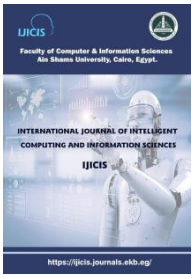
IJICIS, Vol.22, No.4, 1-12
DOI: 10.21608/ijicis.2022.117905.1160

International Journal of Intelligent Computing and Information Sciences

https://ijicis.journals.ekb.eg/



Source code generation-based on NLP and ontology

Anas Aloklah*
Faculty of Computer and Information Sciences, Ain Shams University
anas.hameed@cis.asu.edu.eg

Walaa Gad
Faculty of Computer and Information Sciences, Ain Shams University.
walaagad@cis.asu.edu.eg

Mostafa Aref
Faculty of Computer and Information Sciences, Ain Shams University.
mostafa.aref@cis.asu.edu.eg

Abdel Badeeh Salem
Faculty of Computer and Information Sciences, Ain Shams University.
absalem@cis.asu.edu.eg

Received 2022-01-23; Revised 2022-07-29; Accepted 2022-08-01

Abstract: *Generating source code is necessary especially as software evolves in complexity and demand. Finding a mechanism to generate the source code according to the requirements will save time for developers at the stage of development of the software. In this paper, a mechanism is proposed to generate the source code based on the database schema and user requirements (user story). This model contains three layers: The first layer is to analyze each of the database schema, extract the relationships between the tables, determine the meanings of the fields and analyze the user's story to find the functions performed by each role of the software users. The second layer is deducing new functions based on what was mentioned in the first layer and extracting the knowledge that contains the solutions to the problems that are inferred. The knowledge bases used are WordNet and Backend Ontology built from scratch. In the third Layer, the solutions are converted to source code based on templates extracted from the knowledge and configured, that is applied to the templates. The model showed success in generating the source code, generating PHP source code for a site that is tested and generated seventy percent of what was required to be written by programmers.*

Keywords: *Automatic Programming, source code generation, Natural Language Processing, Ontological Engineering, knowledge Engineering*

* Corresponding author: Anas Aloklah
Faculty of Computer and Information Sciences, Ain Shams University
E-mail address: anas.hameed@cis.asu.edu.eg

1. Introduction

In the software development cycle [1], the development process is one of the most important processes, as it writes the source code that achieves the requirements that were set in the plan and design. Therefore, it takes a long time. Experienced developers take less time to develop because of the recurrence of the problem. As for the less experienced developers, they resort to using old source code that they wrote and uses it as a template to modify it according to the new requirement. Therefore, there is a need for an automated way to generate the source code as per the requirement. Moreover, there is no standardized approach or mechanism for generating the source code as this varies with the purpose of the source code and the different inputs. In generating source code from pseudocode, Deep Learning (DL) [2], Statistical Machine Translation (SMT) [3] or Neural Machine Translation (NMT) [4] are used for this purpose [5-8]. The Neural Machine Translation (NMT) is more efficient than SMT. In addition, the SMT consumes time in the training process [9,10]. The most popular model in the NMT uses Recurrent Neural Network (RNN) [6,7,8] for Translation. The main problem in RNN is changing weights to very small value in the training process which is called vanishing gradient [11] or changing weights to have the large value which is called exploding gradient. Thus, using the Long Short-Term Memory (LSTM) solves this problem [12]. Recently, a Machine Translation Model (MTM) based on Deep Learning (DL) is proposed by Google [13]. The main core of the model is based on the self-attention layers. The DL is used for generating source code from image of sketch design [14,15]. Both models use the Convention Neural Network (CNN) as the main unite of model design, but different in general model and method. The Knowledge Base (KB) is a good solution for source code generation when the input is complex because the result is related to many rules with the same input. The KB can be updated with any error in a result to correcting this error. The develop the model based on KB to generate source code from user request [16,17]. The Natural Language Processing NLP uses for source code generation when the input and result are subject to grammar. The uses NLP to convert user story to prolog and ontology languages [18]. In this paper, a novel model for source code generation based on NLP and KB is proposed. The proposed model has two inputs: user story and databased schema. The user story is a text that describes the user requirements. The proposed model consists of three layers: analyzer, reasoner, and convertor. The analyzer analyzes the inputs to extract the relations of databased schema, sentence meaning and rules. Reasoner uses the output of analyzer to find solutions of problems detected in inputs. The convertor converts the solutions from reasoner to source code. The main achievement of the proposed model is to extract the features of inputs and taking into consideration user inputs.
This paper is organized as follows: section 2 introduces a literature review; section 3 presents the proposed model and section 4 shows experimental results and finally section 5 is the conclusions.

2. literature review

In the translation of a language into other languages using the Machine Translation (MT), there are three approaches: Rule-Based Machine Translation (RBMT), SMT and NMT. In [5], authors use the Predictor Networks Based (PNB) on DL for generating source code from pseudocode. This model works as sentence to sentence and uses the C2W [19] model to encode the input tokens and uses the bidirectional LSTM (BiLSTM) to build words in the text fields. This model consists of three types of Predictor Networks (PN): firstly, character generation to predict character in the training data and uses

Softmax function to predict character in output layer. Secondly, copy singular field to predict singular field; such as, a type of sentence in dataset. Thirdly, copy text field to predict the words in text field using RNN to achieve this object. The decoding phase uses a stack-based decoder with beam search. In [6], the NMT is used as sentence to tree by applying the Abstract Syntax Trees (ASTs) in source code. The model uses the Abstract Syntax Description Language (ASDL) [20] framework for describing data. The architecture of model consists of encoder and decoder. The encoder uses the BiLSTM for input sentence. The decoder includes four classes of modules: Firstly, the composite type modules for detecting the rule of sentence; such as, if statement, while, for, return, etc. This module used a feed forward network and SoftMax in last layer. Secondly, the constructor module updated vertical LSTM states of rule detect and compute the next field of rule using feed-forwards network and vertical LSTM. Thirdly, the constructor field module processes the children of rule detected. Fourthly, primitive type modules process the value of the role detect using vertical LSTM and SoftMax. In [7], authors update the model in [6] that uses one grammar module in decoder, APPLYRULE[r] and GENTOKE[v]. The APPLYRULE[r] produces the sentence rules. The GENTOKE[v] puts the value (v) in the node tree by a token word and uses the DNN as a connection between encoder and decoder. The authors in [8], apply the retrieval mechanism [21] in [7] work. The retrieval mechanism consists of four steps: firstly, retrieve M sentence from training dataset is the most similar to the input. Secondly, translating the retrieve input and extracting the n-gram action subtrees corresponding to the retrieved input. Thirdly, change the subtrees by replacing the words of retrieve input with corresponding input. Finally, each decoding changes the weights to increase the probability of subtrees. Furthermore, The MT is used to convert the source code to pseudocode. In [22,23], RBMT approach is used to translate source code to Pseudocode the RBMT and SMT are less in performance than of NMT [9,10]. Thus, the NMT is used with different methodology to convert the source code to pseudocode [24,25]. Therefore. Transformer [13, 26] is adapted in MT to convert the source code to pseudocode. In [14], the Computer Vision (CV) and Region Based Convolutional Neural Networks (R-CNN) are used to generate source code from hand draw image sketch. The CV is used to edge-merged assembling, slope filtering and noise removing. The R-CNN is used for detecting a GUI Object. In [15], CNN and LSTM are adapted to generate source code of android , iOS, and web UI from image UI sketch. In [18], authors extract conception model from user story and convert to prolog or ontology language by using the NLP. The user story is set of sentences every sentence has standard predefined format [27]. The standard format sentence content of tree part. Firstly, the rule: who uses the function. Secondly, the meaning: what is the function of the rule. Thirdly, the end: why do we use this function as it is optional. Each of the three parts have indicator to know the part. Figuer.1, shows an example for sentence of user story. The "As a" is an indicator of role "Visitor", "I am able to" is an indicator of function " use the contact form " and "so that" is indicator of the last part " I can contact the administrator".

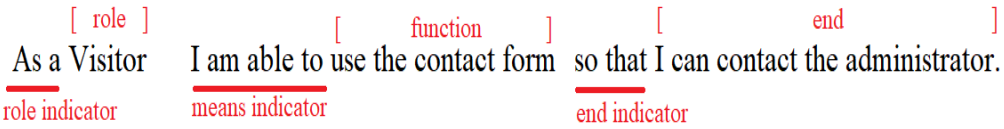


Figure. 1: An example of user story sentence

There are other indicators such as "As an", "As a", "As" for role indictor , "I 'm able to", "I'm able to", "I am able to", "I want to", "I want to be able to", "I wish to", "I can", "I would like to" for means indictor and "So that", "In order to", "So", "Because of" for end indictor.
In [16], authors proposed a design model for generating source code of web servers' data retrieval from user request using semantic web defined in ontology. The model depends on three components: specification, configuration, and template. The specification extracts the feature from the user request. The template extracts the program code templates from prototype. The configuration detects and maps between the specification feature extracted and template.

3. Proposed model

The proposed model generates source code form user's story and database schema. The proposed has three layers as shown in Figuer.2. The three layers are analyzer, find solution (reasoner) and convertor.

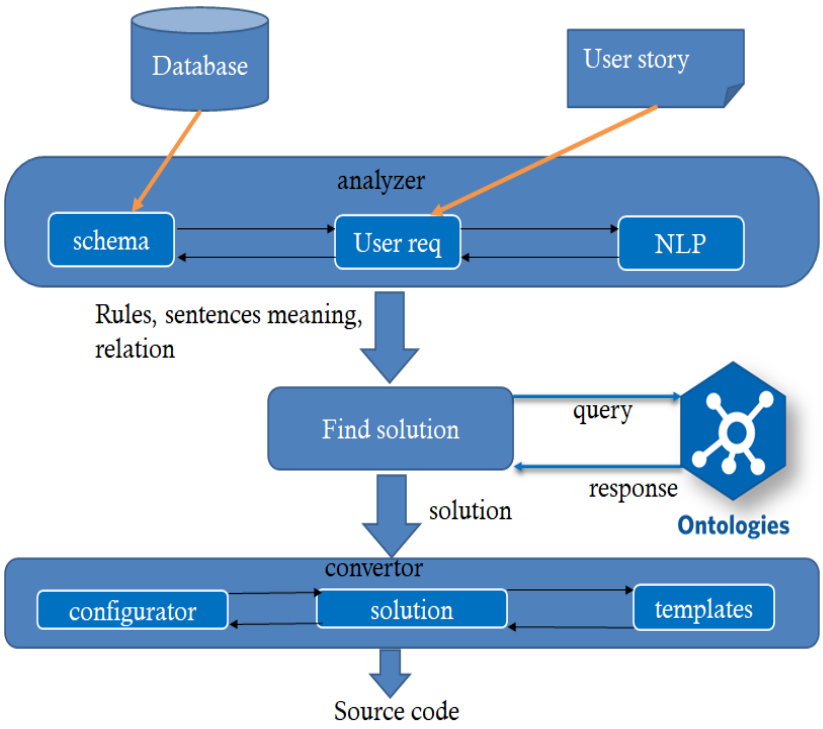


Figure. 2: The proposed model architecture for source code generation

3.1. Analyzer layer

The analyzer layer consists of

- Schema
- User requirements
- NLP

Schema component extracts the schema of database, and the relation between the tables of database. The extraction of schema by set SQL commands, then find the relation between the table by search each column in a table so that the column name is similar as the name of the table. Use the semantic similarity [28] and WordNet [29] to find the similarity between column name and table name. Figure.3 presents the algorithm of relation extraction.

Algorithm 1 relation extraction

```
1  Initialize relation set
2  Foreach table in tables list
3      Get columns list from table
4      Foreach column in columns list
5          Foreach table2 in tables list
6              If table2 name != table name
7                  If column name semantic similar table2 name
8                      Inset into relation set (table2 name, table name, column name)
9  Output relation set
```

Figure. 3: The algorithm of relation extraction

The user requirements component extracts the roles and functions of each role from the user story by using the modified algorithm in [18]. The algorithm uses the WordNet [29] to identify the type of tokens: NOUN, PROPN, VERB, ..etc. The indicators are used to find the role, means, and end in the user story sentence. there is a set for roles each new role inserts into the roles set. the means part explanation to detect the function name and the target of the function. Then the information about the means part (function name and the target of the function) is added to its role in the set of roles. the functions that do not have a target after explanation the means part that means this function they belong to the system. The end part is ignored because it is not important in generating the source code. Moreover, the user requirements component extracts the rules when the target of a function affects the table of the database, and this effect of the table is related to another table.

3.2. Find solution layer

This layer expresses the solution model as content list of solutions, list of commands and the schema. The solutions are extracted from executes rules which are using the backend ontology [30]. The backend ontology is ontology modeling the program languages, SQL and framework and content solutions for backend web domain. The commands extracted with same phase extract solutions. The commands have commend for system; such as, make file, copy file, executes function in system,…etc.

3.3. Convertor layer

The convertor layer converts the solution model to source code. The convertor layer consists of the following components:

- Solution
- Template
- Configurator

The solution component executes commands in solution model if the commend does not affect another solution. Moreover, move the solutions list into dictionary, then process the solution by request the template component for the template of solution if the solution has template. Then, send the template of solution to configurator to process the template. It also combines the results of the Configurator component.

The template component searches for template code for solution. Figure.4 presents the template of php class model. In line 2 the OPTIONAL means what in brackets is related with conditions, it means that if this class model of the table has a sub table that is included in template. The * means that this optional may be repeated between 0 to N. The #SubTable replaced with the class name of sub table. In line 3 the #Class_name replaced with the class name. In lines 6 and 8 the @ means, this is a subproblem must get a solution of then and replace the @problem_name by sub solution. All actions in the template are processed in the Configuration component.

```
1   <?php
2   OPTINAL*(include("#SubTable"."php"))
3   class #Class_name OPTINAL(extends Model)
4       {
5          //Properties
6          @php_model_Properties_list
7          //methods
8         @php_model_methods_list
9       }
10  ?>
```

Figure. 4: The template of php class model

The configurator component processes the template of code and request of the sub solution from solution component and replacing the #,@ , OPTINAL tags with what is needed. Then send the source after ending the process of template to solution component.

4. Experimental Results

The proposed model is evaluated using different database schema, user stories and type of solution. This section presents simple case content database that has four tables, user story and the type of solution which is Model View Controller (MVC) architecture. The backend ontology has solution of model and controller. The view solution does not implement yet. Figure 5 presents the simple E-commerce database schema. The table in schema user, category, product, and order. Figure 6 shows an example of user stories.

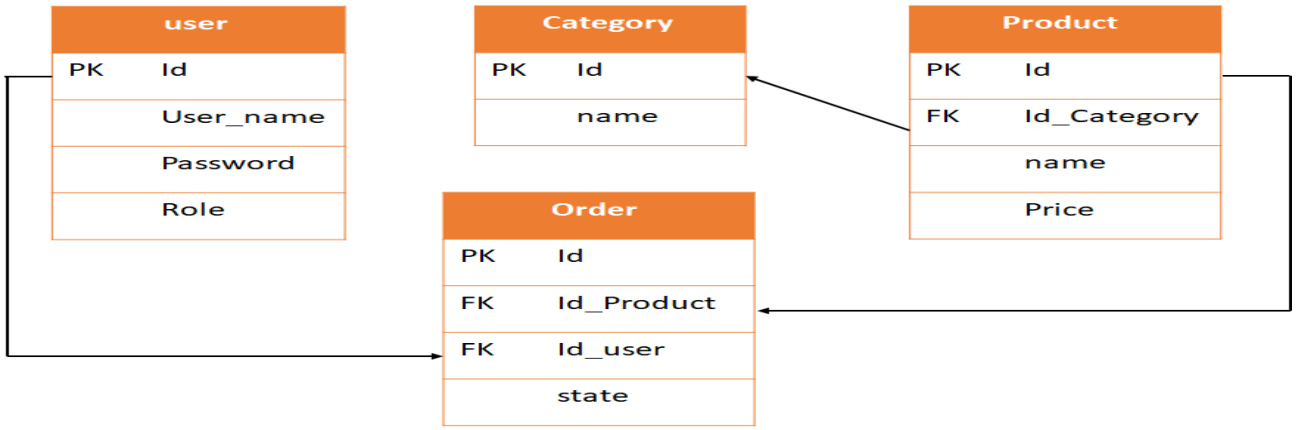


Figure. 5: Simple E-commerce database schema.

```
As a Administrator, I can add a new user.
As a Administrator, I can add a new product.
As a Administrator, I can add a new category.
As a Administrator, I can change my account password.
As a Administrator, I can update a user.
As a Administrator, I can update a product.
As a Administrator, I can update a category.
As a Administrator, I can update a order.
As a Administrator, I can delete a user.
As a Administrator, I can delete a product.
As a Administrator, I can delete a category.
As a Administrator, I can delete a order.
As a Administrator, I am able to log out.
As a Visitor, I can sign up.
As a Visitor, I can log in.
As a User, I can add a new order.
As a User, I can delete a order.
As a User, I am able to log out.
```

Figure 6: an example of user stories.

The product related to category by foreign key " Id_category" which means that when deleting a row from category, all the rows of the product have the same "id" in "Id_category" and will be deleted. This action is shown in Table 1, which presents the source code of the class model category and product.

Table 1 the result of category and product class model

Category.php	Product.php

<?php
include(Product.php)
class Category
{
//Properties
public $id;
public $name;
…
⋮
function delete_Category($id)
{
$servername = "localhost";
$username = "root";
$password = "2w3e4r5t6y7";
$dbname = "eCommerce ";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
die("Connection failed: " . $conn->connect_error);
}
$sql = "delete from Category WHERE `Category`.`id` = $id";
$product = new Product;
$product->delete_Product_by_id_Category($id);
$result = $conn->query($sql);
return $result;
}
}
…
⋮
?>
	<?php
	include(order.php)
	class Product
	{
	//Properties
	public $id;
	public $id_ Category;
	…
	⋮
	function delete_Product_by_id_Category ($id_ Category)
	{
	$servername = "localhost";
	$username = "root";
	$password = "2w3e4r5t6y7";
	$dbname = " eCommerce";
	// Create connection
	$conn = new mysqli($servername, $username, $password, $dbname);
	// Check connection
	if ($conn->connect_error) {
	die("Connection failed: " . $conn->connect_error);
	}
	$sql = "delete from Product WHERE ` Product `.`id_ Category ` = $id";
	$order = new Order;
	$order ->delete_Order_by_id_Category($id);
	$result = $conn->query($sql);
	return $result;
	}
	}
	…
	⋮
	?>

Category php imports the class Product and creates object, then calls delete_Product_by_id_Category method to delete rows with same id. Product. php generates the delete_Product_by_id_Category

method. Table 2 shows the number of lines for the generated source code file. Table 3 describes the generated source code requirements from the source code from system.

Table 2 number of lines code of each file generated

File code	Number of lines
User.php	314
Category.php	151
Product.php	206
Order.php	214

Table 3 what are the code generated and not generated of requirements

Requirement	generated
Create user class model for user table	Yes
Create product class model for user product	Yes
Create category class model for user category	Yes
Create order class model for user order	Yes
Create add user function in user class model	Yes
Create update user function in user class model	Yes
Create delete user function in user class model	Yes
Create login and logout functions in user class model	Yes
Create change password function in user class model	Yes
Create add product function in product class model	Yes
Create update product function in product class model	Yes
Create delete product function in product class model	Yes
Create add category function in category class model	Yes
Create update category function in category class model	Yes
Create delete category function in category class model	Yes
Create add order function in order class model	Yes
Create update order function in order class model	Yes
Create delete order function in order class model	Yes
Create state of order function	No
Create Cart class	No
Create function add and delete item into Cart	No
Create transportation cost function	No
Create tax cost function	No
Create total cost function	No
Create payment function	No

Equation 1 calculates the precent of source code generation

$$generated\ percent = \frac{\sum Requirement\ \ is\ generated}{total\ of\ Requirement} \times 100 \quad (1)$$

the number of experiments is related by backend ontology which with any addition or updates in backend ontology. the source code generation is tested for this update. the total of experiments more than 150 experiments.

5. Conclusions and Future Work

In this paper, a novel source code generation model is proposed for automatic source code generation from the database schema and user story. The proposed model consists of three layers: analyzer, find solution (reasoner) and convertor. In the analyzer layer, database schema and user story are analyzed to extract relation, sentence means and rules. The find solution layer finds the solution using Backend Ontology and makes the solution model. The last layer converts the solution model to source code by the cooperation of the three components, the solution, the configurator, and the template. The proposed model is evaluated manually because there is not automatic method to evaluate. The result generated by the knowledge base, not by dataset. The experimental results are promising because the model generates about 70% of what we want of the source code.

References

1. N. Al-Saiyd, Source code comprehension analysis in software maintenance, In: The 2nd International Conference on Computer and Communication Systems (ICCCS), 2017, p.1-5.
2. T. Iqbal, S. Qureshi, The survey: Text generation models in deep learning, Journal of King Saud University - Computer and Information Sciences, 2020
3. A. R. Babhulgaonkar, S. V. Bharad, Statistical machine translation, In: 1st International Conference on Intelligent Systems and Information Management (ICISIM), 2017 , p.62-67.
4. P. Koehn, Neural Machine Translation , Cambridge University Press, Online publication date:May 2020, Print publication year:2020, ISBN:9781108608480, DOI:https://doi.org/10.1017/9781108608480.
5. W. Ling, P Blunsom, E. Grefenstette, K. Moritz Hermann, T. Kočiský, F. Wang, A Senior, Latent Predictor Networks for Code Generation, In: 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2016, p.599–609.
6. M. Rabinovich, M. Stern, D. Klein, Abstract syntax networks for code generation and semantic parsing. In: 55th Annual Meeting of the Association for Computational Linguistics (ACL),2017 , p.1139–1149.
7. P. Yin and G. Neubig, A syntactic neural model for general-purpose code generation. In: 55th Annual Meeting of the Association for Computational Linguistics (ACL),2017, p. 440–450.
8. S. A. Hayati, R. Olivier, P. Avvaru, P. Yin, A. Tomasic, G. Neubig, Retrieval-Based Neural Code Generation, In: Conference on Empirical Methods in Natural Language Processing,2018, p.925–930.
9. R. Sennrich, B. Zhang, Revisiting Low-Resource Neural Machine Translation: A Case Study, In: the 57th Conference of the Association for Computational Linguistics, 2019 , p.211–221.

10. S. K. Mahata, S. Mandal, D. Das, S. Bandyopadhyay, SMT vs NMT: A Comparison over Hindi & Bengali Simple Sentences,2018 , arXiv:1812.04898v1 [cs.CL]
11. M. Roodschild, J. Sardiñas , A. Will, A new approach for the vanishing gradient problem on sigmoid activation, In: Artificial Intelligence, 9 (2020) 351–9.
12. S. Hochreiter and J. Schmidhuber, Long short-term memory, Neural computation, 9(8) (1997) 1735–45.
13. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones et al., "Attention is all you need," in Proc. NIPS, Long Beach, CA, USA, 2017.
14. B. Kim, S. Park, T. Won, J. Heo, B. Kim, Deep-Learning Based Web UI Automatic Programming, In: Conference on Research in Adaptive and Convergent Systems,2018, p.64-65.
15. T. Beltramelli, pix2code: Generating Code from a Graphical User Interface Screenshot, In: the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, 2018, DOI:10.1145/3220134.3220135
16. I. Magdalenić, D. Radošević, D. Kermek, Implementation Model of Source Code Generator, JOURNAL OF COMMUNICATIONS SOFTWARE AND SYSTEMS, 7(2) (2011) 71-9.
17. I. Magdalenic, D. Radoševic and T. Orehovački, Autogenerator: Generation and execution of programming code on demand, Expert Syst. Appl. 40(8) (2013) 2845–12.
18. M. Robeer, G. Lucassen, J. M. E. M. van Der Werf, F. Dalpiaz, S. Brinkkemper , Automated Extraction of Conceptual Models from User Stories via NLP, In: 24th International Requirements Engineering (RE) Conference,2016 , p. 196-205.
19. W. Ling, T. Lu´ıs, L. Marujo, R. F. Astudillo, S. Amir, C. Dyer, A. W .Black, I. Trancoso, Finding function in form: Compositional character models for open vocabulary word representation, In: Conference on Empirical Methods in Natural Language Processing,2015, p.1520-1530.
20. D. C. Wang, A. W. Appel, J. L. Korn, C. S. Serra, The zephyr abstract syntax description language, In: Conference on Domain-Specific Languages on Conference on Domain-Specific Languages (DSL), 1997, p.17–32.
21. J. Zhang, M. Utiyama, E. Sumita, G. Neubig, S. Nakamura, Guiding neural machine translation with retrieved translation pieces. In Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL),2018 , p.1325-1335
22. S. Rai and A. Gupta, Generation of Pseudo code from the Python source code using rule-based machine translation, arXiv e-prints, 2019.
23. Y. Oda, H. Fudaba, G. Neubig, H. Hata, S. Sakti, T. Toda, S. Nakamura. 2015, Learning to generate pseudo-code from source code using statistical machine translation (t). In: 30th IEEE/ACM International Conference on Automated Software Engineering (ASE),2015 , p.574–584.
24. A. Alhefdhi, H. Dam, H. Hata and A. Ghose, Generating Pseudo-code from source code using deep learning, In: 25th Australasian Software Engineering Conference (ASWEC) ,2018 , p. 21-25.
25. Y. Deng, H. Huang, X. Chen, Z. Liu, S Wu et al., From code to natural Language: type-aware sketch-based seq2seq learning, In: 25th International Conference on Database Systems for Advanced Applications(DASFAA),2020 , p. 352-368.

26. W. Gad, A. Alokla, W. Nazih, M. Aref , A. Salem, DLBT: Deep Learning-Based Transformer to Generate Pseudo-code from Source Code, CMC Computers, Materials and Continua 70(2) (2021) 3117-26.
27. Y. Wautelet , S. Heng ,M. Kolp ,I. Mirbel , (2014), Unifying and extending user story models. In: international conference on advanced information systems engineering (CAiSE) ,2014 ,p.211–225.
28. C. Corley , R Mihalcea, Measuring the Semantic Similarity of Texts. Proceedings of the ACL Workshop on Empirical Modeling of Semantic Equivalence and Entailment,2005 , p.13–18.
29. Princeton University “About WordNet" Available online: https://wordnet.princeton.edu/ (accessed on 3 March 2021).
30. https://github.com/anasAloklah/backend-web-ontology . (accessed on 23 July 2022).