Artificial Intelligence Methods for Services and Product Sustaining Phase

Alexandra Sokolova, Danil Safronov, Kirill Stonozhenko, Maxim Solomonov, Igor Nikiforov, Artem Kovalev
Peter the Great St.Petersburg Polytechnic University
Saint Petersburg, Russia
{sokolova.ae, safronov.d, stonozhenko.km, solomonov.ms, nikiforov_iv, kovalev.ad}@edu.spbstu.ru

***Abstract*—The main problem the project addresses is reducing time and effort spent by a vendor company on product sustaining phase. This is done by development of the approach for assisting with customer request resolution based on intellectual methods and machine learning algorithms.**

Solution collects resolved issues from bug tracking systems, local documentation and confluence pages, creates their respective vector models and teaches the algorithm on the data. For each retrieved unresolved issue, related resolved ones and documentation pages are found as a result of using trained algorithm. System also collects user-written rules, that are based on information from the issue, and checks them on each unresolved issue to give recommendations regarding its status change or additional needed information. Based on received information, the final report is constructed to show related issues, documentation and recommendations for each unresolved issue in a user-friendly manner.

System was tested on Apache Kafka project issues and compared to manual approach performed on same data. The average time to analyze unresolved issues using the automated approach was 12.2 minutes, and the average time spent with the manual approach was 18.4 minutes, which means that our solution decreases complexity of issue analysis by ~33%.

I. Introduction

The maintenance is one of the most expensive phases in a software product lifecycle [1]. In 2005, the maintenance accounts for 67% of the project total cost [2]. This is caused by a growing of number of places, where an error should be fixed (implementation, workflow and/or documentation). Every bug in product/service or even a question is the subject for customer to create a request to the development company for an assistance. A vendor company needs to spend around 40% of the development time and involve a lot of human resources on finding solutions to customer's requests during maintenance stage. Moreover, teaching newcomers at support department to understand and operate the provided product/service remains one of the most complex issues on this stage.

Improving the quality of support improves a long-term relationship between the service/product provider and the customer and also makes educational process easier for the new support engineers [3]. Thus, the proposed solution to automate and assist with debugging product issues/bugs/questions can be considered as an actual task for the field.

Since then, many approaches appeared to reduce this problem [4]. One of them is bug tracking systems, like Jira [5] or Bugzilla. Their primary objective is making a connection between customer and developer more subtle. Bug tracking systems allow to form, collect and store issues about errors, bugs or proposals of improvements for a software product [6].

However, this approach has its flaws. A developer team has to spend time on analyzing new issues. Such as: prioritize, find identical, check correctness and so on. These tasks are able to be automated with using intellectual methods and machine learning algorithms. It is the main goal of our project.

The most similar tools, which partly solve the problem, are Jira plugin Automation for Jira Server [7] and software tool Amelia [8].

Automation for Jira Server gives a possibility to create if-then-else rules for different events. For example, if issue's type is "bug" then assign it to engineer "A". This tool has intuitive and simple interface. But it is not able to work with unstructured data in the description or commentaries in an issue.

Amelia is the smart assistant for employees in a company. It aims at forming answers on natural language for the most frequent questions and does not really supports an integration in development process with Jira or Bugzilla.

Our way gives more opportunities for engineers to process incoming requests. It is based on generating a report with the meta-information about every issue. This report let developing team see dependencies between issues and the most relevant problems and suggest a solution for each step.

II. Literature Review

The main part of our approach is algorithm of finding similar documents (issues and documentation) [9]. We have compared following text information processing methods.

BM25 [10] – is the ranking function, based on TF-IDF [11]. It is used in modern search engines (ElasticSearch). BM25 has the same advantages like TF-IDF: reduction of impact of the more frequently used words and normed word weight. Indeed, function contains regulating factors. The disadvantage is negative index value for words with high frequency.

Latent semantic analysis (LSA) [12] – the algorithm, which is able to identify correlations between documents in a corpus and to systematize ones into topics. The advantages: working with both documents and terms, using in

classification and clusterization and avoidance of influence of polysemy and homonymy. But this algorithm requires much computing resources and has the very fast growth of execution time with increasing of analyzing data.

Semantic search using ontologies [13], [14], [15] – the algorithm based on semantic network as indexing method. The advantages: wider context recognition than in other algorithms and the opportunity to execute search requests for terms with words, which did not appear in learning dataset. Nevertheless, there is no method to full-automatically extract whole ontologies from training corpus [16].

Doc2Vec [17], [18] – the algorithm, which uses neural network to create vectors from each document and cousin distance to compare them. The advantages: faster learning stage and search request execution than in other algorithms.

Full comparation between these algorithms is presented in Table I. Table legend: N – dictionary size, M – term size, D size of documents collection, k – matrix rank (LSA).

After comparation we have chosen Doc2Vec algorithm to search similar text information in our solution, because its advantages have been important for us [19], [20].

TABLE I. COMPARATION BETWEEN TEXT PROCESSING ALGORITHMS

Processing Algorithm	Learning Comple-xity	Search Comple-xity	Context Recogni-tion	Search recall
BM25	O(ND)	O(D*(N+ +M/2))	No	>30%
LSA	$O(ND)^{2k+1}$)	O(ND)	Yes	30-50%
Ontologies base	Manual	O(ND)	Yes	<50%
Doc2Vec	O(D(N+D) + +Dlog(N))	O(D* *log(N))	Yes	30-50%

III. APPROACH DESCRIPTION

A. System requirements

The key features of the project can be divided into three parts based on their outcome.

First key point is an ability to search similar customer requests in the database of already resolved issues.

The approach classifies customer's requests with machine learning algorithms (Doc2Vec) based on the history database issues and provide the list of semantically similar cases [21], the list of appropriate engineers and the list of reliable labels.

Semantically similar cases help to understand if the problem has already been resolved or give a quick overview on the taken approaches and steps to resolve the case. A section with appropriate engineers must allow responsible persons to contact with or ask for a help. Classification of cases by problematic area also this gives a possibility to identify the product component, that needs additional attention for improvements, bugs elimination or additional documentation.

Secondly, it will provide semantic search over the documentation [22] in addition to cases.

The result of the search is the table with mapping of the document name and page number, that contains semantically related information. This helps the engineer to quickly provide proper references to the customer or even to learn more about problematic component and confirm if the request is a bug or not.

The last feature is application of set of static rules for providing formal guidelines.

Based on the customizable templates, the systems suggest engineer to pass the case to proper state ("OPEN", "IN_PROGRESS", "INFORMATION REQUESTED", etc.) or to request additional information. For example, when customer complains about an error in product, but forget to attach a log file with stack trace.

All the approaches are implemented in software tool, which is called "Iresolver". It allows automate resolution of customer requests, which in turn allows to decrease the amount of time spent by technical support engineers and product developers on finding solution to the customer's requests. In Table II requirements for the solution are stated, functional (F) as well as non-functional (NF).

TABLE II. FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

ID	Description
F - 1	Software solution should be able to retrieve opened and closed issues from the following bug-tracking systems: JIRA, Bugzilla. • closed issues should be used for training machine learning algorithm; • for opened issues the recommendations should be generated.
F - 2	Software solution should be able to process documentation in PDF, TXT and set of confluence wiki pages formats in order to provide page number with semantically similar information to the opened request.
F - 3	Software solution should generate report for opened issues that contains recommendations about: • semantically similar cases; • references to documentation and pages; • persons to contact with; • case classification; • additional information to request and ask the customer.
F - 4	System should generate final report with recommendations in HTML and TXT formats.
F - 5	Software system should be configured with YML configuration file and contain list of properties for configuration: • url and credentials for bug tracking system; • filters for select closed and opened cases; • report format.
F - 6	When user launches the system, the system should retrieve the unresolved issues from the set Jira project, vectorize with the help of model, compare them to the resolved issues and documentation and provide the report on each issue.
F - 7	The system should have the following functionality for manual launch: create documentation dataset; • create documentation vector model; • create issues dataset; • create issues vector model; • create confluence dataset; • create confluence vector model.
F - 8	When customer runs "create issues dataset" command, system should retrieve the resolved issues from the set Jira

	project and provide the dataset.
NF - 0	System integration in customer's company business processes should be effective and fast with minimal expenses for end company.
NF - 1	System should provide a final issue report on the unresolved cases in final and reasonable time.
NF - 2	System should follow security rules and prevent data leaks when working with customer's data.
NF - 3	System should follow GDPR.

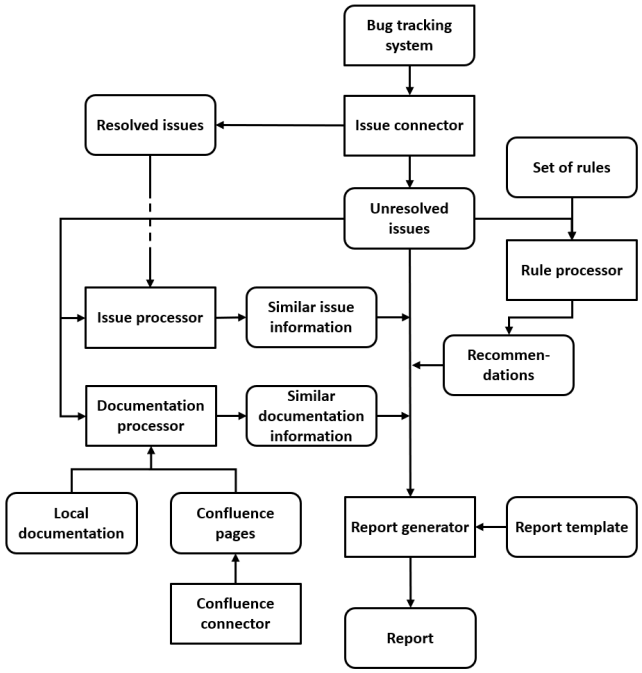

Fig. 1. System architecture

B. Overall system architecture

Main inputs of the system are:

- custom queries, which we get from bug tracking systems, such as Jira and Bugzilla, through issue connector.
- local documentation and pages from Confluence portal which are collected through documentation connector.
- set of rules, written by user and used by rules processor.
- report template, which is used by report generator.

In the course of using the system, we need to collect resolved and unresolved issues, documentation, create their respected vector models, use them in the trained algorithm/rules processor and retrieve information regarding similar issues, documentation and recommendations. The output of the system is a report, which contains all collected information.

Fig. 1 shows overall system architecture, main points of the execution, objects that are involved. Boxes with sharped angles contain software modules, with rounded – input or output data. Table III contains a description of elements in Fig. 1.

TABLE III. ELEMENTS DESCRIPTION IN FIG. 1

Box name	Meaning
Customer query	An existing bug report which contain information about issue
Bug tracking system	Software application or a portal in which bug reports are created and registered, the main purpose of it is to trace the process of solving the problem
Issue Connector	System module, in which needed issues are retrieved from bug tracking system with the help of filters
Unresolved issues	Collected issues with the Unresolved status, for which our system will find similar resolved issues using algorithm and rules
Resolved issues	Collected issues with the Resolved status, which are needed for creating vector model and teaching the Doc2Vec algorithm
Local documentation	Documentation files in pdf format that exist in a local folder
Confluence pages	Articles and documentation that exist on Confluence portal
Documentation connector	System module, in which documentation from local folder and Confluence pages is retrieved
Similar issue information	Result of Doc2Vec usage which contain information regarding resolved issues similar to unresolved ones, including links, appropriate engineers, possible labels
Similar documentation information	Result of Doc2Vec usage which contain information regarding documentation similar to unresolved issues, including page number and link
Set of rules	Local *.drl files which contain rules built by Drools template that can apply to issues and analyze their size, status and attachments, and write certain recommendations for the user
Rules processor	System module, in which rules are being retrieved and used
Recommendations	Natural language sentences, that were collected through rules being true for each issue
Report template	Apache Velocity template of the final report, which contains all collected results from previous modules
Report generator	System module, in which report template is filled with actual system results for each issue

C. Data flow model

Fig. 2 shows the general data flow in our system, what files are created, what processes are performing with the data.

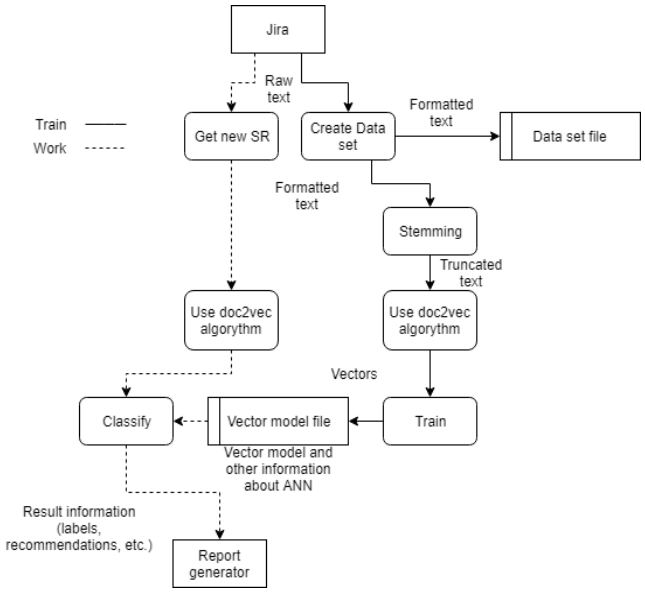

Fig. 2. Data flow model

From Jira/Bugzilla we get text that is transformed in dataset file [23]. Then that text is getting stemmed [24] and trained in Doc2Vec, the output is vector model file. The unresolved issues are collected as text, together with vector models are given to the classifier, which output is given to report generator. Its output is the final result.

III. Implementation

A. Software frameworks

After analyzing all requirements, we described common usage scenarios and interfaces and chose software frameworks for realizing main features. Table IV includes technologies that we used and their brief description and usage.

TABLE IV. Used technologies

Tool or framework	Description
Java 8	Main programming language. It combined adaptability for different OSs, wide functional of standard library and many existing external libraries and frameworks.
DeepLearning4j	The machine learning framework. We use its implementation of Doc2Vec algorithm to solve semantic search task.
ND4J	The part of DL4J. It makes possible to work with NDArrays and perform all common mathematical operations with them.
Drools	This framework allows to create special *.drl file with rules, which check one option of a processing issue. For example, length of description is not equal to zero.
Apache POI	The framework for parsing *.doc or other documentation files.
Jira Rest Java Client	The library for getting a list of Jira issues with REST API.
Bugzilla for Java	The library for getting a list of Bugzilla issues with REST API.
Confluence Rest Client	The library for getting wiki pages with REST API.
Logback	The logging framework. It has many options to configure log templates.

Software is developed on Java language version 8. It was the last version at the beginning of the development and now one of our tasks is update to Java 14. As IDE we use JetBrains IDEA 2019. For version control we have chosen Git, because it is commonly used by developers. All operations could be launched with batch scripts (.bat for Windows/.sh for Linux) and user can set tool configuration with special YML-file.

System consists of modules that can be classified as connectors, processors and reporters. Gradle is used in our project due to its easy configuration and building. This tool gives us options to add new modules or to update frameworks versions without any changes in the code.

B. Algorithmic components

We use Doc2Vec algorithm in the processor as our main algorithmic component. It's the innovative approach that we use for semantic search [25, 26]. The goal of the algorithm is to create numeric representation of the document. Doc2Vec is based on Word2Vec model with addition of paragraph vector. Doc2Vec uses two ideas for vectorized word representation: Distributed Memory (PV-DM) and Distributed Bag Of Words (DBOW). PV-DM is based on CBOW [27], which is used in Word2Vec. Model is learning to guess the word by its context. The main idea of PV-DM is that central word is guessed by input context words and paragraph id, as the model was previously taught on words from the paragraph. The other model of Doc2Vec algorithm is Distributed Bag Of Words (DBOW), which ignores the context of words and guess words from the input paragraph vector.

As for the documentation processing, the main component is Apache POI that works with text files that is used to create a documentation dataset.

As for the vector model that is formed and serialized into the archive, it contains the following files:

- codes.txt - a text file with codes for the Huffman tree;
- config.json - a text file in JSON format, which contains the settings of the Doc2Vec algorithm;
- frequencies.txt - a text file that contains the tf-idf and bag-of-words metrics for each word;
- huffman.txt - a text file with the coordinates of the points of the Huffman tree;
- labels.txt - a text file containing a list of identifiers for resolved requests. Each identifier is encoded in base64 format and is on a separate line;
- syn0.txt - a text file that contains a word encoded in base64 format and its weights on each line that make up a unique numerical vector. The numbers of weights are real and their size is from -1 to 1. This file contains the weights of the connections between input and hidden neurons in the neural network, which uses the H-Softmax [28] function as an activation function;
- syn1.txt - a text file that contains a weight matrix for hierarchical osoftmax. This file contains the weights of the connections between the hidden and output neurons in the neural network, which uses the H-Softmax function as an activation function. After learning the algorithm, you can start using it. To do this, the model from the VectorModel.zip file is loaded into the memory, in which each document is presented as a numerical vector and is mapped to a request key. After that, a set of JiraIssue objects that represent unresolved requests is supplied to the input of the classifier. Text data is extracted from these objects and fed to the input of the Doc2Vec algorithm.

C. User Interfaces

One of the interfaces is a configuration file. User can set preferences in configuration file for system to tune performance of the system. For example, sets preferences for connector, what information to retrieve and from where, tunes Doc2Vec algorithm, sets paths for all input/output files needed.

Another one interface is the command line interface. User launches application through the console with the commands or scripts with the same name, for example:

- create-issues-data-set – creates resolved issues dataset;
- create-issues-vector-model – creates vector model of the resolved issues dataset;

- create-documentation-data-set – creates local documentation dataset;
- create-documentation-vector-model – creates vector model of the local documentation dataset;
- create-confluence-data-set – creates Confluence pages dataset;
- create-confluence-vector-model – creates vector model of the Confluence pages dataset;
- run – retrieves unresolved issues, vectorizes them, compares to the resolved issues and documentation, gives the final report.

The last user interface is browser with the final report in which there are numbers of similar issues, appropriate engineers, links to helpful documentation, ID of helpful confluence page and suggestions on further actions.

The most preferred work mode is distributive. It consists of the following parts:

- bin – folder with bat/sh scripts for launching all user task;
- config – folder with system and logging configuration file, report template;
- data – folder which stores output dataset and vector models of all modules;
- lib – folder which contains source code and "third-party" libraries;
- logs – folder with logged information;
- output – folder which contains created final visual and text report;
- rules – folder with static rules description.

D. Hardware requirements

Software solution should launch on modern PCs. Minimum hardware requirements for the solution to run are:

- 64-bit versions of Microsoft Windows 10, 8, 7 (SP1);
- 2 GB RAM minimum, 8 GB RAM recommended;
- 2.5 GB hard disk space, SSD recommended;
- 1024x768 minimum screen resolution.

IV. RESULTS

A. Testing method

Testing of our product generally consists of two parts – code testing and system testing.

Code testing included designing, implementation and running of unit and integration tests of different modules of our application. Tests were developed with Junit testing tool and Mockito framework that provides mock-objects implementation. All tests written for the project are stored in Git repo.

Also, we follow ci/cd ideas and provide test automation and code lining. All stages of code testing are executed with Travis pipelines after each commit of any developer. It allows us to make our code more stable and safer.

End-2-end testing were done on Apache Kafka – real project provided with JIRA bug-tracking system. iResolver was deployed on Laptop Lenovo Thinkpad T560 with RAM 16GB and HDD 500GB. Deployment on personal computer is the only way of deployment we offer for now.

```
connectors:
  jira:
    url: https://issues.apache.org/jira
    anonymous: true
    credentialsFolder: ../data/
    resolvedQuery: project = Kafka AND resolution != Unresolved
    unresolvedQuery: project = Kafka AND resolution = Unresolved
    batchSize: 100
    batchDelay: 0
    resolvedIssueFields:
      - summary
      - description
    unresolvedIssueFields:
      - "*all"

  confluence:
    url: https://cwiki.apache.org/confluence
    anonymous: true
    credentialsFolder: ../data/
    spaceKeys:
      - KAFKA
    limitPerRequest: 500
```

Fig. 3. Connector module configuration

There are 7096 unresolved requests, 799 confluence pages and 1043 Kb of documentation in this project. Learning time was 8 min. 31 sec. After this step, the tool created about 140 Mb artifact data. Summary report creation time was about 1 min. An example of connector module configuration used in testing is shown on Fig. 3. Fig. 4 is showing an example of final report for one of the Kafka issues.

B. Results

Developed system has been used to compare the average time of finding necessary information for processing issue using semi-automated approach and the average time of manual gaining information for issue resolution.

As we can see from Fig. 5, the average time to find the proper information using the automated approach was 12.2 minutes, and the average time spent with the manual approach was 18.4 minutes. Therefore, decrease in the complexity is approximately 33%.

When engineer performs manual search, they need to search all resolved issues by keys and entering filters, then look through each issue to see if it matches. Regarding documentation, here user should search by keywords in the whole document body and analyze paragraphs in relation to unresolved issues. This process is not effective time wise so automation of it lifts a part of the task off engineers.

V. CONCLUSION

During this project, we developed a working prototype of a software solution.

Distinctive features of our solution include:

- usage of an approach based on the Doc2Vec algorithm to identify semantically similar customer requests;
- usage of semantic search on documentation and wiki pages;
- usage of confluence wiki pages as a source for documentation;
- usage of rule-based engine to generate formal recommendations to the user, which contain natural

language sentences that provide user with advises on whether to ask for additional information about the issue, missing files, changing the status of the issue or assign it to the appropriate engineer.

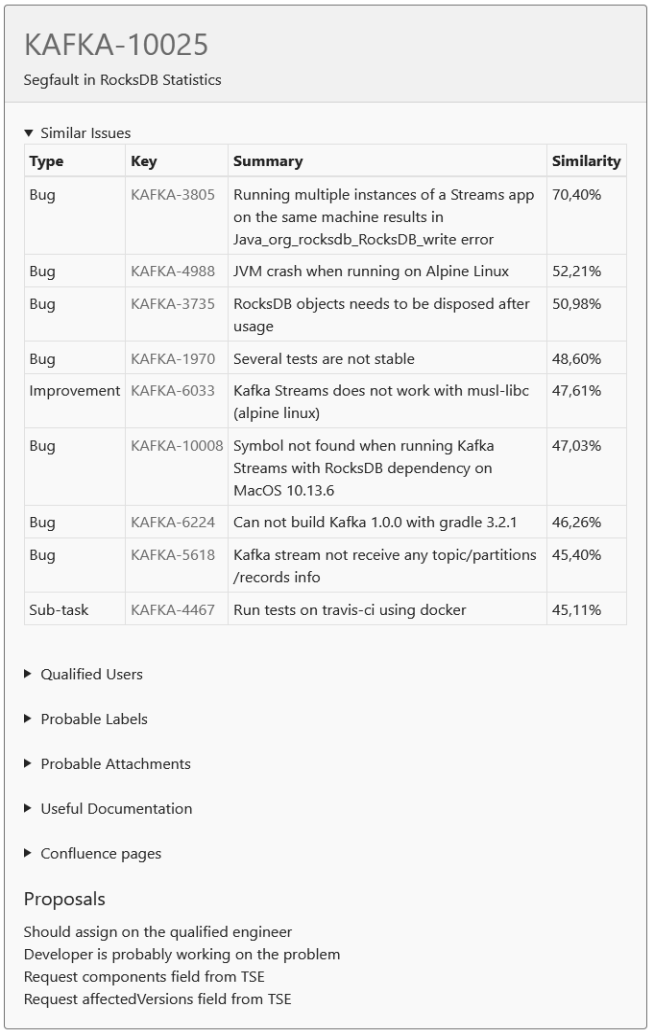

Fig. 4. Report example

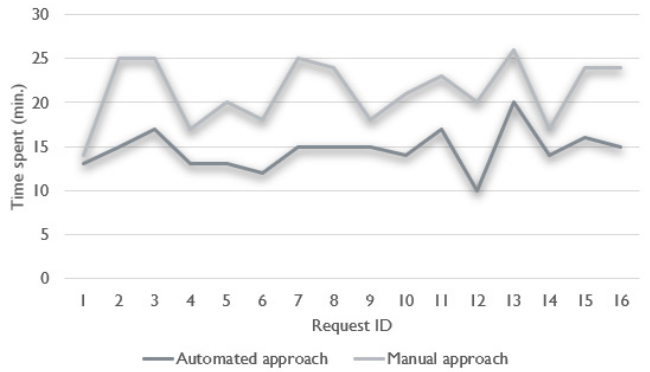

Fig. 5. Time difference between manual and automated approaches

The research helped us to get the first feedback from other engineers and execute a testing on the real data. Moreover, we researched different technologies developed in area of automation and machine learning. Analysis of it changed the vision of our project and proposed a new development path. We have made some points of future work:

First idea is to develop other deployment approaches based on containerized architecture of applications. Kubernetes yaml-file deployment description and Helm chart design are future areas of discussion.

Our IResorver tool could be provided with connectors for other bug-tracking tools such as YouTrack, Wrike, Slack or GitHub.

The process of research of different machine learning libraries and frameworks brought us to eli5 library for Python apps that visualize the results of text classification, including the reports of Doc2Vec implementation. Such highlighting instrument is not built in deeplearning4java, so the analogue of this instrument could be designed for iResolver project to provide more intuitive demonstration of issues classification.

The last point to consider is that Docker-based architecture would provide new abilities for integration and end-to-end testing.

ACKNOWLEDGEMENT

The research was funded as a part of the state assignment for basic research (a code of the research is 0784-2020-0026).

REFERENCES

[1] V. Kotlyarov, P. Drobintsev, N. Voinov, I. Selin, A. Tolstoles, “Technology and Tools for Developing Industrial Software Test Suites Based on Formal Models and Implementing Scalable Testing Process on Supercomputer”, *in Proc. Tools and Methods of Program Analysis (TMPA)*, Dec. 2017, pp. 51-63.

[2] E.E. Ogheneovo, “On the relationship between software complexity and maintenance costs”, *Journal of Computer and Communications*, vol. 2, 2014, p. 1.

[3] O. V. Mamoutova, M. B. Uspenskiy, A. V. Sochnev, S. V. Smirnov and M. V. Bolsunovskaya, "Knowledge Based Diagnostic Approach for Enterprise Storage Systems", *in Proc. IEEE 17th International Symposium on Intelligent Systems and Informatics (SISY)*, 2019, pp. 207-212.

[4] Y.B. Leau, W.K. Loo, W.Y. Tham, and S.F. Tan, “Software development life cycle AGILE vs traditional approaches”, *International Conference on Information and Network Technology*, vol. 37, no. 1, 2012, pp. 162-167.

[5] J. Fisher, D. Koning, and A.P. Ludwigsen, “Utilizing atlassian jira for large-scale software development management”, *in 14th International Conference on Accelerator & Large Experimental Physics Control Systems (ICALEPCS)*, Oct. 2013.

[6] D. Bertram, A. Voida, S. Greenberg, and R. Walker, “Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams”, *in Proc. 2010 ACM Conf.*, Feb. 2010, pp. 291-300.

[7] Automation Lite for Jira, Web: https://marketplace.atlassian.com/apps/1211836/automation-lite-for-jira.

[8] IPsoft Amelia, Web: https://www.ipsoft.com/amelia.

[9] V.I. Gorodetsky, O.N. Tushkanova, “Semantic Technologies for Semantic Applications. Part 1. Basic Components of Semantic Technologies”, Scientific and Technical Information Processing, vol. 46, 2019, pp. 306-313.

[10] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford, “Okapi at TREC-3”, *in Proc Text Retrieval Conf. (TREC 1994)*, 1994, p. 19.

[11] J. Ramos, “Using tf-idf to determine word relevance in document queries”, *in Proc. Machine Learning First Instructional Conf.*, vol. 242, Dec. 2003, pp. 133-142.

[12] T. Landauer, P. W. Foltz, and D. Laham, “Introduction to Latent Semantic Analysis”, *Discourse Processes*, vol. 25, 1998, pp. 259-284.
[13] J. F. Sowa, “Top-level ontological categories”, *International Journal of Human-Computer Studies,* vol. 43, Nov. 1995, pp. 669-685.
[14] D. Baryev, I. Konovalov, N. Voinov, “New Approach to Feature Generation by Complex-Valued Econometrics and Sentiment Analysis for Stock-Market Prediction”, Cyber-Physical Systems and Control, Nov. 2019, pp. 573-582.
[15] E.N. Desyatirikova, A. Osama, V.E. Mager, L.V. Chernenkaya, A.S. Ahmad, “Enhancing the Performance of Reservation Systems Using Data Mining”, Cyber-Physical Systems and Control, Nov. 2019, pp. 413-421.
[16] A.O. Aleksyuk, V.M. Itsykson, “Semantics-Driven Migration of Java Programs: A Practical Application”, Automatic Control and Computer Sciences, vol. 52, pp. 581-588.
[17] N. Maslova and V. Potapov, “Neural network Doc2Vec in automated sentiment analysis for short informal texts”, *Lecture Notes in Computer Science,* vol. 10458, 2017, pp. 546-554.
[18] A. Kovalev, N. Voinov, I. Nikiforov, “Using the Doc2Vec Algorithm to Detect Semantically Similar Jira Issues in the Process of Resolving Customer Requests”, *Studies in Computational Intelligence*, vol. 868, 2020, pp. 96-101.
[19] L. Kang, “Automated Duplicate Bug Reports Detection”, Blekinge Institute of Technology, 2017, p. 79.
[20] D. Kim, D. Seo, S. Cho, P. Kang, “Multi-co-training for document classification using various document representations: TF–IDF, LDA, and Doc2Vec”, *Information Sciences*, vol. 447, Mar. 2019, pp. 15-29.
[21] L. Hiew, “Assisted detection of duplicate bug reports”, University of British Columbia, 2006.
[22] A. Sokolova, M. Solomonov, A. Kovalev, I. Nikiforov, “The semantic search in documentation in automated customer issue solving system”, *in Proc. SPbPU Science Week*, 2019, pp. 87-90, (in Russian).
[23] A.H. Branco and J.R. Silva. “Contractions: breaking the tokenization-tagging circularity”, *Lecture Notes in Computer Science*, vol. 2721, 2003, pp. 167-170.
[24] D. Sharma, “Stemming algorithms: A comparative study and their analysis”, *International Journal of Applied Information*, 2012.
[25] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, “A discriminative model approach for accurate duplicate bug report retrieval”, *in Proc. 32nd ACM/IEEE International Conf.*, 2010, pp. 45–54.
[26] A.T. Nguyen, T.T. Nguyen, T.N. Nguyen, D. Lo, and C. Sun, “Duplicate bug report detection with a combination of information retrieval and topic modeling”, *in Proc. 27th IEEE/ACM International Conf.*, 2012, pp. 70–79.
[27] G.L Giller, “The Statistical Properties of Random Bitstreams and the Sampling Distribution of Cosine Similarity”, *Giller Investments Research Notes*, no. 20121024/1, 2012.
[28] T. Pellegrini, “Comparing SVM, Softmax, and shallow neural networks for eating condition classification”, *in Proc. 16th Annual ISCA Conf.*, 2015, pp. 899-903.