DOI: 10.15514/ISPRAS-2022-34(2)-2

An algorithm of test generation from functional specification using Open IE model and clustering

K.S. Kobyshev, ORCID: 0000-0002-1120-9569 <kobyshev2.ks@edu.spbstu.ru>
S.A. Molodyakov, ORCID: 0000-0003-2191-9449 <molodyakov_sa@spbstu.ru>
Peter the Great Saint Petersburg Polytechnic University,
29, Politechnicheskaya st., St. Petersburg, 195251, Russia

Abstract. Automated test coverage is a widespread practice in long-live software development projects **for now.** According to the test development approach, each automated test should reuse functions implemented in test framework. The provided research is aimed at improving the test framework development approach using natural language processing methods. The algorithm includes the following steps: preparation of test scenarios; transformation of scenario paragraphs to syntax tree using pretrained OpenIE model; test steps comparison with test framework interfaces using GloVe model; transformation of the given semantic tree to the Kotlin language code. The paper contains the description of a prototype of the system automatically generating Kotlin language tests from natural language specification.

Keywords: automatic test; computational linguistics; relation extraction; open information extraction; dependency tree parsing; natural language processing; clustering; E2E test; GloVe; Kotlin

For citation: Kobyshev K.S., Molodyakov S.A. An algorithm of test generation from functional specification using Open IE model and clustering. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 2, 2022, pp. 17-24. DOI: 10.15514/ISPRAS–2022–34(2)–2

Алгоритм генерации тестов из функциональной спецификации с использованием модели Open IE и кластеризации

К.С. Кобышев, ORCID: 0000-0002-1120-9569 <kobyshev2.ks@edu.spbstu.ru>
С.А. Молодяков, ORCID: 0000-0003-2191-9449 <molodyakov_sa@spbstu.ru>
Санкт-Петербургский Политехнический университет Петра Великого,
Россия, 195251, Санкт-Петербург, Политехническая ул., 29

Аннотация. Автоматизированное тестовое покрытие на данный момент является широко распространенной практикой в долгосрочных проектах разработки программного обеспечения. Согласно подходу к разработке тестов, каждый автоматизированный тест должен повторно использовать функции, реализованные в тестовой среде. Представленное исследование направлено на совершенствование подхода к разработке тестовой среды с использованием методов обработки естественного языка. Алгоритм включает следующие этапы: подготовка тестовых сценариев; преобразование абзацев сценария в синтаксическое дерево с использованием предварительно обученной модели OpenIE; сравнение шагов тестирования с интерфейсами тестового фреймворка с использованием модели GloVe; преобразование заданного семантического дерева в код языка Kotlin. Статья содержит описание прототипа системы автоматической генерации языковых тестов Kotlin из спецификации на естественном языке.

Ключевые слова: автоматический тест; компьютерная лингвистика; извлечение отношений; извлечение открытой информации; разбор дерева зависимостей; обработка естественного языка; кластеризация; E2E-тест; GloVe; Kotlin

Для цитирования: Кобышев К.С., Молодяков С.А. Алгоритм генерации тестов из функциональной спецификации с использованием модели Open IE и кластеризации. Труды ИСП РАН, том 34, вып. 2, 2022 г., стр. 17-24. DOI: 10.15514/ISPRAS–2022–34(2)–2

1. Introduction

Automated test coverage is a widespread practice in long-live software development projects for now. The coverage can be implemented on different levels of testing pyramid: unit tests, integration tests, API (Application programming interface) tests, E2E (End-to-End) tests [1]. The practice of test coverage allows us to decrease complexity of code refactoring process, also tests can be used as primary code documentation according to the Test-Driven Development methodology [2].

Also, there is another popular approach of test framework development. This approach complements the automated test coverage approach. According to the test development approach, each automated test should reuse functions implemented in test framework [3]. Therefore, after implementation of test framework, we have a clear architecture of test infrastructure without code duplication and with reusable test steps and objects.

The provided research is aimed at improving the test framework development approach, at reducing labor costs of this approach using natural language processing methods. We consider the existing test automation approaches, define their shortcomings and analyze how these shortcomings can be addressed.

2. Problems of existing testing automation approaches

When a programming system is quite complex, usually, analysts prepare a document describing the system behavior called a functional specification. Usually, in case of complex and long living projects, the functionality should be delivered by short release cycles, a program build should be delivered immediately after functionality implementation. In this case it is necessary to check not only the new functionality, but also existing earlier. In other words, it is necessary to complete the automated regression testing in such cases. Consider the testing automation methods presented in Table 1 and define their disadvantages.

Table 1. Existing approaches characteristics

Approach / Characteristic	Classic	BDD	Formal verification methods	Neural Network training
Test structuredness	-	++	++	--
Analyst participation	--	++	++	+
Source code complexity resistance	++	++	--	++
Reliability	+	+	++	--
Automation	--	--	+	++

According to the classic testing automation approach, analyst should prepare a functional specification that is used for automatic test preparation by QA engineers (Quality Assurance engineer). Automatic tests are prepared manually. This method forces analyst and QA engineers to work separately. Participation of analysts is minimal and interaction between analysts and QA engineers is done over the document – functional specification. Also, QA engineers are responsible of test framework structure support. This approach excludes the full automation of test preparation.

The BDD approach (Behavior-Driven Development) is based on test framework interfaces preparation by analyst with using of domain-oriented language [4]. Analysts prepare structure of test framework and QA engineers implements the test framework. This approach allows to achieve the high level of test structuredness. This approach like the classic approach, excludes the full automation of test preparation.

A set of formal verification methods allows us to check completely the program correctness according to functional specification requirements, made with, for example, language of temporal logic [5]. The performance of verification process significantly degrades with increasing of cyclomatic complexity of program source code. The formal verification process is a check of all possible program states, which can cause the "combinatorial explosion". Therefore, the formal verification usually applied for prototype of program instead of the source program.

Also, there is an approach based on the training of neural network [6]. Authors proposed to train neural network by random input data for program and given from its output data. This approach does not take in account analyst participation and testing is based on already prepared program. But this approach cannot guarantee the reliability because it is impossible to make the completely correct trained neural network model. Also, it is impossible to continue the model training with new program changes.

So, the following problems were found out during the existing methods analysis:

- Chaotic state, low level of test structuredness.
- Analysts work separately from QA engineers, absence of correct unified understanding of expected system behavior. Their work can be done only through documents, functional specification, consisted of non-strict natural language texts.
- Low testing system performance when source code of checked program is complex.
- Absence of guarantee that automatic testing system is completely correct.
- A lot of manual work on preparation of test infrastructure.

Consider the proposed approach, and how this approach allows us to deal with these enumerated shortcomings.

3. Proposed test development automation approach

Consider the solution proposed in the current research schematically presented in fig. 1.

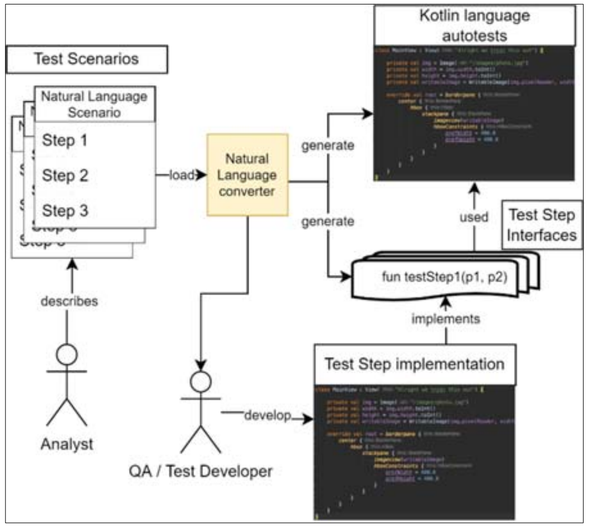

Fig. 1. The proposed solution for automatic test generation

We proposed to organize the development process by the following way:

- Analysts prepare natural language scenario set.
- Natural language test scenarios are transformed to interfaces of test steps by the proposed automatic software tool and to tests that are using interfaces from the generated test framework.
- QA engineer implements given test step interfaces on Kotlin programming language.

So, the main idea is to convert automatically non-strict natural language test scenarios to the stricter Kotlin programming language using existing natural language processing methods. Thanks to automated natural language processing of test scenarios, we achive such advantages of the considered BDD test automation approaches as good test structuredness, consolidated understanding of system behavior between analysts and test developers, high reliability of testing system. Also, we decreased the manual work on test infrastructure preparation with automated test scenario processing. All, that test developer should do is implementation of test framework interfaces. The structure of tests and tests themselves will be prepared automatically.

Consider the proposed solution in detail, each step of the proposed test scenario processing algorithm.

4. Test generation algorithm steps

Consider the proposed algorithm steps schematically presented in fig. 2.

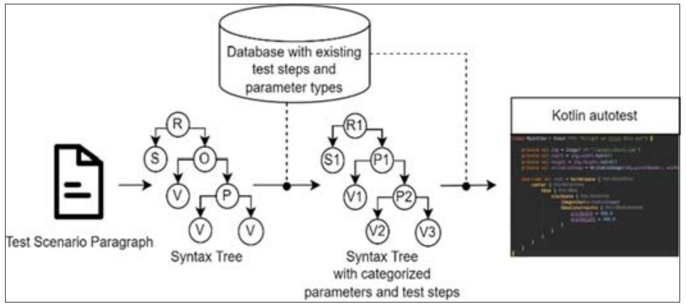
Database with existing test steps and parameter types
Test Scenario Paragraph
R
S
O
V
P
V
V
Syntax Tree
R1
S1
P1
V1
P2
V2
V3
Syntax Tree with categorized parameters and test steps
Kotlin autotest

Fig. 2. Steps of the proposed test generation algorithm

The proposed method includes the following steps:

1. A test scenario name is taken as a test method name.

2. The test scenario is divided to sentences. Each sentence will be transformed to the one line of final code.

3. Each sentence is transformed to the syntax tree using the pretrained OpenIE model [7].

4. Test step, parameter group and separate parameter names are associated with test step, parameter group and parameter types using GloVe model [8, 9].

5. The given semantic tree is transformed to the Kotlin language code.

Consider steps 3, 4, 5 in detail.

5. Syntax tree preparation

OpenIE model is used to build the syntax tree from test scenario sentence [7]. Before OpenIE processing, the text data should be prepared by the following algorithms: tokenization [10], lemmatization [11], part-of-speech definition [12], building the dependency tree D [13]. Triplets are formed with using of OpenIE according to the expression (1), where s is a subject, R is a relation, o is an object:

$$T_i = s_i R_i o_i \tag{1}$$

In some cases, an object contains a set of several interconnected natural language words. The object can be presented in a form of a part of dependency tree, therefore according to the expression (2):

$$a_i \in D_i \tag{2}$$

This view allows us to present the object as a hierarchic structure of different parameters, that will make automatic tests more descriptive. The dependency tree can be presented by expressions (3) and (4), where P are tree nodes, and V are leaves. In other words, these leaves are values V of parameters P. And parameters P can include other parameters P or values V, so o can be presented in a form of hierarchic structure, so tests will contain trees of parameters P with values V:

$$o_i = P \cup V = (P_1, P_2, \dots, P_k) \cup (V_1, V_2, \dots, V_k) \quad (3)$$

$$\forall n, P_n = \left(P_x, P_{x+1}, \dots, P_y\right) \cup (V_m, V_{m+1}, \dots, V_l) \quad (4)$$

For now, when the current step is completed, found subjects, relationships, parameter sets, and values are not associated with any types. In the next step, they will be classified to form interfaces of test framework.

6. Test element type definition

As a result of the previous step, we got a hierarchically connected subjects *s*, relationships *R*, parameter sets *P*, values *V*. Each *s, R, P, V* is associated with some source natural language word or word set. Any natural language word can be presented in a form of coordinates vector in semantic space. Close *s, R, P, V* can be grouped to clusters associated with test framework interfaces.

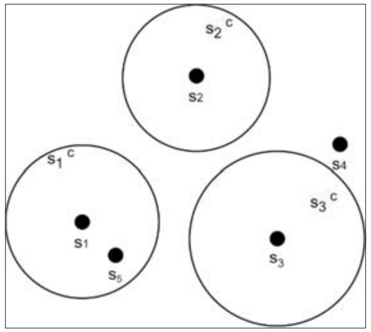

Fig. 3. Clustering of natural language words in semantic space

For now, there are many ways to get natural language word coordinates in semantic space. The most used for today models presenting word semantic coordinates are: RNNLM [14], word2vec [8], GloVe [15], fastText [16]. The GloVe model was used in the proposed method because this model takes in account in significant degree word cooccurrence frequency, that is important for our clustering.

As it was discussed earlier, we got a syntax tree D and a set (s, R, P, V). Also, before clustering, we have a set (s^{c0}, R^{c0}, P^{c0}, V^{c0}), associated with a cluster set (s_0', R_0', P_0', V_0') found earlier on clustering of previous test scenario sentence words.

Each subset s, R, P, V is divided to clusters separately. Consider an example in the fig. 3 in two-dimensional space, when clusters s_1^c, s_2^c already found from previous test scenario sentences and for now we want to parse 3 remaining sentences and define their s, R, P, V types or clusters.

After parsing of three remaining sentences, as a result, algorithm extracts subjects s_3, s_4, s_5 from these three sentences. Clusters of these subjects are defined in the following way. So, we get a point in the two-dimensional semantic space. If there are no clusters in radius r from the given point, then the cluster with radius r will be placed at this point and the point will be a cluster center. If the point is in the other cluster zone, then this point will be associated with that cluster. If the point is not in cluster, but the r-radius circle from this point intersects with any cluster, then the point will be associated with the closest cluster.

We can see on the fig. 3 that clusters s_1^c, s_2^c were found at the beginning. Then algorithm accepted the point s3, that was associated with the cluster s_3^c, because the r-radius circle from this point is not

intersected with any existing r-radius clusters. The r-radius circle of point s_4 is intersected with cluster s_3^c, that is why it was associated with the cluster s_3^c. The point s_5 was associated with the cluster s_1^c because it was inside of the r-radius circle of this cluster.

The last remaining step is to get the Kotlin language code from the given semantic tree.

7. Semantic tree transformation to the Kotlin language code

The last step is to get the Kotlin language code from the given typed semantic tree. As a result, we will get an automatic test on the domain-oriented language and interfaces of the test framework. Consider transformation rules presented in the Table 2, where you can see examples of the parsed sentence in the "before" column and prepared automatic test code fragment in the "after" column.

Table 2. Transformation rules to the Kotlin language code

Transformation rule	Before	After

Subject	User paid free package
	User - subject
		`user {`
		`    …paid free package...`
		`}`

Subject grouping	User paid free package.
	User got payment bill.
		`user {`
		`   ...paid free package,`
		`      got payment bill…`
		`}`

Relationship	User paid free package	`user {`
		`   paid(…)`
		`}`

Object	User paid free package	`user {`
		`   paid(Package(…))`
		`}`

Parameter	User paid free package	`user {`
		`   paid(Package(type=…)`
		`}`

Value	User paid free package	`user {`
		` paid(Package(type=FREE)`
		`}`

Test Scenario	Payment flow:
	User paid free package.
	User got payment bill.
		`@Test`
		`fun paymentFlow() {`
		`user {`
		` paid(Package(type=FREE)`
		` got(PaymentBill())`
		` }`
		`}`

The found subject is transformed to the lambda expression with context. QA engineer should implement the context class. If the same subject is appeared in two test scenario sentences, then those subject lambda expressions will be grouped to the one lambda expression. The found relationship is transformed to the method call, and that method should be implemented. Parameters are transformed to the class field names. Values are transformed to the primitive types of the Kotlin language or Strings. Then all code is wrapped to the test method having the name like the test scenario name.

8. Prototype of the proposed solution

A prototype of the proposed solution was implemented on Java language. The developed system uses a pretrained OpenIE model in a form of Maven package manager dependency called Stanford NLP. A pretrained GloVe model was used. This model was given from Wikipedia of 2014 year and

Gigaword text corpuses. The model contains 400 thousand words and their coordinates in 100-dimensional space and takes 822 Mb of memory. The GloVe model was stored and indexed in Mongo database. For now, the prototype gives true results for simple test scenarios, however, we found that it does not work correctly in some complex test scenarios including multiple words in subjects and relationships. Therefore, we need to investigate more and improve clustering stage of the proposed algorithm for now.

9. Conclusion

In the provided research we analyzed existing automated testing approaches and defined their disadvantaged. After the analysis we proposed solution based on natural language processing of test scenarios and transformation of them to the Kotlin language autotests and test framework.

As a result of research, we implemented a prototype of the proposed algorithm on Java language in a form of Maven open-source library. The developed solution includes a pretrained OpenIE model from Stanford NLP library. A pretrained GloVe model was used to automate a search of test items categories. This model was given from Wikipedia of 2014 year and Gigaword text corpuses. The model contains 400 thousand words and their coordinates in 100-dimensional space and takes 822 Mb of memory. The GloVe model was stored and indexed in Mongo database.

For now, the prototype gives true results for simple test scenarios, however, we found that it does not work correctly in some complex test scenarios including multiple words in subjects and relationships. Therefore, we need to investigate more and improve clustering stage of the proposed algorithm for now.

References / Список литературы

[1] N. Radziwill, G. Freeman Gr. Reframing the Test Pyramid for Digitally Transformed Organizations. Software Quality Professional, vol. 22, issue 4, 2020, pp. 18-25.

[2] I. Karac and B. Turhan. What Do We (Really) Know about Test-Driven Development? IEEE Software, vol. 35 issue 4, 2018, pp. 81–85.

[3] M.F. Fontoura. A systematic approach for framework development. PhD thesis, Pontifical Catholic University of Rio de Janeiro, 1999.

[4] M. Irshad, R. Britto and K. Petersen. Adapting Behavior Driven Development (BDD) for large-scale software systems. Journal of Systems and Software, vol. 177, 2021, article no, 110944, 20 p.

[5] W. Wasira. Existing Tools for Formal Verification and Formal Methods. 2020. DOI: 10.13140/RG.2.2.12162.22721.

[6] A.D. Danilov and V.M. Mugatina. Verification and testing of complex software products based on neural network models. VSTU Bulletin, vol. 12, no. 6, 2016, pp. 62-67 (in Russian) / А.Д. Данилов, В.М. Мугатина. Верификация и тестирование сложных программных продуктов на основе нейросетевых моделей. Вестник Воронежского государственного технического университета, том 12, no. 6, 2016 г, стр. 62-67.

[7] G. Angeli, M. Premkumar, C. Manning. Leveraging Linguistic Structure for Open Domain Information Extraction. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, 2015, pp. 344–354.

[8] L. Ma and Y. Zhang. Using Word2Vec to process big text data. 2015 IEEE International Conference on Big Data, 2015, pp. 2895-2897.

[9] A.D. Kovalev, I.V. Nikiforov, P.D. Drobincev. Automated approach to semantic search through software documentation based on Doc2Vec algorithm. Information and control systems, no. 1, 2021, pp. 17-27 (in Russian) / А.Д. Ковалев, И.В. Никифоров, П.Д. Дробинцев. Автоматизированный подход к семантическому поиску по программной документации на основе алгоритма Doc2Vec. Информационно-управляющие системы, no. 1, 2021 г., pp. 17-27.

[10] R.M. Garcia-Teruel, H. Simon-Moreno. The digital tokenization of property rights. A comparative perspective. Computer Law & Security Review, vol. 41, issue 2, 2021, pp. 1-16.

[11] B. Vimala, E. Lloyd-Yemoh. Stemming and Lemmatization: A Comparison of Retrieval Performances. Lecture Notes on Software Engineering, vol. 2, no. 3, 2014, pp. 262-267.

[12] S. Chotirat, P. Meesad. Part-of-Speech tagging enhancement to natural language processing for Thai wh-question classification with deep learning. Heliyon, vol. 7, issue 10, 2020, article no. e08216, 13 p.

[13] R. Zmigrod, T. Vieira, R. Cotterell. On Finding the K-best Non-Projective Dependency Trees. In Proc. of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, 2021, pp. 1324-1337.

[14] G. Lecorve, P. Motlicek. Conversion of Recurrent Neural Network Language Models to Weighted Finite State Transducers for Automatic Speech Recognition. In Proc. of the 13th Annual Conference of the International Speech Communication Association, 2012, 4 p.

[15] J. Pennington, R. Socher, C. Manning. Glove: Global Vectors for Word Representation. In Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532-1543.

[16] I.N. Khasanah. Sentiment Classification Using Fast Text Embedding and Deep Learning Model. Procedia Computer Science, vol. 189, 2021, pp. 343-350.

Information about authors / Информация об авторах

Kirill Sergeevich KOBYSHEV – postgraduate student at High School of Software Engineering. Research interests: computational linguistics, natural language processing.

Кирилл Сергеевич КОБЫШЕВ – аспирант Высшей школы программной инженерии. Область научных интересов: компьютерная лингвистика, обработка естественного языка.

Sergey Aleksandrovich MOLODYAKOV – Doctor of Technical Sciences, Professor of High School of Software Engineering. Research interests: image recognition, digital signal processing, video processors.

Сергей Александрович МОЛОДЯКОВ – доктор технических наук, профессор Высшей школы программной инженерии. Область научных интересов: распознавание изображений, цифровая обработка сигналов, видеопроцессоры.