Original

# summary

Dejun Qian

electronseu@gmail.com

This paper [1] addresses the problem of automating the verification process of software product based on the Application Programming Interface (API) documents written in natural language. Software reuse gives software developers the ability to build larger software products with higher quality in shorter time. The reused software library is usually provided to the developers in the forms of source code or binary executable, along with the document explaining to how to use itself. Reading and understanding the document is important, or even the only way when no source code is provided, to correctly use the provided functionalities. However, the developers are easily overlook some documents, and misuse some of the functions, which will leads to potential inconsistency between the software product and the documents. Many verification tools are developed to detect the inconsistence, however these tools only accept formal specifications as their input. They can't understand documents written in natural language. This paper presents an approach to fill the gap between what is needed by the existing verification tools and what the document can provide.

This paper developed a technique to get the code contract from the library document. The technique developed in this paper can read the API document, and output the formal specification in the form of code contract which can be accepted by several verification tools. Roughly, the technique is composed of two parts: contract sentence selection and code contract generation. These two parts can be further divided into five steps: parsing, pre-processing, text analysis, post-processing and code

Vejun 4

contract generation. In the parsing step, the parser process the API documents and extracts intermediate contents of the method description, such as, summary description, argument description, return description, exception description and remark description. In the pre-processing step, the pre-processor takes the intermediate content, and perform three categories of processing: meta-data augmentation, noun boosting and programming constructs and jargon handling. These first two steps are critical, as they can improve the results of the third step a lot. The third step is text analysis. In this step, this paper developed a text analysis engine framework, and adopting an existing POS tagger to annotate POS tags for the input sentences. After that, they use an NLP technique, called shallow parser to classify the sentences into semantic templates, based on the lexical tokens generated by the POS tagger. The shallow parser also generates FOL expressions after it has done the classification. In the post-processing step, the paper does three types of semantic analysis: removing irrelevant modifiers in predicates, classifying predicates into a semantic class based on domain dictionaries, and augmenting expressions. In the final step, by using predefined mapping from predicates to code contracts, the code contract generator translates the FOL into code contracts.

The author wrote a prototype to demonstrate the technique developed, and did three experiments to evaluate the effectiveness of the approach. The experiment results show that this approach gives 92% precision and 92% recall towards the contract sentence detection, and achieves 82% of accuracy of code contract generation.

## References

[1] Pandita, R., Xiao, X., Zhong, H., Xie, T., Oney, S., and Paradkar, A. Inferring method specifications from natural language api descriptions. In *Proceedings of the 2012 International Conference on Software Engineering* (Piscataway, NJ, USA, 2012), ICSE 2012, IEEE Press, pp. 815–825.

Peer review by Disha

# summary

Dejun Qian

electronseu@gmail.com

This paper [1] addresses the problem of automating the verification process of software product based on the Application Programming Interface (API) documents written in natural language. Software reuse gives software developers the ability to build larger software products with higher quality in shorter time. The reused software library is usually provided to the developers in the forms of source code or binary executable, along with the document explaining to how to use itself. Reading and understanding the document is important, or even the only way when no source code is provided, to correctly use the provided functionalities. However, the developers are easily overlook some documents, and misuse some of the functions, which will leads to potential inconsistency between the software product and the documents. Many verification tools are developed to detect the inconsistence, however these tools only accept formal specifications as their input. They can't understand documents written in natural language. This paper presents an approach to fill the gap between what is needed by the existing verification tools and what the document can provide.

This paper developed a technique to get the code contract from the library document. The technique developed in this paper can read the API document, and output the formal specification in the form of code contract which can be accepted by several verification tools. Roughly, the technique is composed of two parts: contract sentence selection and code contract generation. These two parts can be further divided into five steps: parsing, pre-processing, text analysis, post-processing and code

contract generation. In the parsing step, the parser process the API documents and extracts intermediate contents of the method description, such as, summary description, argument description, return description, exception description and remark description. In the pre-processing step, the pre-processor takes the intermediate content, and perform three categories of processing: meta-data augmentation, noun boosting and programming constructs and jargon handling. These first two steps are critical, as they can improve the results of the third step a lot. The third step is text analysis. In this step, this paper developed a text analysis engine framework, and adopting an existing POS tagger to annotate POS tags for the input sentences. After that, they use an NLP technique, called shallow parser to classify the sentences into semantic templates, based on the lexical tokens generated by the POS tagger. The shallow parser also generates FOL expressions after it has done the classification. In the post-processing step, the paper does three types of semantic analysis: removing irrelevant modifiers in predicates, classifying predicates into a semantic class based on domain dictionaries, and augmenting expressions. In the final step, by using predefined mapping from predicates to code contracts, the code contract generator translates the FOL into code contracts.

The author wrote a prototype to demonstrate the technique developed, and did three experiments to evaluate the effectiveness of the approach. The experiment results show that this approach gives 92% precision and 92% recall towards the contract sentence detection, and achieves 82% of accuracy of code contract generation.

## References

[1] Pandita, R., Xiao, X., Zhong, H., Xie, T., Oney, S., and Paradkar, A. Inferring method specifications from natural language api descriptions. In *Proceedings of the 2012 International Conference on Software Engineering* (Piscataway, NJ, USA, 2012), ICSE 2012, IEEE Press, pp. 815–825.

PSU CS 569/669 Scholarship Skills, Winter 2013

# One-page Summary — Peer Review

First study the grading rubric for the one-page summary paper, which is outlined below.

1. Conciseness (20)
2. Clarity and simplicity (20)
3. Core Rules (24)
    a. Use active voice
    b. Put key ideas in lead positions
    c. Don't make unsubstantiated statements
    d. Use a consistent lexical set
    e. Define terms when first used
    f. Avoid single-sentence paragraphs
4. Mechanics (15)
5. Grammar (16)
6. Bibliography Format (5)

For each part of the rubric, either

- Find a part of the paper that satisfies the rubric. Draw a box around it and label it with the rubric number.
- Find a part of the paper that violates a rubric point. Draw a red box around it, label it with the rubric number, and explain why you think it fails to measure up.

Comment on the structure of the summary. Was it organized in a way that made it the summary easy to read? What you suggest about restructuring the summary? Should parts of the text be:

- Moved?
- Reorganized into bulleted lists?
- Elided?
- Rewritten? ⟶ The explanation of the third step
- Strengthened by adding additional evidence or discussion?
- Clarified by removing extraneous or irrelevant material?

List the 3 best things about the summary. Be specific: explain why you found these things useful.

1. The introduction paragraph is simple and easy to understand. It is helpful for a person new to the area.

2. The experiment portion is concise, gives good overview of results achieved

3. The flow of the document is good, makes it easy to relate to the topic

In your opinion, what items constitute the weakest part of the summary? Be specific, and suggest 2 ways that the author could strengthen these items.

1. Not defining terms when they are used first

2. Grammar (singular vs plural mainly)

Here are some other things that you might look for and comment upon.

- Good use of quotations.
- A bibliography.
- Correct spelling, punctuation, and grammar.
- Original and interesting to the reader. Does the author of the summary tell the reader what she thinks, rather than just repeating the facts?

Back to the class web-page.

Review from the grader

# summary ← title?

Dejun Qian

electronseu@gmail.com

↑ "This" means this paper ↓

This paper [1] addresses the problem of automating the verification process of software product based on the Application Programming Interface (API) documents written in natural language. Software reuse gives software developers the ability to build larger software products with higher quality in shorter time. The reused software library is usually provided to the developers in the forms of source code or binary executable, along with the document explaining to how to use itself. Reading and understanding the document is important, or even the only way when no source code is provided, to correctly use the provided functionalities. However, the developers are easily overlook some documents, and misuse some of the functions, which will leads to potential inconsistency between the software product and the documents. Many verification tools are developed to detect the inconsistence, however these tools only accept formal specifications as their input. They can't understand documents written in natural language. This paper presents an approach to fill the gap between what is needed by the existing verification tools and what the document can provide.

This paper developed a technique to get the code contract from the library document. The technique developed in this paper can read the API document, and output the formal specification in the form of code contract which can be accepted by several verification tools. Roughly, the technique is composed of two parts: contract sentence selection and code contract generation. These two parts can be further divided into five steps: parsing, pre-processing, text analysis, post-processing and code

or "a"? ← the doc? awk awk w.c. y (?) cannot w.c. w.c. The paper could not do this. w.c. w.c.

Do NOT double space after periods.



awk — wierd.
w.c. — word choice

contract generation. In the parsing step, the parser process the API documents and extracts intermediate contents of the method description, such as summary description, argument description, return description, exception description and remark description. In the pre-processing step, the pre-processor takes the intermediate content, and perform three categories of processing: meta-data augmentation, noun boosting and programming constructs and jargon handling. These first two steps are critical as they can improve the results of the third step a lot. The third step is text analysis. In this step, this paper developed a text analysis engine framework, and adopting an existing POS tagger to annotate POS tags for the input sentences. After that, they use an NLP technique called shallow parser to classify the sentences into semantic templates, based on the lexical tokens generated by the POS tagger. The shallow parser also generates FOL expressions after it has done the classification. In the post-processing step, the paper does three types of semantic analysis: removing irrelevant modifiers in predicates, classifying predicates into a semantic class based on domain dictionaries, and augmenting expressions. In the final step, by using predefined mapping from predicates to code contracts, the code contract generator translates the FOL into code contracts.

The author wrote a prototype to demonstrate the technique developed, and did three experiments to evaluate the effectiveness of the approach. The experiment results show that this approach gives 92% precision and 92% recall towards the contract sentence detection, and achieves 82% of accuracy of code contract generation.

## References

[1] Pandita, R., Xiao, X., Zhong, H., Xie, T., Oney, S., and Paradkar, A. Inferring method specifications from natural language api descriptions. In *Proceedings of the 2012 International Conference on Software Engineering* (Piscataway, NJ, USA, 2012), ICSE 2012, IEEE Press, pp. 815–825.

# Improvements

**Dejun Qian**
electronseu@gmail.com

1. changed the title from "summary" to "Summary of 'Inferring Method Specifications from Natural Language API Descriptions'"
2. changed "This paper" to "Pandita et al." or "the authors"
3. changed one complex sentences into simple sentences
4. defined three terms
5. resolved the ambiguity of two sentences
6. changed one sentence into active voice
7. corrected several grammar errors

Revised

# Summary of "Inferring Method Specifications from Natural Language API Descriptions"

Dejun Qian

electronseu@gmail.com

Pandita et al. [1] address the problem of automating the verification process for software product, using the Application Programming Interface (API) documents written in natural language as the baseline. Software reuse gives software developers the ability to build larger software products with higher quality in shorter time. The reused software library is usually provided to the developers in the form of source code or binary executable, along with some documents explaining how to use the library. Reading and understanding the documents is important to correctly use the provided functionalities. It is even the only way when no source code is available. However, the developers easily overlook some documents, and misuse some of the functions, which may lead to potential inconsistency between the software product and the documents. Many verification tools are developed to detect the inconsistency, however these tools only accept formal specifications as their input. They cannot understand the documents written in natural language. Pandita et al. [1] present an approach to fill the gap between what the existing verification tools need and what the document can provide.

The authors develop a technique to get the code contract from the library documents. The technique can read the API documents, and output the formal specification in the form of code contract which can be accepted by several verification tools. The technique is composed of two parts: contract sentence selection and code contract generation. These two parts can be further divided into five steps: parsing, pre-processing, text anal-

ysis, post-processing and code contract generation. In the parsing step, the parser processes the API documents and extracts intermediate contents of the method description, such as summary description, argument description, return description, exception description and remark description. In the pre-processing step, the pre-processor takes the intermediate content, and performs three categories of processing: meta-data augmentation, noun boosting and programming constructs and jargon handling. These first two steps are critical as they can improve the results of the text-analysis step. In the text-analysis step, the authors develop a text analysis engine framework, and adopt an existing Parts-Of-Speech (POS) tagger to annotate POS tags for the input sentences. Then, they use an Natural Language Processing (NLP) technique known shallow parser to classify the sentences into semantic templates, based on the lexical tokens generated by the POS tagger. The shallow parser also generates First-Order Logic (FOL) expressions after it has done the classification. In the post-processing step, the paper does three types of semantic analysis: removing irrelevant modifiers in predicates, classifying predicates into a semantic class based on domain dictionaries, and augmenting expressions. In the final step, by using predefined mapping from predicates to code contracts, the code contract generator translates the FOL into code contracts.

The author designed a prototype to demonstrate the technique developed, and conducted three experiments to evaluate the effectiveness of the technique. The experiment results show that this technique gives 92% precision and 92% recall toward the contract sentence detection, and achieves 82% of accuracy of code contract generation.

## References

[1] PANDITA, R., XIAO, X., ZHONG, H., XIE, T., ONEY, S., AND PARADKAR, A. Inferring method specifications from natural language api descriptions. In *Proceedings of the 2012 International Conference on Software Engineering* (Piscataway, NJ, USA, 2012), ICSE 2012, IEEE Press, pp. 815–825.