Future Computing and Informatics Journal

Volume 4
Issue 2 *2019 Volume 4, Issue 2*

Article 1

2019

Ontological Engineering For Source Code Generation

anas hamid aloklah
anasaloklahaaa@gmail.com

walaa gad
walaagad@hotmail.com, walaagad@hotmail.com

mostafa mohamed aref prof
ain shams university, mostafa.aref@cis.asu.edu.eg

abd el-badea Mohamed salem prof
ain shams university, absalem@cis.asu.edu.eg

Digital Commons
ELSEVIER

Ontological Engineering For Source Code Generation

Anas Hamid Alokla[a], Walaa Khaled Gad [b], Mustafa .M Aref [c], Abdel-badeeh .M Salem [d]

Faculty of Computers and Information Sciences, Ain Shams University, Egypt
[a]*anasaloklahaaa@gmail.com* [b]*walaagad@hotmail.com*
[c]*mostafa.aref@cis.asu.edu.eg* [d]*absalem@cis.asu.edu.eg*

A B S T R A C T

Source Code Generation (SCG) is the sub-domain of the Automatic Programming (AP) that helps programmers to program using high-level abstraction. Recently, many researchers investigated many techniques to access SCG. The problem is to use the appropriate technique to generate the source code due to its purposes and the inputs. This paper introduces a review and an analysis related SCG techniques. Moreover, comparisons are presented for: techniques mapping, Natural Language Processing (NLP), knowledgebase, ontology, Specification Configuration Template (SCT) model and deep learning.

Keywords*:* Automatic Programming,, Source Code Generation, Ontological Engneering , knowledge Engineering, Natural Language Processing.

1. Introduction

Source Code Generation (SCG) is a sub-field of Automatic Programming (AP), which is a kind of computer programming that allows machine to generate software programs and hence programmers can program using a high-level abstraction. The AP has the following three types: Generative Programming (GP), Source Code Generation (SCG) and Program Synthesis. The SCG which produces a computer program is based on specific. There is no intellectually specific accurate method for accessing the source code generation intelligently. SCG consists of the following three parts: (1) user request (2) knowledge present as a template of code (3) configurator to configure between user request and code template. The Specification Configuration Template (SCT) Model is the core technique in (Ivan et al, 2013) and (Ivan et al, 2011). In the SCT model, Specification (S) includes: property of the generated application, attributes names and values. Template (T) is a source code of the target language in the templates where the source code replaces marks connected by Specification and replacing marks with variable names, variables and values. Configuration (C) is a connector between Specification and Templates. The template in SCT model could have connections to sub-SCT models (Ivan et al, 2011b) but there isn't an automatic method to measure the performance of the output in SCT model. The simplest way to access SCG is by mapping from source to target language but it is not enough to reach the desired results and therefore it is better to get what is required in the manner of semantic (Daniele et al, 2015). Both of SCT and Semantic are needed to use Ontology encompassing a representation, formal naming, and definition of the categories, properties,

Published by Arab Journals Platform, 2019

and relations between the concepts, data, and entities. Ontology accesses source code from a natural language sentence based on Natural Language Processing where sentence analysis links between roles and relations in sentences to get the (SMT). NMT includes encoder and decoder Deep Learning Model which is based on the probability of consecutive words from training dataset contents of source language and target language. Most NMT are built by a Recurrent Neural Network (RN
N); however, there are some problems such as vanishing and exploding gradient which were corrected by adding special neural network as Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber,1997). In 2017, Google

concept model and the mapping to ontology (Robeer et al, 2016). The Machine Transition is currently used in both the Neural Machine Translation (NMT) and Statistical Machine Translation
designed a model which works in attention layer called Transformer (Uszkoreit t al, 2017). NMT is used to convert Pseudo-Code to a programming language code (Abdulaziz et al, 2018)(Yin et al, 2017)(Maxim et al, 2017). There are two measures used to evaluate translation accuracy and BLEU score. BLEU score is a matric widely used to measure the performance of machine translation techniques. There are two methods to access SCG from a layout image. The First Method uses image

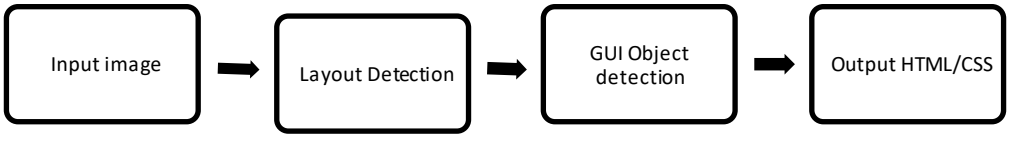

Fig 1 - Deep-Learning Based Web UI Automatic Programming pipeline

processing to detect the layout and R-CNN which recognize the object in the image (Bada et al, 2018).The second method uses CNN model in the encoder to analyse the image in input, and the RNN in the decoder to analyse and link the sentence in the output with input (Tony Beltramell,2018).
"Deep Learning" is a branch of Machine Learning in Artificial Intelligence (AI) which is based on artificial neural covers a discussion of the techniques results. Section 4 covers the Paper's Conclusions.

2. Source Code Generation (SCG) Techniques

In this section, the different (AI) techniques to access SCG are presented. These techniques are categorized into the

networks. It has multi-architectures such as deep neural networks, deep belief networks (DFN), recurrent neural networks (RNN), and convolutional neural networks (CNN) which have been applied in computer vision, speech recognition, natural language processing.
The rest of the paper is organized as follows. Section 2 covers the analysis of techniques to access the Source Code Generation (SCG). Section 3
following sub-sections: (1) Deep Learning (DL); (2) Natural Language Processing; (3) Semantic and Mapping; (4) Specification Configuration Template model and knowledge base. Every one of these techniques
works with different input data to generate the source code.

2.1. Source Code Generation based on Deep Learning

In (Bada et al, 2018), faster R-CNN and Computer Vision (CV) are used to convert hand-drawn sketch design of web Faster R- CNN is used to detect several features within an image. Fig (1) explains the pipeline of SCG based on DL and CV.

The pipeline consists of the following two phases: Layout Detection, and GUI Object Detection. The Layout Detection uses the following three algorithms to extract the Layout. The First algorithm edge-merged assembles and arranges edges correctly if the features are continuous, and arranged on both horizontal and vertical (X, Y) axes when they are merged and converted into a single straight line. The Second algorithm slope filtering seeks to reset the slope because the layout consists of horizontal and vertical components where the slope must be either 0 or 90 degrees to avoid noise in the sketch. The third algorithm is a correspondence line which detects column and row by finding a continuous line. The GUI Object detection and faster R-CNN are used to recognize html objects such as Button, RadioButton, checkBox, editText, text from image, followed by converting the object to HTML by XML labelling. The accuracy of recognizing objects in layout is 91%.
In (Tony Beltramell,2018), Deep Learning Model is divided into the following two models: Vision Model and Language Model. The (pix2code) is a vision model which uses CNN to solve the vision problem by learning features from layout images and mapping the image with input. The CNN works with fixed-vector and a layout which has different sizes where the layout images are resized to (255*255). The Language Model uses the RNN but with Long Short-Term Memory (LSTM) as a neural node to avoid the vanishing and exploding gradients problem where the language model works with context as input. De-coder in pix2code works by getting feature from output of CNN and tokens from the language model. The decoder learning is from features and tokens to input second LSTM. The output of the second LSTM is presented as Domain Specific Language (DSL) code. The dataset consists of: layout image and DSL code where the layout image is used in encoder and the DSL code is used in decoder. The dataset has the following three types of layouts: (1) web-based UI (HTML/CSS) (2) Android UI (XML) (3) iOS UI (Storyboard). The output of the model is DSL code. To convert DSL code to target language through the use of the compiler in the sampling phase. The Error with test set in the best case is 11.01 in web-based UI (HTML/CSS).
In (Abdulaziz et al, 2018), the designs of NMT module were to convert Python code to Pseudo-code (code2pseudocode). The encoder takes the input sentence of the code token and the decoder gets output sentence of the pseudo-code. Each encoder and decoder is called the Recurrent Neural Network (RNN) which connects them with Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber,1997) units. To improve the performance of the translation the attention mechanism is implemented (Bahdanau et al,2014) which aligns between items from the input and output sequences. The attention layer decides which input tokens have heavier weights in predicting the next output token. The (code2pseudocode) training on Django (Wang et al, 2016) dataset with 32 batch size and 23 epochs where 6% of the dataset was used for test performance and 94% was used for training. The method to code HTML/ CSS. The computer vision has layouts with distinct simple rule characteristics. Deep Learning Algorithms are used for complex or diverse object detection; but the

Published by Arab Journals Platform, 2019

used to evaluate of generated pseudo-code is called BLEU score (Papineni et al, 2002).

In (Maxim Rabinovich et al,2017), the Neural Machine Translation (NMT) module was designed to convert pseudo-code to Python code which is called ASN. The NMT works as sequence to sequence. To improve the method of working, the result applies a change in sequence output to tree output by applying the Abstract Syntax Trees (ASTs) in the decoder. The reason for using ASTs, is because the code in program languages can parser as a tree that can correct output if it has a syntax error. The ASN model uses the Abstract Syntax Description Language (ASDL) (Daniel C et al, 1997) framework. The ASDL parses the code and applies the grammar to the output. The input is a sequence of tokens (pseudo-code) from the HEARTHSTONE (HS) (Haitao Mi et al,2016) dataset where the input does not have parsing.The Model Architecture has the encoder and decoder with hierarchical attention. All work in ASN model where the decoder is the content collection of mutually recursive modules (Composite type modules, Constructor modules, Constructor field modules and Primitive type modules). The modules match and work with elements of the AST grammar, and are formed in a way that reflects the structure of the tree being generated. In the decoding process, a vertical LSTM state passes sequentially, into four modules to propagate the information. The loss function is negative log likelihood in the training step. The following is the details of the neural machine translation architecture:

Encoder using component-specific bi-directional LSTM for each of the components of the input.

Decoder decomposes into four classes of modules, which are working in constructing the grammar:

- The Composite type Modules select the rule (the stmt is if, while, for, return, etc.) in the input statement using vertical LSTM and apply a feed forward network followed by applying SoftMax output layer to choose a constructor. The result passes into the next module.
- The Constructor Modules compute the updated vertical LSTM states of rule selected for focus on the next field (if the rule selected is if statement the taken is test, body or else) feed-forwards network of Constructor module followed by using vertical LSTM.
- The Constructor Field Modules work on a number of children in role selected
- Primitive type modules work on the value of the role selected for example; if(x==5) then the role is if condition identifier is x. using vertical LSTM and SoftMax.

To improve result addition Supervised Attention (SUPATT), where SUPATT works into Alignment Component under supervised (Shirley et al, 2018).

In (Yin et al, 2017), (YN17) model is related to work on (Maxim et al, 2017) where work is in two directions:

- Grammar, the grammar model applied in sentences code and NL by an Abstract Syntax Tree AST, because the researches hypothesize that when use a structure sentence that performs to limit the search space and improves the code results. The hypothesis uses information structure to help into flow into the Recurrent Neural Network (RNN) and a decoder to allow for additional neural connections which reflect the recursive structure of an AST. The Grammar Model has two actions to work:
  - APPLYRULE[r]: Applies action production rule r (extraction what the role in the sentence, the research divides the sentences code by structure, if, loop, function call) in current AST.

GENTOKE[v]: Puts the value (v) in the node tree by a token word.

- Neural Machine Translationto convert a language to another language by the use of the neural network.

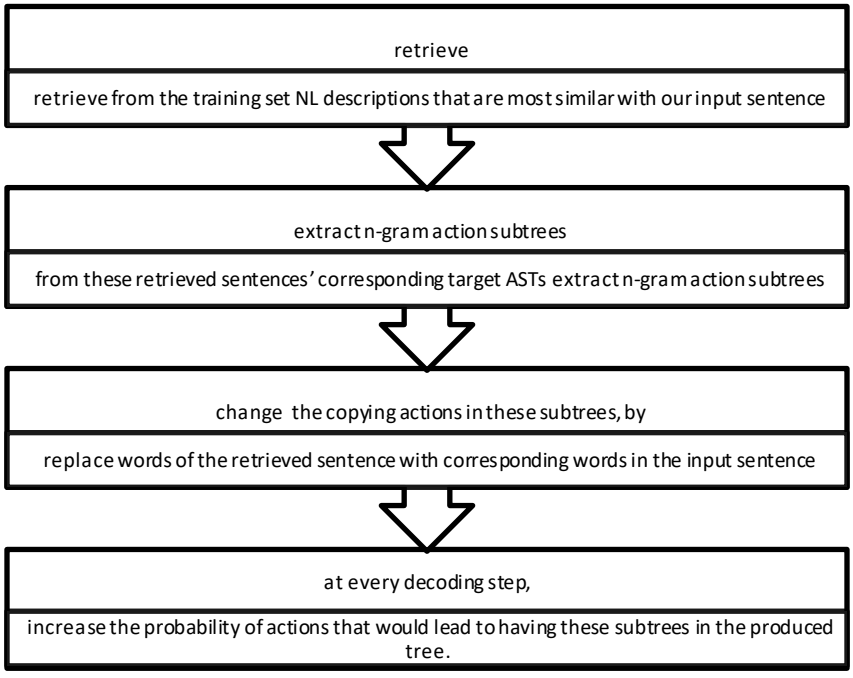

Fig 2 - The pipeline of the Retrieval-Based Neural Code Generation RECODE

Encoder has n of Weight W and h for each W when n= number of words in sentence and used bidirectional Long Short-Term Memory (LSTM). The encoder passes Context Vector (ct) to the decoder because the decoder uses soft attention where it uses a Deep Neural Network (DNN) with a single hidden layer to compute attention weights with the following (Bahdanau et al,2014)mechanism.

Decoderworks by RNN but represent model the sequential generation process of an AST and similar a vanilla LSTM, with supplemental neural connections to mirror the topological composition of an AST. In a decoder, the grammar model is applied for each node AST.

In (Shirley et al, 2018), the model retrieved in (Yin et al, 2017) is worked on the same dataset without IFTTT and object research but changing the model of NML to improve accuracy. The model used from (Jingyi Zhang et al,2018), is applied on multi languages (en, fr, de). The following pipeline is used to achieve Retrieval-Based Neural Code Generation RECODE as shown in Fig2.

Published by Arab Journals Platform, 2019

Find the similar NL input with training set, equation (1) used for this purpose

$$sim(q, q_m) = 1 - \frac{d(q,q_m)}{max(|q|,|q_m|)} \quad (1)$$

Where q is sentence input, q_m is sentence in the training set. $d(q, q_m)$ is the edit distance, m is the number of sentences in training which is set to retrieve only the top (M) sentences according to this metric where M is a hyper-parameter. These scores will later be used to increase action probabilities accordingly. Equation (1) from (Wang Ling et al,2015) uses AST in output like the previous paper and gets n-gram for the structure tree. The (n-gram) word was presented as sub-tree from the target code corresponding to the retrieved NL descriptions. Some sub-tree missing important information when getting (n-gram) forms the tree. To solve this problem, find candidates node in the tree and comparing with machine translation which uses n-grams of words.

Compute the edit distance (d) between input sentence and the retrieved sentence, by applying a one-to-one sentence alignment method for avoiding un-common words. Change all copies of rules extracted n-gram to correspond to the relevant rule; and delete the n-gram sub-tree; replace to be relevant in the predicted tree. N-gram sub-trees from all the retrieved sentences are assigned a score, based on the best similarity score of all instances where they appeared. We normalize the scores for each input sentence by subtracting the average over the training dataset. At decoding time, incorporate this retrieval of the derived scores into beam search, so as to increase the probability of actions that would lead to having these sub-trees in the produced tree.

In (Wang et al, 2016) the design Predictor Networks is different from deep learning models and prediction functions where this model is called the Latent Predictor Networks (LPN). The object of the LPN is to generate code from pseudo-com or natural language. The LPN model works through sentence-to-sentence framework. A token of sentence is encoded by C2W model (Wang Ling et al,2015). The LPN uses a bidirectional LSTM to build words in the text fields. The input isn't having a fixed size of vector to solve this problem must use learning a linear projection mapping. To compute a scalar coefficient, apply to each token thanh, a liner and Softmax function. The LPN model divides the prediction into the following three types: (1) Character Generation to predict character that by observes characters from the training data where the Softmax function is used to predict character. (2) Copy Singular Field to predict singular field like as the type of card or the value of the attack and cost attributes in HS dataset. Log P (y | x) function is used to predict singular field. (3) Copy Text Field to predict the words in text field to achieve this object uses RNN. A stack-based decoder with beam search is used to decode in the LPN model. Table 1 shows the comparison of performance measure of the four previous papers.

Table 1- Performance measure between four NMT models (Yin et al, 2017)

datasets / Model name	Hearthstone HS Accuracy	Hearthstone HS BLEU	Django Accuracy	Django BLEU
LPN (Wang et al, 2016)	4.5	65.6	62.3	77.6
ASN (Shirley et al, 2018)	18.2	77.6	-	-
ASN+ SUPATT (Shirley et al, 2018)	**22.7**	**79.2**	-	-
YN17 (Tony Beltramell,2018)	16.2	75.8	71.6	84.5
RECODE(Yin et al, 2017)	19.6	78.4	**72.8**	**84.7**

2.2. Source Code Generation based on Natural Language Processing

(Robeer et al, 2016) which works by the extraction of the conceptual models from user stories (list of natural language sentences following a standard format) where the conceptual model is presented as Ontology and converted to OWL2 and Prolog language. NLP is used to pars the sentences from user story, to expend the following three aspects: (1) Role: is a user or system need for the functionality. (2) Means: is a content function provided to user or system. (3) End is the optional aspects to explain why the user or system needs this function. Any sentence which doesn't have neither a role nor means, can't cause the sentence not to be understood.

Although many different templates exist, 70% of practitioners use the template "As a (type of user), I want (some goal) [so that some reason]" (M. Cohn,2004). Roles are contents indicator role and functional role; Means parsing to Means indicator, subject, main verb, and object. The main verb and object are contents to function for example I want to add media, means indicator I want to, the main verb is added for object media, now we have function addMedia.

Example: As an Administrator, I'm able to delete a destination. Table 2 explain sentence "as admin I'm able to delete destination" parsing in Universal (Masaru Tomita et al,1988) and Penn Treebank (Alexander Krotov et al,1988) methods. Information extracted from previous sentence, where the indicator is a word or phrase mention to function , verb or object. That show in table 3.

Published by Arab Journals Platform, 2019

Table 2 - Explains sentence parsing in user story

parser	As	Admin	I	'm	able	To	delete	Destination
Universal	ADP	PROPN	PRON	VERB	ADJ	PART	VERB	NOUN
Penn Treebank	IN	NNP	PRP	VBP	JJ	TO	VB	NN

Table – 3 show the words indicator and functionality of them.

indicator	Role	Means
value	As	I'm able to
Functional	Administrator	delete
Main object		destination

This is a method for extracting a function and role from one sentence. For extraction of conceptual model from user stories, must link between other sentences in the user story. (Robeer et al, 2016) has implemented VISUAL NARRATOR tool (Garm Lucassen et al,2017). The NARRATOR tool has an algorithm for the extraction of the conceptual model.

Algorithm Extract model form user story.

Algorithm 1 - Extract model form user story

```
Input user story (text)
For each sentence s from user story S
    Split by indicators to extract role r, means m, end e.
    Join r, m, e in S' set
    for each p from {r, m, e}
        Pt is Parsing tree of p, join nouns in C set, join subject in C set
        for each comp-nouns extract relation Is a and has a join in R set and every comp-noun join in C set
For each (r, m, e) from S'
    replace 'I' from m by r example: as user, I want to login. User is r => user want to login in site
    for each p from (m, e)
        find subject subj, mean-verb v and object obj
        create relation v(subj, obj) join in R set if subj and obj in C set
        create relation v(subj, system) join in R set if subj and obj is not in C set
Output Conceptual Model (Concept set C and relation set R)
```

Then, the Conceptual Model is converted to OWL2 and Prolog.In(Y. Oda et al,2015), the Statistical Machine Translation (SMT) is a designed model to generate pseudo-code from source code of python (T2SMT) where T2SMT uses a tree to string the translation rule, because the code can parse to a tree. The Training process of T2SMT consists of a source language and a target language where the source language consists of Training (1) parse python code to tree by AST. (2) extract leaf nods content tokens of code. (3) Words alignment is processed to find the relation of word between source and target sentence. The calculation of words alignment uses a probabilistic model and un-supervised machine learning techniques. (4) Rules are extracted from word alignment data, syntax tree and token array of target language. The extraction rules use GHKM algorithm (M. Galley et al,2004) to extract tree-to-string translation rules where these extract rules use a calculation of probabilities as a score to sort the rule table.

The following are the steps of target language to Training: (1) Token array phase converts the target language sentence to a token array. (2) Training language model phase uses $Pr(t)$ function equation (2) to apply to the target language to measure the fluency of the sentence (t). The object of the training language model is to extract language model. The Training model used to generate pseudo-code in T2SMT is called Travatar (G. Neubig,2013). The used dataset is Django with two languages pseudo-code i.e. English and Japan. The BLEU score is used to evaluate the pseudo-code generation. Table 4 - shows comparisons of performance measures of T2SMT and *code2pseudocode*.

$$Pr(t) \equiv \prod_{i=1}^{|t|} Pr(t_i|t_1, t_2, .. t_{i-1}) \quad \textbf{(2)}$$

Table 4 - Performance Measure T2SMT and code2pseudocode models (Abdulaziz et al, 2018)

Models	BLEU%
code2pseudocode(Abdulaziz et al, 2018)	54.78
T2SMT (Y. Oda et al,2015)	54.08

2.3. Source Code Generation Based on Semantic & Mapping

In (Daniele et al, 2015) The Semi-automatic generation is system presents itself as a desktop application with simple GUI. API layer is supervised by the user. The system shows a list of the interested namespace (show by semantic word) in GUI, and user change the properties of the given class or its superclass as their domain. The system generates API by mapping RDF. Figure 3 to shows the steps get API.

Published by Arab Journals Platform, 2019

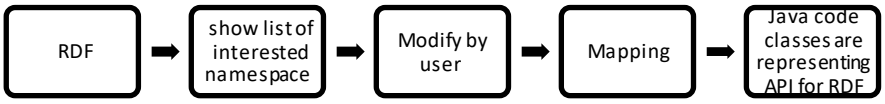

Fig 3 - the steps get API of an RDF in java languages

2.4. Specification Configuration Template Model and KnowledgeBase

In (Ivan et al, 2013) presents an Implementation Model of a Source Code Generator (IMSCG). The main objective of IMSCG is dynamic generation of ontology supported Web services for data retrieval. The user can generate new applications by defined user's request using Semantic Web Applications and generate source code for this purpose. The model uses for dynamic generation of ontology supported Web services for data retrieval is the SCT model.

The definition of IMSCG is independent of the programming language and can be implemented in different programming languages. The verification of the presented IMSCG is done by its implementation in Java for the purpose of dynamic generation of Web services for data retrieval. The method of generating source code is under level Generative Programming (GP). Generative application development is the process of parallel development of generators, together with the target applications (K. Czarnecki & U. Eisenecker,2000).

The Pipeline of generative application development is as follows:

- Extracting Program code templates from prototype.
- Replacing marks by features extracted from the specifications.
- The configuration detects mapping between the specifications and templates.

After generating an application, many mistakes could happen so we must improve that generative application by developing through the following three levels of developer's roles: The domain engineer's, The software programmer's role, The user's role where it is important that the user's role should deal with the application specification which must be separated from the program code level.

Afterwards, the generate application can face mistakes which are classified into the following three main mistakes:

- Mistakes in Specifications. Un-defined attributes, unacceptable attribute values or missing the attributes. The correction should be in the testing phase.
- Improper code templates. Syntax/logical errors in code template correction in the testing phase. The responsibility Software programmer's role
- Improper configuration. the configuration could not find the required template. Correction is responsibility of the domain engineer's role in the testing phase.

The Steps from the user request to application. User defines request by semantic meaning of data using ontology. Processing of the user request and building the application specification of web service application. The generator generates source code after getting input from the configuration of the source code, program code templates, and application specification. Compiling source code deployed to the web container and make

available to use. The Web services that are generated to retrieve data from data sources use the following types: XML, MS Excel, and RDBMS Oracle.
In (Ivan et al, 2011) related to the Implementation Model of Source Code Generator paper. uses SCT model like Implementation Model of Source Code Generator paper (Ivan et al, 2011), where core model developed on (Radoševic et al, 2012).
This model generates and executes a source code, but it differs from the previous model (Ivan et al, 2011) in the Role and Lifecycle where the auto-generator has an adaptation to the Auto-generators require a component before the testing and prototype as input into the SCT Generator. There are the following three Roles in the Auto-generator:

- Domain Expert: The Role is responsible for: application analysis, the building of prototypes and the creation of program code templates.
- Generator Developer: The Role is responsible for: the creation of program code templates, building of source code generator configuration which defines the specification elements and building of auto-generator request handler.
- Application User: The User Role is: Sending requests, receiving responses but their application developer is used for the creation of specifications for one application.

Results vary according to purpose and Technique used in table 5 shows compare between Techniques, input and output.

3. Discussions

There are different approaches to access the Source Code Generation but they aren't good choices use DL because they depend on a dataset and the current datasets aren't big enough where the accuracy in the best case is 72.8 in the Django dataset RECODE model and 22.7 in the HS dataset ASN+ SUPATT model. The good approach is using the SCT model because it depends on knowledge base and ontology. The present knowledge as templates of code which are converted to source code with user requirements and can correct mistakes in the testing phase and add or modify into the knowledge base to solve new problems. The disadvantages of using NLP are miss-understanding and not understanding of sentences. To solve these problems, the sentences need to be drafted.

4. Conclusion

This paper introduced the SCG techniques are introduced to generate SCG i.e. (deep learning, NLP, semantic, ontology, knowledge base, and SCT model) to generate SCG but there isn't one technique used to access SCG; and there is a combination of techniques used to improve the results. Recent research use deep learning technique to SCG which use models CNN, RNN, LSTM, attention and supervised attention. The reason for the different used techniques is due to the purpose of SCG where the output is always a text, but the input differences between image, text and user request is due to the purpose of SCG. There is also no standardized way to measure the results as they differ according to the technique used where in some of them, there is no automatic method to measure performance which is measured by the observation of results.

Published by Arab Journals Platform, 2019

Table 5 – Different the Techniques of Intelligent Source Code Generation

Authors and Year	Techniques	Input	Output	Dataset
(Bada et al, 2018)	Deep-Learning, Computer vision.	Sketch image	HTML & CSS	-
(Tony Beltramell,2018)	Deep learning	Sketch image	web-based UI (HTML/CSS), Android UI (XML), iOS UI (Storyboard)	Pix2code
(Abdulaziz et al, 2018)	Deep learning, NMT	Code in python	Sentence in natural language (Pseudo-Code)	HS – Django
(Shirley et al, 2018)	Deep learning,	Pseudo-Code	Code in python	HS – Django
(Yin et al, 2017)	Deep learning,	Pseudo-Code	Code in python	HS – Django
(Maxim et al, 2017)	Deep learning,	Pseudo-Code	Code in python	HS – Django
(Wang et al, 2016)	Deep learning	Pseudo-Code	Code in python	HS - Django – MTG
(Robeer et al, 2016)	NLP	user story	OWL 2 and Prolog program	user story
(Yusuke et al, 2015)	NLP, SMT	Code in python	Sentence in natural language (Pseudo-Code)	Django (English and Japanese)
(Daniele et al, 2015)	Semantic, Mapping	RDF	Java code	-
(Ivan et al, 2013)	SCT model, knowledge base	User request	HTML (and contained js), XML and CGI script in Python.	-
(Ivan et al, 2011)	Ontology, SCT model	User request	Java code	-

REFERENCES

Ivan Magdalenić, Danijel Radošević, and Dragutin Kermek, Implementation Model of Source Code Generator, JOURNAL OF COMMUNICATIONS SOFTWARE AND SYSTEMS, VOL. 7, NO. 2, JUNE 2011

Ivan Magdalenic, Danijel Radoševic and T. Orehovački, Autogenerator: Generation and execution of programming code on demand, Expert Syst. Appl. 40(8) (2013) 2845–2857.

Ivan Magdalenic& Danijel Radoševic ´ I. (2011b). Source code generator based on dynamic frames. Journal of Information and Organizational Sciences, 35(2), 73–91.

Daniele Toti and Marco Rinelli,Semi-automatic generation of an Object-Oriented API framework over semantic repositories,2015 International Conference on Intelligent Networking and Collaborative Systems,DOI: 10.1109/INCoS.2015.22.

M. Robeer, G. Lucassen, J. M. E. M. van Der Werf, F. Dalpiaz, and S. Brinkkemper (2016). Automated Extraction of Conceptual Models from User Stories via NLP. In 2016 IEEE 24th International Requirements Engineering (RE) Conference (pp. 196-205). IEEE.

S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Abdulaziz Alhefdhi , Hoa Khanh Dam , Hideaki Hata , Aditya Ghose. Generating Pseudo-Code from Source Code Using Deep Learning. Published in: 2018 25th Australasian Software Engineering Conference (ASWEC). Pages 21-25.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In The 55th Annual Meeting of the Association for Computational Linguistics (ACL), pages 440–450.

Maxim Rabinovich, Mitchell Stern, and Dan

Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In Proceedings

of the 55th Annual Meeting of the Association for Computational Linguistics (ACL), pages 1139–1149.

Bada Kim, Sangmin Park, Taeyeon Won, Junyoung Heo, Bongjae Kim Deep-Learning Based Web UI Automatic Programming, Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems, Pages 64-65,2018.

Tony Beltramelli,pix2code: Generating Code from a Graphical User Interface Screenshot,Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems Article No. 3. Paris, France — June 19 - 22, 2018

D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*,2014.

Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura.

2015. 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), pages 574–584.

K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in Proceedings of the 40th annual meeting on association for computational linguistics. Association for Computational Linguistics, 2002, pp. 311–318.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL), pages 1139–1149.

Daniel C. Wang, Andrew W. Appel, Jeff L. Korn, and Christopher S. Serra. 1997. The zephyr abstract syntax description language. In Proceedings of the Conference on Domain-Specific Languages on Conference on Domain-Specific Languages (DSL), 1997. USENIX Association, Berkeley, CA, USA, DSL'97, pages 17–17.

Haitao Mi, Zhiguo Wang, Abe Ittycheriah, Supervised Attentions for Neural Machine Translation,Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 2283–2288,

Shirley Anugrah Hayati, Raphael Olivier, Pravalika Avvaru, Pengcheng Yin, Anthony Tomasic, Graham Neubig,2018. Retrieval-Based Neural Code Generation. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 925–930.

Jingyi Zhang, Masao Utiyama, Eiichiro Sumita, Graham Neubig, and Satoshi Nakamura. 2018. Guiding neural machine translation with retrieved translation pieces. In Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL).

Wang Ling, Tiago Lu´ıs, Lu´ıs Marujo, R´amon Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing.

M. Cohn User Stories Applied: for Agile Software Development. Redwood City, CA, USA: Addison Wesley Professional, 2004.

Masaru Tomita, Marion Kee, Hiroaki Saito, Teruko Mitamura and Hideto Tomabechi ,The Universal Parser Compiler and Its Application to a Speech Translation System,Published in the Proceedings of the Second International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages, at Carnegie-Mellon University in Pittsburgh, Pennsylvania, June, 1988.

Alexander Krotov , Mark Hepple , Robert Gaizauskas and Yorick Wilks ,Compacting the Penn Treebank Grammar,Proceeding ACL '98/COLING '98 Proceedings of the 36th Annual

Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1 Pages 699-703 Montreal, Quebec, Canada August 10 - 14, 1998

Garm Lucassen. Marcel Robeer. Fabiano Dalpiaz. Jan Martijn E. M. van der WerfSjaak Brinkkemper, 2017, Extracting conceptual models from user stories with Visual Narrator, springer Requirements Engineering September 2017, Volume 22, Issue 3, pp 339–358.

Y. Oda, H. Fudaba, G. Neubig, H. Hata, S. Sakti, T. Toda, and S. Nakamura, "Learning to generate pseudo-code from source code using statistical machine translation (t)," in Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on. IEEE, 2015, pp. 574–584.

M. Galley, M. Hopkins, K. Knight, and D. Marcu, "What's in a translation rule?" in Proc. NAACL-HLT, 2004, pp. 273–280.

G. Neubig, "Travatar: A forest-to-string machine translation engine based on tree transducers," in *Proc. ACL*, Sofia, Bulgaria, August 2013, pp. 91–96.

K. Czarnecki, U. Eisenecker, Generative programming: methods, tools and applications, Addison-Wesley, 2000.

Radoševic´ , D., Orehovacˇki, T., & Magdalenic´ , I., 2012. Towards software autogeneration. In Proceedings of the 35th MIPRO jubilee international convention on intelligent systems, Rijeka, Croatia (pp. 1257–1262).