AutoCom: Automatic Comment Generation for C Code

Zhikang Tian
Shenzhen Foreign Language School
China

Yuekang Li
Nanyang Technological University
Singapore

ABSTRACT

Code comments improve program comprehension and program maintenance. However, the lack of comments is a common problem in industry. It is time- and manpower-consuming to add comments for large code bases. Thus, it is desirable to develop techniques for automatic comment generation. Previous works for automatic comment generation use deep learning or machine learning techniques. These techniques require a large amount of training data which is often unavailable or hard to acquire. This paper proposes a light-weight approach called AutoCom for automatic comment generation. In AutoCom, we first analyze the source code to extract key information. Then, we use the extracted information to search and filter for appropriate text from a large programming Question and Answer (Q&A) site. Lastly, we use NLP techniques to convert the search result into code comments. In addition to the generated code comment, we also add predefined comments for library function usage in the source code. With AutoCom as the back-end, we built a web service which allows the user to upload source code and get it commented.

KEYWORDS

automated comment generation; documentation; program comprehension

1 INTRODUCTION

Comments play a very important role in code comprehension in program development and maintenance, especially after codes are created by programmers. As it is common for developers to shift between projects or departments, codes without comments would be hard to modify by programmers who take over. In the meantime, regularly commenting the codes would be distracting for programmers, as writing comments will break the train of thought of writing codes. Thus, the efficiency of the programmers might be brought down, and the codes written not fluent or even not functioning well. If the comments can be written automatically, it will save the programmers from writing comments and also help the successors in understanding previous codes. This thought leads to the birth of AutoCom, which is a program that can automatically match the codes with useful comments.

Existing automatic comment generation methods are not flawless: Most of them mainly focus on deep learning and machine learning. These techniques require a large amount of training data to fully function, but the data are hard to acquire. Thus, human labor is still not fully emancipated from the onerous process, because the large number of training data must be written by programmers based on their experience. Besides it also requires large amount of time to get these data. Secondly, those training data are complex, and it is hard to guarantee that no error has been made when writing. If any errors exist in the data, the result of commented codes will be wrong and useless. Thirdly, the large amount of new training data are continuously increasing, and they need to have a large database to save them, which requires money and space.

To address these issues, we proposed a new approach — AutoCom, a light-weight and full functioning tool in automatic C code comment generation. In AutoCom, we take down all the source code apart and extract out the key information of the source code. Then, we use the extracted key information to form the appropriate key words or filters and further establish a search on a large Question and Answer (Q&A) Site StackOverflow, which contains large number of source code and code description written by programmers. These Q&A work as the source to extract comments. After this, we analyze these sentences using Natural Language Processing (NLP) techniques, giving each sentence a weighting. We also convert the search results into useful comments. Lastly, we insert the comments into the source code. Additionally, we also add predefined comments for C Standard Library function usages in the source code, which requires fewer source code to be searched on StackOverflow. This novel design can save a lot of time when generating codes.

We implemented AutoCom as a Python-based commandline tool and wrapped it with a web-server to provide service: http://tianzhikang.pythonanywhere.com/. We also prepared an introduction video for AutoCom at: https://youtu.be/jxP389kFb7U.

2 METHODOLOGY & IMPLEMENTATION

In this section, we introduce details about the design of AutoCom. Figure 1 shows an overview of AutoCom's workflow. The input is C program source code and the output is the source code with comments added. The workflow of AutoCom has four steps:

❶ We need to extract key information from the source code. To do so, we first convert the convert the source code into abstract syntax tree (AST) to identify the different components. Then we extract function names and constant strings as individual pieces of key information from the AST.

❷ We conduct a fuzzy search on the Q&A site for each piece of extracted information. After searching, we merge all the searching results (questions on the website) and rank them according to the sum of vote up/down scores and number of answers. More up votes and answers lead to a higher rank, and then the searching results with ranks higher than a threshold (the threshold can be specified by the user) will be kept and used to generate comments. Algorithm 1 shows the how the search results are filtered and retained. Note that here we discard the results of the keywords with more than 30 search results. The rationale is that if a keyword produces too many search results, it means the keyword is too general to represent the features of the code. For example, the function name *"main"* is a bad keyword for searching with. Here, we set 30 as the threshold after some experiments.

DOI reference number: 10.18293/SEKE2021-202

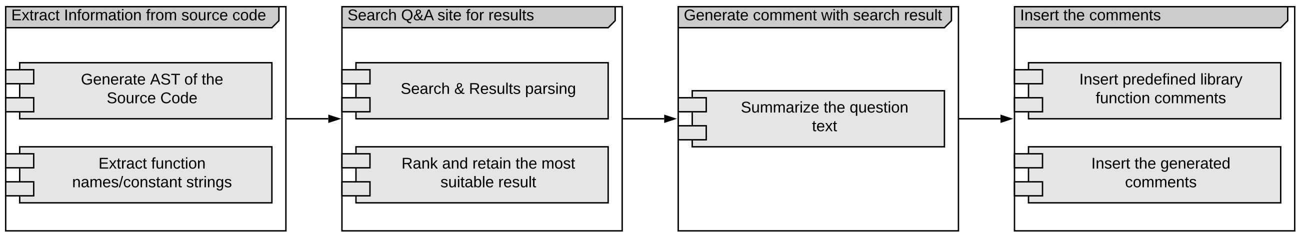
Extract Information from source code
Generate AST of the Source Code
Extract function names/constant strings
Search Q&A site for results
Search & Results parsing
Rank and retain the most suitable result
Generate comment with search result
Summarize the question text
Insert the comments
Insert predefined library function comments
Insert the generated comments

Figure 1: Overview of the workflow of AutoCom

❸ After the most suitable search results have been recognized and retained, we then apply text summary techniques to generate the summaries of the searched text. In AutoCom, we allow the users to choose either Latent Semantic Analysis (LSA) summarizer [8] or Edmundson Heuristic Method (EHM) summarizer [9]. For LSA, users only need to input the original text as well as a list of stop words to get the summary. For EHM, users need to provide two additional lists of bonus words and stigma words. In EHM, the additional words (cue phrases, keywords, title words) will increase the weight of a sentence while the stigma words shall decrease it. As for the additional words, we reuse the keywords to search the results. This means that sentence containing the keywords for searching are more likely to be included into the summary. As for the stigma words, we provide a list of words which are commonly used in the Q&A context of the website, such as *what*, *why*, *error*, *warn*, etc. This ensures that the commonly used sentences related to the question asking/answers or descriptions of the errors are excluded from the summary.
❹ Last but not least, the generated summaries are inserted back into the code as comments. The summary will serve as a comment for the entire code. During this process, we also add predefinedcomments for standard C library function calls. This is achieved by maintaining a dictionary of standard library functions and their descriptions and by adding the corresponding descriptions as comments for each occurrence of the library function call in the code.

The key logic of AutoCom is implemented with around 400 lines of Python code. We use Py-StackExchange [6] to query the StackOverflow API. We use Beautiful Soup [3] for parsing the Q&A webpage and extracting the text content. We use pycparser [7] to parse the C code and produce the abstract syntax tree of the input code. Lastly, we use nltk [5] and sumy [2] to summarize the text.

Algorithm 1: *Algorithm for Retaining Search Results*

```
input  : A set of keywords $K$, Top $n$ search results to keep
output : A list of search results for comment generation $L$
1 $L \leftarrow \emptyset$;
2 foreach $k \in K$ do
3     $R \leftarrow$ search($k$);
4     if $|R| > 30$ then
5         continue;
6     $R$.sort();
7     $L$.insert($R[0 : n]$);
8 return $L$;
```

Moreover, we wrap the key logic of AutoCom with a webserver based on Django [4]. In other words, we make AutoCom a web-service so that everyone can try and use it. AutoCom is currently hosted on PythonAnywhere [1]:http://tianzhikang.pythonanywhere.com/.

3 FUTURE WORK

In the future, we have three directions to improve the performance of AutoCom. First, we can improve the quality of information extraction from the source code. Currently we are using syntactic information like function names or constant strings as the keywords for searching. First, we plan to take a further step to extract some basic semantic understanding about the code. Second, we plan to apply finegrained analysis for the search result. Currently we are extracting only the text of the question. We can also extract the code fragments from the question if the question contains some. Then we can analyze the semantic of the extracted code fragment to see if it matches our original source code. If the code fragment matches our original source code, it means the question is suitable to generate comments. Third, we can use more advanced NLP techniques to generate comments. Currently, we are using traditional text summarization methods like LSA or EHM. In the future, we are planning to adopt more advanced text summarization techniques [10–12] in AutoCom.

REFERENCES

[1] Automatic comment addition, 2020.
[2] Automatic text summarization, 2020.
[3] Beautiful soup, 2020.
[4] django: the webframeworks for perfectionists with deadlines, 2020.
[5] Natural language toolkit, 2020.
[6] Py-stackexchange: An api wrapper for python, 2020.
[7] pycparser, 2020.
[8] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.
[9] H. P. Edmundson. New methods in automatic extracting. *J. ACM*, 16(2):264–285, April 1969.
[10] Yaser Keneshloo, Naren Ramakrishnan, and Chandan K. Reddy. Deep transfer reinforcement learning for text summarization. *CoRR*, abs/1810.06667, 2018.
[11] Abdullah Al Munzir, Md. Lutfor Rahman, Sheikh Abujar, Ohidujjaman, and Syed Akhter Hossain. Text analysis for bengali text summarization using deep learning. In *10th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2019, Kanpur, India, July 6-8, 2019*, pages 1–6. IEEE, 2019.
[12] Shengli Song, Haitao Huang, and Tongxiao Ruan. Abstractive text summarization using LSTM-CNN based deep learning. *Multim. Tools Appl.*, 78(1):857–875, 2019.