Санкт-Петербургский государственный университет

Направление Математическое обеспечение и администрирование информационных систем
Кафедра системного программирования

Сокольвяк Сергей Дмитриевич

Разработка плагина IntelliJ IDEA для поиска релевантных дискуссий с Q&A сайтов

Курсовая работа

Научный руководитель:
к. т. н., доцент Брыксин Т. А.

Санкт-Петербург
2019

Оглавление

Введение

В настоящее время контекстная помощь в IDE актуальная и популярная задача для разработчиков. Это обусловлено тем, что при написании кода зачастую возникают вопросы, ответы на которые приходится искать в интернете, переключая свое внимание на окно веб-браузера. Большинство современных решений уже встроены в IDE и предоставляют различные виды контекстной помощи, такие как автодополнения, подсказки, подсветку синтаксиса.

Однако, несмотря на достаточную эффективность существующих решений, они помогают решать далеко не все задачи. Ответы на многие другие вопросы о программировании размещены на так называемых Q&A (Question & Answer, Вопрос – ответ) сайтах. Одним из наиболее популярных Q&A сайтов является StackOverflow[1].

В настоящее время существует ряд распространенных решений, предоставляющих пользователю ответы со StackOverflow, однако многие из этих решений основаны на bag-of-words модели данных[2] и имеют ряд общих проблем. Анализ контекста (кода в окне IDE) весьма нетривиальная задача. Для формирования ответа с дискуссией StackOverflow, необходимо на основе кода, написанного на языке программирования, понять смысл вопросов, которые могли возникнуть у пользователя. Поэтому многие из существующих решений далеко не всегда способны отобразить дискуссии, которые действительно могут быть полезны пользователю. Кроме того, большинство из этих решений разработаны для Eclipse IDE и не поддерживаются IntelliJ IDEA.

Популярность Q&A сайтов, в частности StackOverflow, послужила толчком к разработке плагина для IntelliJ IDEA, который помог бы программистам сократить время обращения к таким сайтам. А ряд проблем в текущих решениях обязует при формировании контекстной помощи учитывать не только синтаксис, но и семантику кода.

[1]https://stackoverflow.com/

[2]Модель *"мешок слов"* – это представление данных, используемое при обработке естественного языка и поиске информации. В этой модели текст представляется как набор слов и чисел, характеризующих частоту повторений соответствующих слов.

Постановка задачи

Целью данной работы является разработка плагина IntelliJ IDEA для поиска и отображения релевантных дискуссий StackOverflow. Данная работа основывается на проекте Context Helper[3]. В этом проекте разработана начальная архитектура, на базе которой можно продолжить дальнейшую разработку плагина. Для достижения поставленной цели были выделены следующие задачи:

1. Изучить SDK платформы IntelliJ.
2. Сделать обзор существующих решений и идей, которые можно использовать для расширения функционала текущего прототипа Context Helper.
3. Реализовать функцию автоматической контекстной помощи при возникновении ошибок компиляции или времени выполнения.
4. Добавить меню настроек для включения или отключения функции автоматической контекстной помощи при возникновении ошибок.

[3]https://github.com/ml-in-programming/context-helper/

1. Обзор существующих решений

1.1. Prompter

Prompter [5] – это совместная разработка группы ученых из Италии и Щвейцарии, представленная в 2014 году, реализованная в виде плагина для Eclipse IDE.

Prompter, работая в фоновом режиме, собирает информацию о всех изменениях, которые происходят в коде. Анализируя полученную информацию, Prompter постепенно формирует запрос. После того, как запрос сформирован, пользователь получает уведомление и самые релевантные дискуссии, соответствующие данному запросу, в отдельном окне в IDE.

За формирование запроса отвечает сущность Query Generation Service. Сформированный запрос передается сущности Search Service, которая извлекает id необходимых дискуссий. А полученные id используются для того, чтобы извлечь дискуссии StackOverflow, которые затем отображаются пользователю. Рисунок 1 демонстрирует архитектуру Prompter.

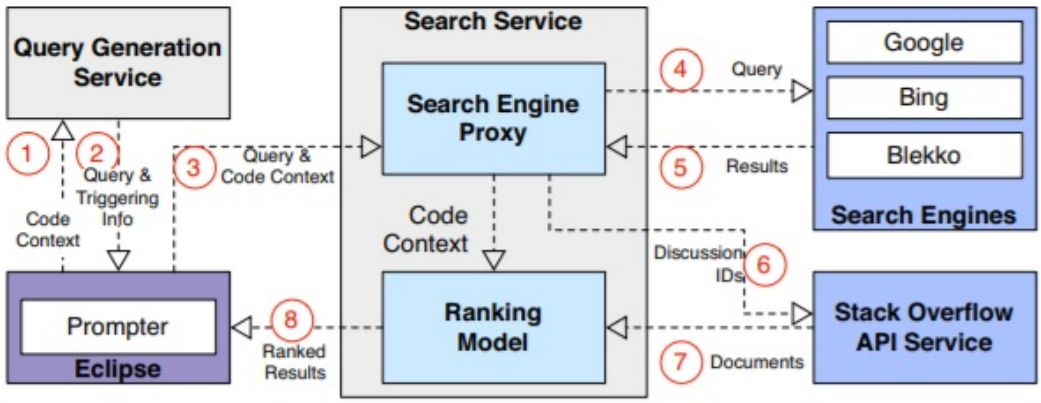

Рис. 1: Архитектура Prompter

При реализации Prompter разработчики использовали усовершенствованный метод bag-of-words [4]. Суть данного метода состоит в следующем:

1. Определяется элемент, который был только что изменен (это может быть класс или метод).

2. Излекается контекст, содержащий измененный элемент (это может быть packageName.ClassName для классов
или packageName.ClassName.methodSignature для методов).

3. К нему добавляется информация о типах и методах использованного API и выбрасываются stop-words[4].

4. Из полученной информации формируется словарь, где каждое слово — *термин.*

5. На основе полученных данных формируется вектор, где каждая координата отвечает одному термину (слову) из контекста, а каждое значение характеризует частоту появления термина в контексте.

6. Для каждого термина в векторе по формуле (1) вычисляется его *энтропия*:

$$E_t = \sum_{d \in D_t} p(d) \cdot log_\mu p(d), \tag{1}$$

где t – термин, E_t – *энтропия* термина, D_t – множество всех StackOverflow дискуссий, содержащих термин t, μ – общее количество дискуссий StackOverflow, p(d) - вероятность того, что случайный термин t входит в дискуссию d. А по формуле (2) его *индекс качества*:

$$TQI_t = v_t \cdot (1 - E_t), \tag{2}$$

где t – термин, TQI_t – *индекс качества* термина t, v_t – частота термина t в текущем контексте, E_t – *энтропия* термина t.

7. На основе терминов с наибольшим *индексом качества* извлекаются StackOverflow дискуссии, которые затем отображаются пользователю.

[4]Стоп-слова — слова, которые являются общими для всех документов поисковой коллекции и бесполезными при поиске какой-либо конкретной информации. К стоп-словам можно отнести предлоги, причастия, междометия, цифры, частицы.

Плюсом данного подхода является рост качества сформированных запросов и релевантности отображаемых пользователю дискуссий в сравнении с обычным bag-of-words методом. Однако введение дополнительных метрик частично решает проблему несоответствия частоты слова к его значимости, но не учитывает порядок слов и их связь. Минусом предложенного подхода является необходимость наличия базы, в которой содержались бы актуальные значения *энтропии* терминов. По состоянию на 2014 год, такая база насчитывает 105439 различных терминов, содержащихся во всех дискуссиях StackOverflow. Очевиден факт, что с каждым годом количество терминов в этой базе только увеличивается, а значения *энтропии* придется регулярно пересчитывать, иначе они потеряют свою актуальность.

1.2. NLP2Code

NLP2Code – решение, представленное в 2017 году группой австралийских ученых и реализованное в виде плагина для Eclipse IDE [1].

NLP2Code позволяет пользователю вставлять отрывки кода с StackOverflow прямо в свой код по нажатию комбинации клавиш, не покидая редактор исходного кода. Рисунок (рис. 2) показывает работу NLP2Code, если пользователь ввел *"split string by"*. Кроме того, пользователь может переключаться между доступными отрывками кода по нажатию другой комбинации клавиш с целью найти наиболее подходящий.

Разработчики NPL2Code использовали так называемый TaskNav алгоритм [3], основанный на task-based подходе. Суть данного подхода заключена в разбиении документации по коду (в случае NLP2Code это все дискуссии StackOverflow с тегом "java") на темы-действия, которые представляют собой глагол с прямым объектом или фразу, например, *"get iterator"*, *"get iterator for collection"*, *"add to collection"*. Исходя из пользовательского запроса, отображаются подходящие темы-действия и самые релевантные дискуссии удовлетворяющие этим темам-действиям.

Плюсом такого подхода является высокая эффективность сформи-

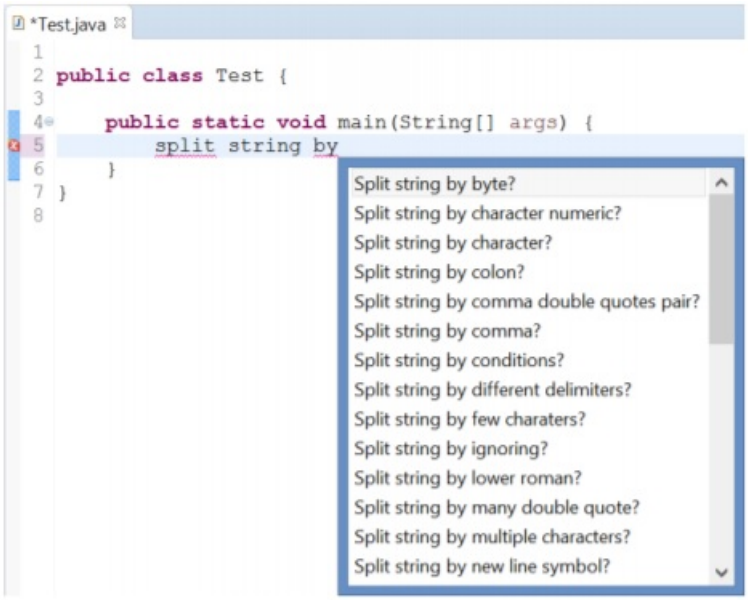

Рис. 2: Пример работы NLP2Code

рованных ответов. Согласно данным, которые приводят разработчики, 74% подсказок были отмечены командой пользователей как полезные. Минусом такого подхода является то, что для получения необходимых дискуссий требуется определить темы-действия на естественном языке. NLP2Code не анализирует программный код пользователя в IDE, а ожидает от него запрос, сформированный на ествественном языке, такой как *"add lines to"* или *"split string by"*.

1.3. StackInTheFlow

StackInTheFlow – это решение, представленное в 2018 году группой ученых из США и реализованное в виде плагина для IntelliJ IDEA [2].

StackInTheFlow генерирует запросы и возвращает пользователю посты StackOverflow с рекомендациями в случае возникновения трудностей, определяемых ошибками компиляции или времени выполнения или отсутствием прогресса в коде редактора. StackInTheFlow имеет следующий набор характеристик:

1. Автоматическое построение запросов, основанных на текущем контексте в окне редактора исходного кода;

2. Вывод постов StackOverflow с рекомендациями при возникновении ошибок на этапе компиляции или во время выполнения;

3. Ручной ввод запросов в окне плагина для получения самых актуальных постов StackOverflow, соответствующих данному запросу;

4. Персонализация результатов на основе истории кликов пользователя.

Механизм генерации запроса на основе выделеного контекста основан на bag-of-words модели данных и заранее сформированном словаре терминов, извлеченных из постов пользователей StackOverflow. Каждый термин в словаре имеет *значимость*, которая измеряется по мере *it-idf*[5]. Для подсчета *tf(term frequency – частота слова)* используется формула:

$$tf(t,d) = \frac{n_t}{\sum_k n_k}, \tag{3}$$

где n_t – число вхождений слова t в документ d, а в знаменателе – общее число слов в данном документе. Для подсчета *idf(inverse document frequency – обратная частота документа)* используется формула:

$$idf(t,D) = log\frac{|D|}{|\{d_i \in D | t \in d_i\}|}, \tag{4}$$

где $|D|$ – общее число документов в коллекции D, число документов из коллекции D, в которых встречается слово t (когда $n_t \neq 0$). Таким образом мера *tf-idf* является произведением двух сомножителей:

$$tf - idf(t,d,D) = tf(t,d) \times idf(t,D), \tag{5}$$

Согласно данным, собраным разработчиками на основе результатов использования плагина 77-ю уникальными пользователями, лишь 17% ответов на вручную сформированные запросы и 6% ответов на автоматические сформированные оказались полезными для пользователей.

[5] *TF-IDF* – статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов или корпуса. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции.

На основе полученных данных можно сделать вывод о малой эффективности описанного выше механизма автоматической генерации запросов и необходимости использования других алгоритмов формирования запроса на основе контекста.

1.4. Прототип плагина Context Helper

В 2017 году началась разработка плагина контекстной помощи Context Helper[6], и на сегодня существует прототип, способный помочь программистам работающим в IntelliJ IDEA. Создание плагина Context Helper мотивировано теми же причинами, что и создание других решений – плагин призван сократить время, которое программисты тратят на обращение к ресурсу StackOverflow путем интеграции подсказок непосредственно в окно IDE. Однако в отличие от NLP2Code, идея Context Helper заключена в том, что он не вставляет куски чужого кода, а показывает дискуссию целиком. При таком подходе пользователь сам решает, каким образом использовать полученный ответ.

На рисунке (рис. 3) показан принцип работы прототипа Context Helper. Нажатие комбинации клавиш приводит в действие сущность под названием ContextExtractor, которая извлекает из контекста ключевые слова и формирует запрос. GoogleCustomSearch Client исполняет сформированный запрос и получает ids постов StackOverflow, удовлетворяющих этому запросу. Сущность StackExchangeAPI Client использует полученные ids и извлекает посты StackOverflow, которые затем отображаются пользователю.

Сущность ContextExtractor, которая отвечает за извлечение ключевых слов из контекста и формирование запроса, использует три различных алгоритма, основанных на использовании возможностей PSI (Program Structure Interface) [7]. Program Structure Interface позволяет на основе кода в редакторе строить иерархическую структуру – дерево PSI элементов. В роли PSI элементов выступают классы, поля,

[6] https://github.com/ml-in-programming/context-helper/

[7] https://www.jetbrains.org/intellij/sdk/docs/basics/architectural_overview/psi.html

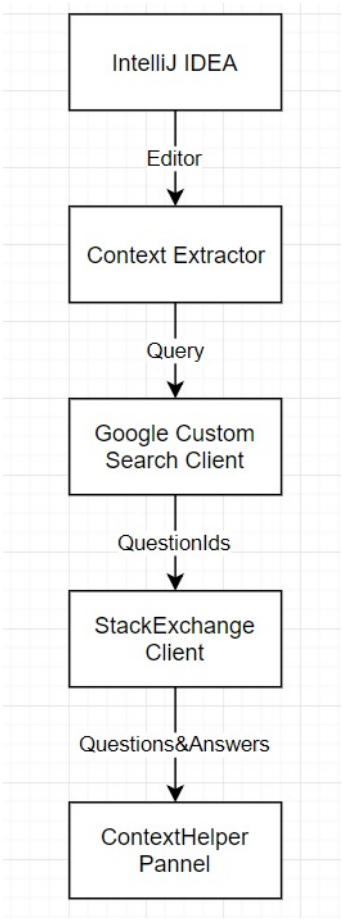

Рис. 3: Принцип работы прототипа Context Helper

методы, переменные и т.п. Такое представление удобно для анализа исходного кода, позволяет получать информацию о связях объектов, их типах и ссылках на другие объекты. Первый алгоритм в качестве текста запроса использует весь исходный код PSI элемента, на котором в момент вызова плагина стоит курсор. Второй алгоритм в качестве ключевых слов для формирования запроса использует наиболее популярные в контексте типы, при этом учитывается степень вложенности типа по отношению к курсору. Третий алгоритм строит запрос на основе использования методов и пользовательских типов.

2. Реализация

2.1. Механизм автоматической контекстной помощи при возникновении ошибок

Для того чтобы реализовать механизм автоматической контекстной помощи при возникновении ошибок компиляции или времени выполнения, необходимо реализовать несколько интерфейсов, предоставляемых IntelliJSDK. Рисунок 4 демонстрирует архитектуру механизма. Синим выделены классы и интерфейсы IntelliJSDK.

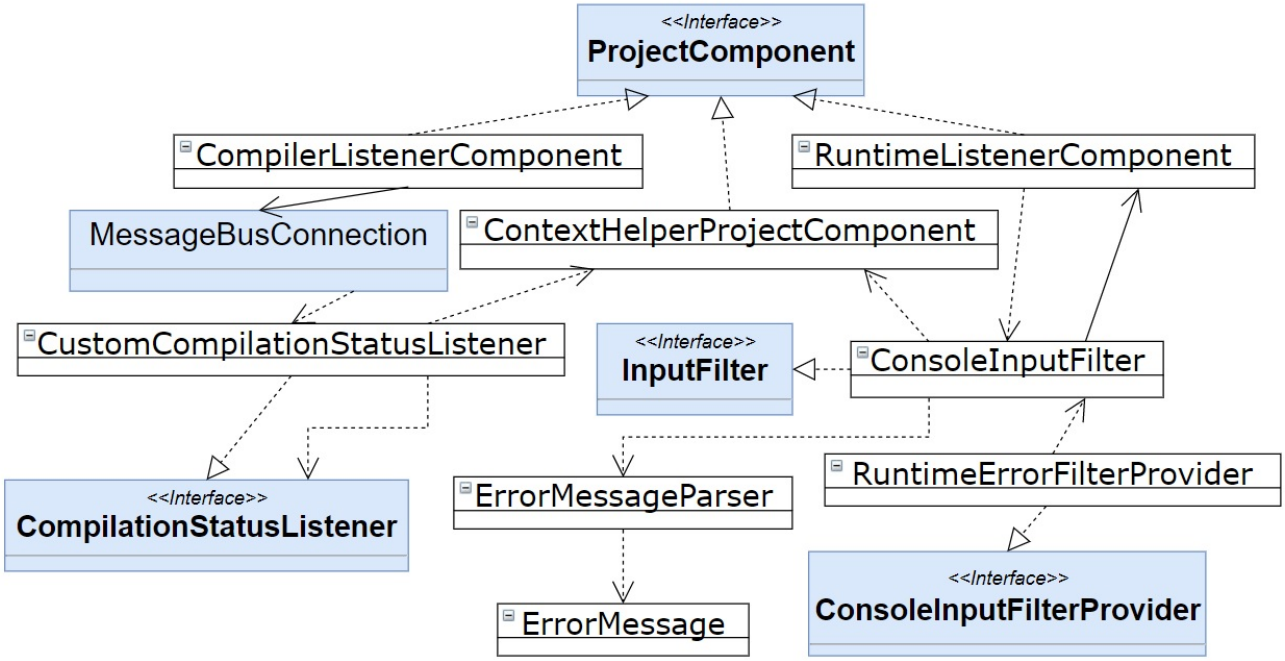

Рис. 4: Архитектура механизма обработки ошибок

IntelliJSDK предоставляет разработчикам возможность самостоятельно определить класс, отвечающий за обработку сообщений от компилятора. На рисунке 4 это класс CustomCompilationStatusListener. При получении от компилятора сообщений об ошибках этот класс обрабатывает сообщения и вызывает формирование запроса. Для того чтобы назначить CustomCompilationStatusListener обработчиком сообщений от компилятора, необходимо использовать класс MessageBusConnection. Его метод subscribe позволяет подписаться на сообщения компилятора и

назначить обработчик этих сообщений.
IntelliJSDK предоставляет разработчикам самостоятельно определять фильтры, которые применяются к сообщениям в окне консоли. Использование таких фильтров позволяет определять поведение IDE при обнаружении определенных сообщений в окне консоли, в том числе ошибок времени выполнения. Мы ограничились одним таким фильтром – класс ConsoleInputFilter. Для того чтобы использовать этот фильтр необходимо определить класс RuntimeErrorFilterProvider, который передает IDE массив реализованных фильтров. Сообщения об ошибках и служебная информация хранится в экземлярах класса ErrorMessage, а за обработку текста сообщений отвечает класс ErrorMessageParser. Классы, реализующие интерфейс ProjectComponent, используются для инициализации отдельных частей плагина. Экземляры таких классов инициализируются IDE для каждого нового проекта при открытии, а уничтожаются при закрытии этого проекта.

2.2. Сохранение пользовательских настроек

IntelliJSDK предоставляет разработчикам возможность сохранять различную служебную информацию на диске, например пользовательские настройки. Для этого необходимо реализовать несколько интерфейсов IntelliJSDK. Рисунок (рис. 5) демонстрирует диаграмму классов сохранения настроек. Синим выделены интерфейсы IntelliJSDK. За сохранение и извлечение пользовательских настроек из файла на диске отвечает класс PersistentStateSettingsComponent, который сохраняет эти настройки в экземлярярах класса State. Класс
ContextHelperConfigurable, реализующий интерфейс Configurable, добавляет в меню настроек IDE дополнительную вкладку с настройками плагина Context Helper. ContextHelperConfigurableGUI предоставляет графический интерфейс настроек плагина.

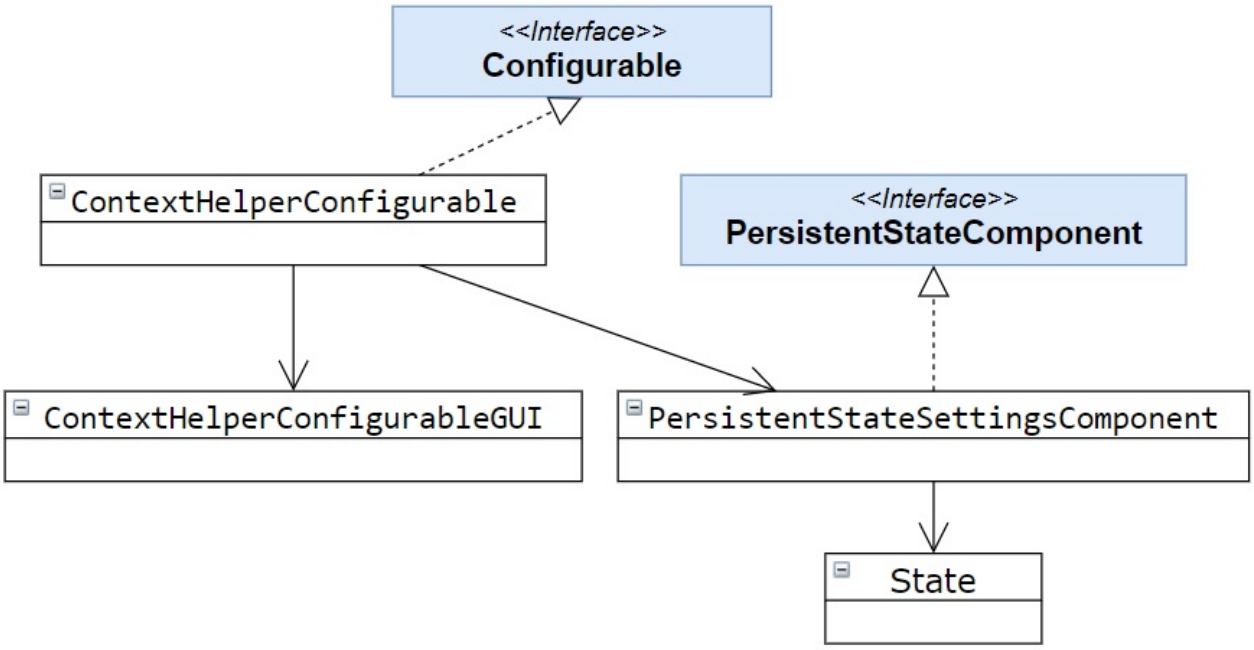

Рис. 5: Архитектура механизма настройки

Заключение

Выполненные задачи:

1. Сделан обзор современных решений NLP2Code, Prompter, StackInTheFlow. Изучены статьи, в которых описаны алгоритмы и подходы, ипользованные в данных решениях.

2. Сделан обзор текущего прототипа плагина Context Helper, изучен принцип его работы.

3. Реализован механизм автоматической контекстной помощи при возникновении ошибок компиляции и времени выполнения.

4. Добавлена вкладка в меню настроек для включения и отключения механизма обработки ошибок.

Список литературы

[1] B. A. Campbell C. Treude. NLP2Code: Code Snippet Content Assist via Natural Language Tasks // Proceedings of the 2017 International Conference on Software Maintenance and Evolution. — 2017. — P. 628–632.

[2] C. Greco T. Haden K. Damevski. StackInTheFlow: behavior-driven recommendation system for stack overflow posts // Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. — 2018. — P. 5–8.

[3] C. Treude M. Sicard M. Klocke, Robillard M. TaskNav: Task-Based Navigation of Software Documentation // Proceedings of the International Conference on Software Engineering. — 2015. — Vol. 2. — P. 649–652.

[4] L. Ponzanelli G. Bavota M. D. Penta R. Oliveto, Lanza M. Mining StackOverflow to turn the IDE into a self-confident programming prompter // Proceedings of the 11th Working Conference on Mining Software Repositories. — 2014. — P. 102–111.

[5] L. Ponzanelli G. Bavota M. D. Penta R. Oliveto, Lanza M. Prompter: A Self-confident Recommender System // Proceedings of the International Conference on Software Maintenance and Evolution. — 2014. — P. 577–580.