineering tools and analyzed using software analysis tools. Despite the rich functionalities offered by analysis tools, a novice user may find them difficult to use due to an unfamiliar tool environment and query language. In this paper, we propose an approach that allows the developer to formulate a query in a natural language in order to overcome these obstacles.

1 Introduction

Software systems continuously evolve and present multiple challenges for performing analysis. To deal with the expanding size and complexity of software systems, developers employ reverse engineering methods and tools to extract information [MJS+00].

With the adoption of model-driven engineering, models have become a potential center of software development due to their usage in numerous reengineering tasks like project analysis, software maintenance and software system understanding [RS04]. Models represent software artifacts, and the source code is one of the important software system artifacts spanning various phases starting from development to maintenance. Different kinds of analysis like data flow analysis, call graph analysis or code metric analysis are performed to understand the source code. Developers ask various questions regarding function definitions or annotations used in classes in order to understand the code. Studies have revealed common search patterns regarding questions that developers ask such as *"Where is this method called?"* or *"Which methods call the method named* `searchForThreat` *in the abc package?"* [SMDV06, SCH98].

There are numerous tools available to perform such analyses [EKRW02, FBTG02]. One of them, implemented in Smalltalk, is Moose [DGLD05]. Moose is a software and data analysis platform designed to be extensible, exploratory and scalable, and it provides the possibility of performing hybrid analysis. It uses a meta-model named FAMIX (FAMOOS Information Exchange Model). FAMIX is a programming language (PL) independent meta-model used to represent object-oriented source code [TDD00]. A developer can use it to analyze the source code of different programming languages. In order to analyze source code in Moose, the developer needs to be familiar with the meta-model and the query language. This is a common problem for a number of analysis tools [EKRW02, FBTG02].

Consider a query the developer wants to ask while analyzing the model: *"Which classes are deprecated?"*. In order to get the list of deprecated classes, the developer needs to turn this natural language (NL) question into the tool's query language. In Moose, queries are formulated in Smalltalk. The Moose query for selecting the classes that are deprecated is as follows:

```
self allModelClasses select:
        [ :each | each isAnnotatedWith:
                        'Deprecated' ]
```

To effectively use the tool's abilities, a developer needs to be aware of the query language of the tool and the meta-model structure, *e.g.,* if the meta-model contains any class or method property related to annotations. If the tool supports "annotated" as a property, the developer can translate the question to the query language and execute it to get a list of all annotated classes. An experienced developer can easily formulate this query, but a novice developer may

Copyright © by the paper's authors. Copying permitted for private and academic purposes.

Proceedings of the Seminar Series on Advanced Techniques and Tools for Software Evolution SATToSE 2018 (sattose.org).
04-06 July 2018, Athens, Greece.

lack the knowledge to turn the question into a query.

To overcome the difficulty of writing code in an unknown query language, developers search the web for code snippets, asking the question in a natural language and modifying the answer to the desired form of the tool [BDWK10]. Since these tools offer many features, they have a steep learning curve. Learning a new query language and supported queries of a tool is a time-consuming process and sometime the efforts required to learn the tool may be too high for the perceived benefits. Asking questions in a natural language is an easier way to overcome these obstacles.

The emergence of software assistants and the need to search beyond the web has led to an interest in natural language question answering systems. The user does not need to know the underlying formal language and can input a query in the NL. The prior example of the Moose query for deprecated classes illustrates the developer's query and its code representation in Smalltalk. We aim to present the developer with sample queries and also provide an interface to write new NL question, execute the corresponding query, and present the result.

There are various challenges involved in this project: (*i*) establish a way to search an NL question in the tool, (*ii*) present sample questions of software analysis scenarios, (*iii*) automatically translate a developer's NL question to the query language of the tool, (*iv*) for automatic translation process, prepare a dataset for Smalltalk analysis domain, and (*v*) support the developer if she uses different words to search for software entities than those actually in use *e.g.,* to find a function named `searchForThreat`, the developer guesses words like `findDanger` or `detectRisk`.

In this paper, we propose an idea to help developer to interact with software analysis tools without learning query language of the tool and reduce complexity to perform an analysis. We plan to develop a plugin for Moose where a developer can leverage the user interface provided by the tool with a feature to input her query in a natural language instead of a tool's query language. We propose to construct a corpus of common software analysis questions. We plan to automatically translate NL questions to the query language of the tool, execute the translated query, and present the results to the developer. In the following section, we describe several approaches used in the past to solve the problem of translation of NL queries to the query language of the tool. We discuss the available choices of translation techniques and propose an appropriate approach for this project.

2 Background

The general problem of translating natural language specifications into executable code has been around for more than fifty years now [Bob64]. There exist several works that tried to translate NL queries to query languages [LJK13, RGMF15, WGRG10]. The query language of the tool can be a database query language or PL. Recent advances in this area include the successful translation of natural language commands to database queries [LJK13]. The problem of translation increases when the target language is a general-purpose PL due to extensive syntax and expressiveness of the language.

We describe several approaches. Some of the approaches are based on the syntax structure of NL queries, like ASTs matching, some consider semantic of the language. Here, we analyze different approaches and determine which approach is suitable for our scenario.

2.1 Syntax based approaches

Syntax-based approaches try to syntactically match words in the natural language query to method or class parameters of the programming language. Using NLP tokenization and Part-Of-Speech (POS) tagging, a NL query can be divided into tokens and a tag can be assigned to the token. The tagged token can be mapped to code. Thus the NL query token "`classes`" can be mapped to "`allModelclasses`" of the target code due to the direct matching of the tag to an entity of the language, but it would fail to map a similar NL query presented like *"all classes that should no longer be used"* due to the limited tagset, and lack of information about the construction of different phrases.

Either we define tags for all possible syntax elements of the language or we can use syntactic structure information to gather more information about the NL query. Analyzing the syntactic structure of queries can give useful insight. Syntactic parsing is used to assign the syntactic structure. Syntactic parsing based code-generation models model code at the level of the parse tree or abstract syntax tree (AST) and can map to code on the basis of tree matching algorithms. However, these techniques fail to generate correct code for semantically equivalent NL queries due to similar words like *"no longer used"* and *"deprecated"*.

2.2 Semantic based approaches

Semantic-based approaches analyze an NL sentence from the point of view of its meaningful representation and the context in which the sentences or words are used. Semantic parsing transforms an NL sentence into a machine-interpretable meaning representation. It is inherently a more complicated task than syntactic parsing, since it requires the understanding of the meaning of words and finding relations among them. To understand the contextual similarity of words in a sentence, word embedding is used. Word embedding has proved to be a powerful approach to represent word relations. It embeds words in a high-dimensional vector space so that words that appear close in the source text are close in the final vector space. The

adoption of word embedding for our project enables, for example, similar vector representations of the words *"no longer used"* and *"deprecated"* implying a close semantic relationship.

In addition to these semantic relationships between words, the structure of a sentence also plays an important role. The NL queries *"Which deprecated methods are the client classes using?"* and *"Which client classes are using the deprecated methods?"* are semantically different queries on a sentence level, composed of the same words in a different order. The use of word embedding might not provide a correct result in this case. So, the structure and sequence of input queries need to be considered to translate to the code correctly. The output code structure of the PL diverges from the input structure of the NL query and target code must be structured and well-formed due to the PL constraints. These constraints necessitate the use of training models to analyze the sequence of words and formalize a mapping between sequence of words in the NL query to the sequence of words in the PL.

2.3 Sequence based approaches

To analyze the sequence and structure of a sentence, sequence based models are used. The n-gram is the most widely used sequence-based model to capture dependencies in sequences. The n-gram model assumes that words are generated sequentially, left-to-right and that the nth word can be predicted using the previous n-1 words. The consequence of capturing a short context is that the n-gram model cannot handle long-range dependencies.

Sequence-based code models have been superseded by deep recurrent neural network (RNN) models to outperform n-grams, and have proven effective to exploit the sequential structure on both the input and output side of query [LGH+16]. RNNs can model variable length of the text, including very long sentences, and can preserve the order of words. This approach achieves the state-of-the-art results on several semantic parsing tasks. But training a data-hungry neural model requires a large dataset and this is one of the significant challenges of our project. Datasets used in preceding neural-model training experiments are available for Python but not for Smalltalk. We plan to connect to various industrial and research communities that are using Moose to gather a dataset.

Recently, Ling *et al.* have proposed a novel neural architecture for code generation of high-level languages like Python and Java [LGH+16]. Other approaches based on neural networks have experimented further for generating general purpose PL code [YN17], [LLBR13]. Yin has incorporated the knowledge of the grammar into the architecture design to achieve better performance on code translation task and we found this approach suitable for our project [YN17]. It allows the strong underlying syntax of the PL to be captured. In the following section, we describe our plan to handle challenges mentioned for our project and about the various components of the neural model selected for our approach.

3 Approach

To present the capabilities of tools we have collected NL questions of software maintenance tasks [dAM08], software testing tasks [Koc16] and development tasks [SMDV06]. We have analyzed numerous case studies about the analysis tool usage in the industry [GÎ7]. These case studies helped us to gain insight into software analysis problems that are important to the user. On the basis of common developer questions and analysis of case studies, we plan to prepare a corpus of sample questions of common software analysis scenarios. Sample questions can be categorized according to the type of information a question is intended to extract *e.g.,* properties of the class, inter-class dependencies, intra-class dependencies, inheritance, string search. The developer would not be restricted to use these predefined questions. We propose a user interface to input an NL question in the tool. We will translate the NL question to Moose's query language *i.e.,*Smalltalk and execute translated code on the model (refer Figure 1). The result after execution will be presented back to the user for further analysis.

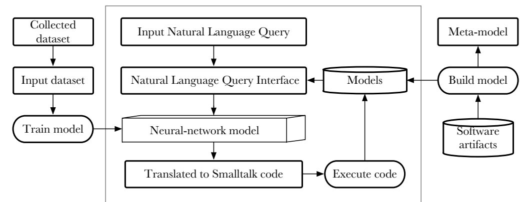

Figure 1: Software analysis in Moose using NLP

We have discussed various approaches to translate an NL query to Smalltalk code. Analysing available results from these approaches and challenges of our project, we concluded that we should use a neural network-based deep learning RNN technique. We plan to collect the data for our project domain from the various sources, prepare a dataset in the form of a pair of the NL query description and its target code snippet, and train the model (refer Figure 1).

We leverage the NLP technique suggested by Yin *et al.* to generate the AST of the NL query [YN17]. Once we obtain the syntax of the NL query, we can use deterministic generation tools to convert the AST into surface code [YN17]. We use a neural network-based encoder-decoder framework to generate the AST. The encoder computes a context-sensitive embedding of each word of an NL description using bidirectional long short-term mem-

ory (LSTM) network and creates vector representations of the NL query. The decoder uses generated vector representations that are fed to RNN to model the sequential generation process of an AST. The underlying syntax of the PL is encoded in the grammar model a priori. Thus the model is aware of the target PL grammar. Theoretically, a sufficiently large encoder-decoder model should be able to perform the machine translation perfectly. However, to encode all words and their dependencies in the arbitrary-length sentences, the vector should have enormous length. Such a model would require massive computational resources to train and to use. This problem can be solved with the attention mechanism; the decoder can use information from an arbitrary part of the encoded input sequence [BCB14].

Training a neural network model is the most important phase and directly related to the accuracy of the code-generation for the NL query. Preparing such a large dataset for the model is one of the significant challenges for our domain. We plan to conduct various studies and seek help from Smalltalk community to gather data similar to the datasets mentioned by Yin [YN17]. Considering the fact that the code has to be a well-defined program in the target syntax, we plan to use a hybrid approach of a syntactic-based and semantic-based neural network model. We are curious whether a neural network based approach can be used to generate executable and functionally coherent source code for Smalltalk.

Besides this translation, we will use heuristic information of the meta-model to enrich our query information. This will increases the accuracy of searches. In order to evaluate the effectiveness of the semantic analysis, we will analyze various metrics available for comparing the accuracy of semantic translations like BLEU [PRWZ02] and MEANT [Lo17]. As we are preparing set of common analysis queries and their corresponding queries in Moose, we will also use these queries to measure our translation results.

4 Related Work

Supporting developers to understand a software system has been a top priority task in software development. Wursch *et al.* described a system that is similar to our approach but while they have used tools from ontology engineering, we are proposing to use NLP techniques [WGRG10]. Recent NLP research is now increasingly focusing on the use of new deep learning methods. There have been attempts to design semantic parsers and syntactic parsers using deep learning methods to translate NL query to query language of the tool [YHPC17, LLBR13, YN17, LGH+16]. We want to leverage these techniques and propose to use them for generating code of Smalltalk from NL queries, so a novice developer can analyze a model easily.

5 Conclusions

Software analysis tools offer diverse features for analyzing software artifacts but also present multiple challenges. These tools demand a user to learn their query language to analyze a model which can be a burden for novice users. We outline an idea of a possible analysis workflow that will ovecome these obstacles. In this paper, we propose a plugin for Moose that will allow a user to input a natural language query and present the result. We will prepare a corpus of questions that are important for the developers and categorized them according to the type of information a question is intended to extract. We want to translate these questions to Smalltalk automatically but simultaneously we do not want to restrict developers to a set of predefined queries. We plan to explore the neural network-based approach to solve the mentioned challenges.

Acknowledgements

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project "Agile Software Analysis" (SNSF project No. 200020-162352, Jan 1, 2016 - Dec. 30, 2018).

References

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[BDWK10] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R. Klemmer. Example-centric programming: integrating web search into the development environment. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 513–522, New York, NY, USA, 2010. ACM.

[Bob64] Daniel G. Bobrow. Natural language input for a computer problem solving system. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1964.

[dAM08] Brian de Alwis and Gail C. Murphy. Answering conceptual queries with Ferret. In *Proceedings of the 30th International Conference on Software Engineering (ICSE)*, pages 21–30, New York, NY, USA, 2008. ACM.

[DGLD05] Stéphane Ducasse, Tudor Gîrba, Michele Lanza, and Serge Demeyer. Moose: a collaborative and extensible reengineering environment. In *Tools for Software Maintenance*

and Reengineering, RCOST / Software Technology Series, pages 55–71. Franco Angeli, Milano, 2005.

[EKRW02] Jürgen Ebert, Bernt Kullbach, Volker Riediger, and Andreas Winter. GUPRO — generic understanding of programs, an overview. Fachberichte Informatik 7–2002, Universität Koblenz-Landau, 2002.

[FBTG02] Rudolf Ferenc, Árpád Beszédes, Mikko Tarkiainen, and Tibor Gyimóthy. Columbus-reverse engineering tool and schema for c++. In *Software Maintenance, 2002. Proceedings. International Conference on*, pages 172–181. IEEE, 2002.

[GÎ17] Tudor Gîrba. Humane assessment by example. Technical report, feenk.com, 2017.

[Koc16] Pavneet Singh Kochhar. Mining testing questions on stack overflow. In *Proceedings of the 5th International Workshop on Software Mining*, SoftwareMining 2016, pages 32–38, New York, NY, USA, 2016. ACM.

[LGH+16] Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, Andrew Senior, Fumin Wang, and Phil Blunsom. Latent predictor networks for code generation. *arXiv preprint arXiv:1603.06744*, 2016.

[LJK13] Percy Liang, Michael I Jordan, and Dan Klein. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446, 2013.

[LLBR13] Tao Lei, Fan Long, Regina Barzilay, and Martin Rinard. From natural language specifications to program input parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1294–1303, 2013.

[Lo17] Chi-kiu Lo. Meant 2.0: Accurate semantic mt evaluation for any output language. In *Proceedings of the Second Conference on Machine Translation*, pages 589–597, 2017.

[MJS+00] Hausi A. Müller, Jens H. Janhke, Dennis B. Smith, Margaret-Anne Storey, Scott R. Tilley, and Kenny Wong. Reverse engineering: A roadmap. In A. Finkelstein, editor, *The Future of Software Engineering 2000*. ACM Press, 2000.

[PRWZ02] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[RGMF15] Mohammad Raza, Sumit Gulwani, and Natasa Milic-Frayling. Compositional program synthesis from natural language and examples. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 792–800. AAAI Press, 2015.

[RS04] Spencer Rugaber and Kurt Stirewalt. Model-driven reverse engineering. *IEEE software*, 21(4):45–53, 2004.

[SCH98] Susan Elliott Sim, Charles LA Clarke, and Richard C Holt. Archetypal source code searches: A survey of software developers and maintainers. In *Program Comprehension, 1998. IWPC'98. Proceedings., 6th International Workshop on*, pages 180–187. IEEE, 1998.

[SMDV06] Jonathan Sillito, Gail C. Murphy, and Kris De Volder. Questions programmers ask during software evolution tasks. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, SIGSOFT '06/FSE-14, pages 23–34, New York, NY, USA, 2006. ACM.

[TDD00] Sander Tichelaar, Stéphane Ducasse, and Serge Demeyer. FAMIX and XMI. In *Proceedings WCRE 2000 Workshop on Exchange Formats*, pages 296–296, Los Alamitos CA, November 2000. IEEE Computer Society Press.

[WGRG10] Michael Würsch, Giacomo Ghezzi, Gerald Reif, and Harald C Gall. Supporting developers with natural language queries. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 165–174. ACM, 2010.

[YHPC17] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *arXiv preprint arXiv:1708.02709*, 2017.

[YN17] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*, 2017.