Санкт-Петербургский государственный университет

Кафедра системного программирования

Кита Михаил Евгеньевич

Ассистент для разработчиков в среде IntelliJ IDEA

Бакалаврская работа

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

Рецензент:
руководитель направления “Машинное обучение и анализ данных”,
ООО “Интеллиджей Лабс” А. А. Шпильман

Санкт-Петербург
2018

SAINT-PETERSBURG STATE UNIVERSITY

Software Engineering

Mikhail Kita

Developer assistant for IntelliJ IDEA

Graduation Thesis

Scientific supervisor:
Associated Prof. T. A. Bryksin

Reviewer:
Head of Machine Learning and Data Analysis Department,
IntelliJ Labs Co. Ltd. A. A. Shpilman

Saint-Petersburg
2018

Оглавление

Введение

С каждым годом сложность разработки программных продуктов неуклонно возрастает, усложняются проекты и задачи. В связи с этим всё более востребованными становятся инструменты, облегчающие разработку и повышающие производительность труда программистов. Современные среды разработки (Integrated Development Environment, IDE) включают множество различных инструментов: средства автодополнения, исправления ошибок, автоматического рефакторинга и многие другие. Они призваны сделать процесс разработки более эффективным путём автоматизации и предоставления различного рода подсказок.

Но часто существующих инструментов оказывается недостаточно. Разработчики, особенно начинающие, зачастую не обладают необходимыми знаниями для работы с API (Application Programming Interface) различных библиотек и фреймворков. Встроенные средства IDE способны показать лишь список возможных методов, но не пример их использования. Поэтому приходится покидать среду разработки и искать нужную информацию в других источниках, таких как GitHub[1] и Stack Overflow[2]. Однако частые переключения между различными окнами нежелательны, поскольку они негативно сказываются на продуктивности разработчиков [1, 2].

Решить данную проблему можно, если полезная информация будет доступна *непосредственно в среде разработки*. Это позволит повысить продуктивность работы, а также использовать дополнительные данные, недоступные вне IDE. Собранный таким образом контекст может улучшить релевантность (т.е. соответствие) предлагаемой информации текущим задачам пользователя [3].

Выделяют два принципиальных способа получения полезной информации: генеративный и ранжирующий. Первый способ заключается в преобразовании некоторого формального описания задачи в полезные данные. Примером могут служить системы для автоматического созда-

[1] `https://github.com/`

[2] `https://stackoverflow.com/`

ния исходного кода, решающего поставленную задачу [4]. Второй способ предполагает поиск, оценку и переиспользование подходящих данных, ранее созданных другими людьми. Он лежит в основе различных рекомендательных систем [5, 6].

На текущем уровне развития технологий второй подход кажется более предпочтительным из-за универсальности. Он позволяет предоставлять решения типичных задач, возникающих в работе программистов, по описанию на естественном языке, может быть применён для разных языков программирования, а также не налагает дополнительных ограничений на форму представления полезной для пользователя информации.

Существующие на сегодняшний день реализации систем, использующих данных подход [7, 8], недостаточно удобны и функциональны. Для некоторых платформ, например, IntelliJ IDEA[3], расширений с указанной функциональностью совсем немного, а качество их рекомендации оставляет желать лучшего.

Постановка задачи

Целью данной работы является создание расширения для IntelliJ IDEA, предоставляющего пользователю подсказки и релевантные фрагменты кода из различных источников. Для достижения поставленной цели были сформулированы следующие задачи:

- провести анализ предметной области и существующих решений;
- реализовать алгоритм рекомендации полезной информации;
- разработать архитектуру инструмента;
- реализовать пользовательский интерфейс;
- провести апробацию разработанного решения.

[3] `https://jetbrains.ru/products/idea/`

1. Обзор литературы

1.1. Существующие решения

На протяжении последнего десятилетия было создано множество рекомендательных систем, в той или иной степени повышающих продуктивность разработчиков. В следующих разделах будет рассмотрена функциональность нескольких наиболее свежих и интересных решений в данной области, а также алгоритмы, которые лежат в основе их работы.

1.1.1. Selene

Начать обзор хотелось бы с представленного в 2011 году инструмента Selene [7]. Он интегрируется в среду разработки Eclipse и позволяет получать релевантные фрагменты кода в реальном времени: при появлении изменений в тексте программы Selene анализирует их и в отдельном окне отображает несколько похожих фрагментов. Процесс не требует участия пользователя, хотя доступна также возможность ручного запуска. В основе алгоритма рекомендации лежит текстовый поиск, поэтому Selene получился достаточно эффективным и независимым от языка программирования.

Процесс работы инструмента разделён на два этапа. Сначала код пользователя из редактора отправляется на сервер, где происходит поиск наиболее похожих фрагментов из репозитория, содержащего 2 миллиона программ с открытым исходным кодом. Хранящийся на сервере код был предварительно векторизован с помощью метода TF-IDF. На втором этапе для найденных файлов построчно считается локальная оценка схожести. Она показывает, насколько рассматриваемая строка файла похожа на код рядом с позицией курсора в текущем открытом документе. Фрагменты с наибольшей оценкой отображались в окне инструмента.

К основным недостаткам инструмента относится отсутствие открытой реализации, из-за чего невозможно использовать его на практике.

Если же говорить о подходе, то следует отметить необходимость отправки кода на удалённый сервер, что является недопустимым для многих пользователей. Также для работы алгоритма требуется сбор и обработка большого объёма данных, вследствие чего усложняется добавление поддержки нового языка.

1.1.2. Seahawk

С распространением Stack Overflow многие исследователи обратили внимание на этот ресурс как на источник знаний. Одним из таких решений является Seahawk [9] – расширение для Eclipse, позволяющее находить подходящие обсуждения, не покидая среды разработки.

Поиск в данном инструменте осуществляется на основе запросов, которые могут быть введены самим пользователем или же сформированы автоматически. Для построения запроса Seahawk извлекает ключевые слова из целевой сущности – того метода или класса, где в текущий момент находится курсор. Учитываются имена переменных, аргументы методов, название самой сущности, а также используемые классы из сторонних библиотек. После этого полученный запрос отправляется на сервер, где происходит поиск по базе предварительно проиндексированных обсуждений из Stack Overflow. Индексация производилась с помощью меры TF-IDF.

Seahawk содержит практически все недостатки предыдущего решения, за исключением работы с кодом пользователя: здесь он обрабатывается локально. Кроме того, из-за множества различных окон инструмент получился достаточно громоздким.

1.1.3. Prompter

В 2014 году авторы расширения Seahawk представили инструмент Prompter [5], также реализованный для Eclipse. В отличие от предшественника, основной упор был сделан не на поиск по текстовому запросу, а на самостоятельный анализ кода в фоновом режиме с целью предоставления подсказок.

Prompter обрабатывает любые изменения в коде и генерирует запрос, включающий ключевые слова из кода пользователя. Чтобы подобрать наиболее полезные ключевые слова для запроса, подсчитывается специальная метрика – энтропия. Её значение будет тем выше, чем реже слово встречается в других документах. Затем сформированный запрос направляется в различные поисковые движки, а найденные ответы ранжируются на основе схожести с кодом пользователя, популярности ответа на сайте, репутации автора и некоторых других параметров [10]. После этого производится оценка релевантности результатов, и в случае, если она превысит заданный уровень, расширение отобразит уведомление для пользователя.

В данном инструменте авторы учли проблемы своей предыдущей работы. Тем не менее, опробовать Prompter не удалось, поскольку на текущий момент он неработоспособен.

1.1.4. Bing Developer Assistant

Следующим шагом стал инструмент Bing Developer Assistant (BDA) [6], представленный группой исследователей из Microsoft в 2016 году. Он работает на базе Visual Studio и предлагает разработчикам широкие возможности.

Во-первых, инструмент предоставляет фрагменты кода по текстовому запросу пользователя. Запрос отправляется в поисковый движок Bing, который находит HTML страницы, содержащие код. Сниппеты извлекаются, ранжируются по релевантности на основе оценок других пользователей и отображаются в окне BDA. Во-вторых, он, будучи интегрированным в систему автодополнения IntelliSense, умеет приводить примеры использования методов. Для реализации такой функциональности авторы собрали более 437 Гб данных с MSDN[4] и GitHub, из которых извлекли данные о вызовах API. В-третьих, ассистент предлагает помощь с возникающими ошибками компиляции. Он собирает различную контекстную информацию об ошибке и ищет решения в интернете.

[4] `https://msdn.microsoft.com/`

Наиболее заметным недостатком реализации является отсутствие подсветки синтаксиса найденных фрагментов кода, что значительно снижает их читаемость. Кроме того, функция помощи с ошибками предоставляет результаты в браузере, заставляя пользователя покинуть IDE.

1.1.5. NLP2Code

Ещё одним инструментом в данной области является расширение NLP2Code [8]. В плане использования инструмент крайне прост: запрос можно вводить непосредственно в редакторе, что не оставляет места для какой-либо избыточной информации. Полученный запрос отправляется в Google Custom Search Engine, который находит подходящие обсуждения на Stack Overflow. Из них извлекаются фрагменты кода, первый из которых автоматически добавляется в редактор.

Недостатком решения является время работы. Поскольку инструмент интегрирован в редактор, то запросы должны исполнятся в реальном времени, в противном случае разработчику придётся прерывать работу, ожидая выполнения запроса. На текущий момент скорость работы NLP2Code оставляет желать лучшего.

1.1.6. Иные решения

Отдельно стоит рассказать об успешных практических решениях, не имеющих публикаций. Одним из них является интеллектуальный помощник Codota[5]. Он интегрируется в среду разработки и позволяет получать фрагменты кода по текстовому запросу. Инструмент интересен тем, что для рекомендации использует сразу несколько алгоритмов исследования данных: прогностический анализ программ, методы обработки естественного языка и машинного обучения. Результаты работы моделей объединяются, чтобы предоставить полезный код для конкретной задачи. Обучение происходит на основе данных из открытых источников, код пользователя не отправляется на серверы компании. Codota доступна для разных IDE, в том числе для IntelliJ IDEA.

[5] `https://www.codota.com/`

Другим похожим инструментом является Kite[6], который также позиционируется как интеллектуальный помощник. Он поддерживает ряд популярных текстовых редакторов и IDE для Python и предоставляет широкую функциональность. Kite интегрируется в систему автодополнения, что позволяет ему ранжировать предлагаемые методы по релевантности. Для каждого метода он отображает несколько наиболее часто встречающихся вызовов, а при необходимости может открыть документацию на конкретный метод. Также он умеет искать сниппеты по запросу и исправлять небольшие ошибки в коде пользователя.

Оба инструмента являются полноценными программными продуктами с закрытым исходным кодом, поэтому информации об их реализации очень мало. Известно, что в них используются методы машинного обучения, откуда следует основной недостаток описанных решений – сложность добавления поддержки нового языка. Для этого требуется собрать набор данных с кодом на новом языке и обучить на нём модели.

1.1.7. Выводы

Таким образом, в рассмотренных решениях представлено множество интересных идей, однако среди них нет идеальных. Результаты сравнения приведены в таблице 1.

Первый столбец таблицы содержит названия решений, рассмотренных в данной работе. Во втором столбце указаны платформы, поддерживаемые каждым инструментом. Третий столбец содержит информацию о целевых языках программирования; цветом показана сложность добавления поддержки нового языка. Зелёный цвет обозначает относительную простоту добавления, оранжевый – необходимость сбора и обработки данных для нового языка, а красный показывает, что требуется собрать данные и обучить на них модель. Три следующих столбца отражают информацию об основной функциональности: поиск по текстовому запросу, предоставление подсказок в автоматическом режиме и помощь с ошибками компиляции.

[6] `https://kite.com/`

Название	Платформы	Языки	Поиск по запросу	Автомат. подсказки	Помощь с ошибками
Selene	Eclipse	Java	—	+	—
Seahawk	Eclipse	Java	+	—	—
Prompter	Eclipse	Java	+	+	—
Bing Developer Assistant	Visual Studio	C C++ C#	+	—	+
NLP2Code	Eclipse	Java	+	—	—
Codota	Eclipse IntelliJ IDEA	Java	+	—	—
Kite	Atom Sublime Text PyCharm	Python	+	—	+

Таблица 1: Сравнение существующих решений.

Что касается алгоритмов, лежащих в основе работы инструментов из обзора, то они в большинстве своём схожи и состоят из двух частей: поиска и ранжирования данных.

Сначала по текстовому запросу выполняется поиск подходящей информации с помощью собственных индексаторов или сторонних сервисов. При этом оба подхода имеют свои преимущества и недостатки. Использование собственных решений для поиска позволяет сделать его более гибким, но требует сбора данных. Сторонние сервисы потенциально обеспечивают лучшую точность рекомендации за счёт большого объёма доступных данных, но налагают ограничения на использование.

Чтобы сделать результаты более релевантными, производится их ранжирование. Здесь можно выделить несколько идей, наиболее интересной из которых является использование кода пользователя в качестве контекста рекомендации. Для этого в рассмотренных решениях применяются векторные модели и техники обработки естественного языка. Разумно также производить ранжирование на основе доступной метаинформации, например, оценок других пользователей.

В данной работе было решено воспользоваться наиболее удачными идеями существующих решений, а затем попытаться улучшить их.

1.2. Метрики для оценки качества ранжирования

Ранжирование – это процесс сортировки набора элементов по релевантности относительно некоторого критерия. Применительно к данной работе критерием является запрос, элементами – документы, содержащие полезную информацию, а релевантность характеризует соответствие документа запросу.

В процессе работы алгоритма ранжирования каждому элементу из множества документов E присваивается вес, характеризующий степень его релевантности данному запросу (чем больше вес, тем выше релевантность). Набор весов можно упорядочить по убыванию, получив таким образом перестановку π.

Чтобы из множества алгоритмов ранжирования выбрать наилучший, необходимо уметь оценивать их качество. Для этого нужна эталонная функция $r_{true} : E \to [0, 1]$, которая характеризует истинную релевантность элемента. Если документ полностью соответствует запросу, функция примет значение 1, а если совсем не подходит, то 0. Для её получения используют два основных подхода. Первый предполагает сбор исторических данных (например, информации о действиях пользователя), в соответствии с которыми присваиваются веса элементам. В основе второго лежит экспертная оценка: привлекается команда экспертов, которая вручную оценивает релевантность элементов.

Метрики для оценки качества ранжирования показывают, насколько веса, присвоенные алгоритмом ранжирования, соответствуют значениям эталонной функции.

Для данной работы метрики должны обладать следующими свойствами. Во-первых, они должны оперировать небинарными значениями. Во-вторых, должен учитываться порядок следования элементов. И в-третьих, результат должен быть нормирован. Среди множества метрик были выбраны две, удовлетворяющие описанным требованиям.

1.2.1. Normalized discounted cumulative gain

Выберем один запрос и K наиболее релевантных ему элементов. Дисконтированная сумма (Discounted Cumulative Gain, DCG)[7] – распространённая метрика ранжирования, которая учитывает порядок элементов в списке путём умножения их релевантности на функцию дисконтирования.

$$DCG_K = \sum_{k=1}^{K} \frac{2^{rel(k)} - 1}{\log_2(k+1)}, \tag{1}$$

где $rel(k) = r_{true}(\pi^{-1}(k))$ – релевантность элемента, который после сортировки списка по убыванию оказался на k-ой позиции.

DCG_K может работать с небинарными входными данными, однако её значения не нормализованы. На практике обычно применяют модификацию DCG_K – $nDCG_K$.

$$nDCG_K = \frac{DCG_K}{IDCG_K}, \tag{2}$$

где $IDCG_K$ – значение DCG_K в идеальной ситуации, когда первые K элементов получили высшие оценки.

Описанная выше метрика оценивает точность ранжирования только для отдельного запроса. Если необходимо учесть несколько запросов, вычисляется среднее значение.

1.2.2. Expected reciprocal rank

Предыдущая метрика не учитывала, как пользователь изучает предложенные документы: предполагалось, что элементы просматриваются независимо друг от друга. Однако на практике просмотр часто выполняется последовательно, а вероятность, что пользователь увидит следующий элемент, зависит от релевантности предшествующих.

Примером метрики, использующей подобную идею, может служить ожидаемый обратный ранг (Expected Reciprocal Rank, ERR) [11].

[7] https://en.wikipedia.org/wiki/Discounted_cumulative_gain

$$ERR_K = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{k}R(k)\prod_{i=1}^{k-1}(1-R(i)), \tag{3}$$

где $R(k) = (2^{rel(k)} - 1)/2^{rel_{max}}$.

Данная метрика показывает, с какой вероятностью среди первых K элементов в упорядоченном наборе найдётся тот, который окажется полезным для пользователя (т.е. пользователь остановится на нём).

2. Алгоритм рекомендации

2.1. Форма предоставляемых данных

Полезная для пользователя информация может представляться в различных формах: фрагменты кода, документация, статьи. Прежде всего необходимо выбрать из них наиболее подходящую. С точки зрения полезности одним из самых важных аспектов рекомендуемой информации является наглядность: необходим пример решения той или иной задачи другими людьми [2]. Исходя из этого, основной формой представления данных выбраны фрагменты кода.

Однако не стоит забывать о ресурсах для обмена знаниями, где пользователи могут задавать вопросы, связанные с программированием. Польза таких обсуждений не ограничивается только примерами кода для решения поставленной проблемы; было установлено, что пояснения и комментарии так же важны, поскольку они облегчают понимание кода [12]. Вот почему в данной работе решено также отображать информацию из обсуждений с сайта Stack Overflow – самого популярного подобного ресурса.

Перейдём к описанию алгоритма рекомендации, общая схема которого представлена на рис. 1.

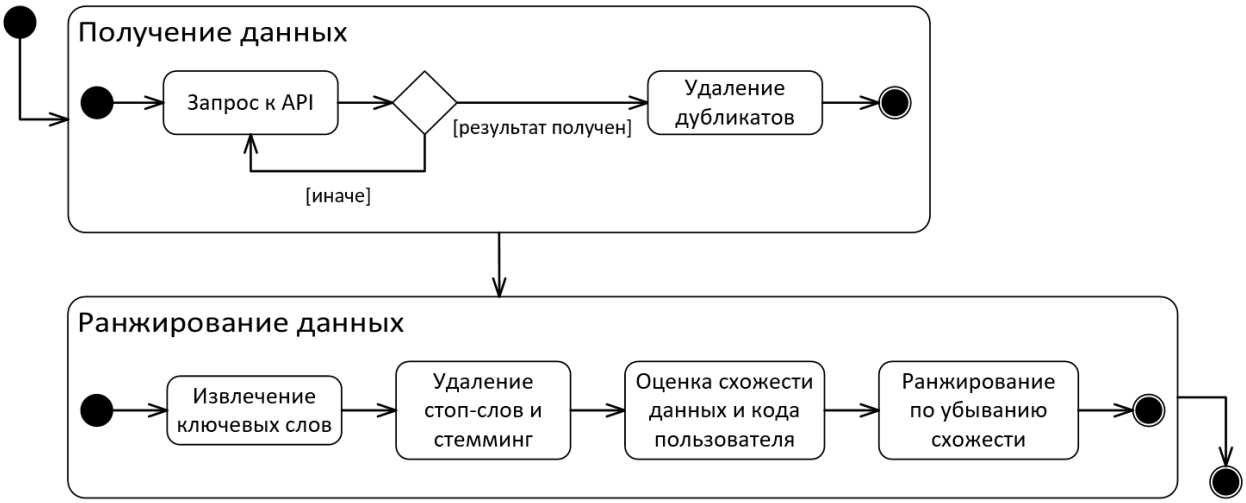


Рис. 1: Диаграмма активности обобщённого алгоритма рекомендации.

2.2. Получение данных

Первым шагом алгоритма является получение нужной информации. В отличие от большинства аналогов, рассмотренных в обзоре, было решено не создавать собственную систему для индексации данных, а воспользоваться уже существующими. Это избавит от необходимости сбора, хранения и обработки большого объёма данных. На сегодняшний день подобные системы просты в использовании и обеспечивают высокую степень релевантности данных. Примерами могут служить API популярных поисковых систем: Google Custom Search[8] и Bing Web Search[9]. К сожалению, они имеют различные ограничения на использование: фиксированное количество запросов в день, необходимость получения ключа доступа, ограничение области поиска и другие. По этим причинам пришлось отказаться от их использования.

В конечном итоге были выбраны два сервиса, в наибольшей степени удовлетворяющие потребностям данной работы. В качестве источника фрагментов кода используется сервис SearchCode[10], который позволяет осуществлять поиск среди более чем 7 миллионов проектов с 10 ресурсов. Для поиска дискуссий используется API Stack Exchange[11]. Оба сервиса бесплатны, не требуют получения ключа доступа, а дневной лимит составляет 10 000 запросов на IP-адрес, что достаточно для нормального использования.

2.3. Удаление дубликатов

Одной из основных проблем, связанных с данными, является их дублирование. Эта проблема особенно актуальна для фрагментов кода: файлы, содержащие различия всего в нескольких символах, считаются различными. Для её решения выполняется фильтрация данных, проходящая в два этапа. Сначала средствами сервиса SearchCode для каждого найденного фрагмента производится поиск схожих с ним файлов.

[8] `https://developers.google.com/custom-search/json-api/v1/overview`

[9] `https://azure.microsoft.com/en-us/services/cognitive-services/bing-web-search-api`

[10] `https://searchcode.com/api/`

[11] `https://api.stackexchange.com/`

Благодаря нечёткому алгоритму поиска удаётся отсечь некоторые файлы с незначительными различиями. Дополнительная фильтрация производится также на стороне клиента. Реализованный алгоритм удаляет из файлов комментарии (именно в них содержится большая часть различий), а затем производит построчное сравнение. При достаточно высокой доле совпадений файлы считаются идентичными, и в финальный набор попадает только один из них.

2.4. Ранжирование полученных данных

Сторонние API обеспечивают лишь предварительный отбор подходящих данных. Чтобы сделать рекомендацию релевантной задачам пользователя, необходимо учитывать контекст. В этом отношении наиболее информативен текущий открытый файл с исходным кодом: предполагается, что именно в нём содержится проблема, решение которой необходимо пользователю в данный момент [9]. Стоит, однако, помнить, что обработка кода пользователя должна производиться полностью локально, поскольку его отправка на удалённый сервер для многих может быть недопустимой.

В следующих разделах будут рассмотрены детали алгоритмов ранжирования для каждой формы представления информации.

2.4.1. Ранжирование фрагментов кода

При создании алгоритма ранжирования фрагментов кода необходимо было соблюсти компромисс между точностью и универсальностью. С одной стороны, рекомендуемый код должен быть релевантным задачам пользователя, а с другой – алгоритм не должен зависеть от языка программирования, на котором пишет пользователь. Как показано в исследовании [13], объединение техник обработки естественного языка и простого анализа кода способно эффективно определить степень схожести между фрагментами кода. При этом данный подход может быть использован для разных языков программирования, так как не полагается на синтаксис конкретного языка.

Оценка релевантности найденных фрагментов кода в данной работе производится с помощью векторной модели [14, глава 6, раздел 3]. Построение векторов осуществляется на основе ключевых слов, полученных из текущего открытого документа пользователя.

Для этого средствами Program Structure Interface (PSI) [15] из пользовательского кода извлекаются все идентификаторы: имена переменных, методов и классов, встречающиеся в коде. Они разбиваются на отдельные слова по регистру и знакам препинания, а затем полученный список ключевых слов проходит обработку, для которой используется библиотека Apache Lucene[12]. Производится удаление стоп-слов, которые не несут смысловой нагрузки, что позволяет уменьшить размерность будущих векторов. Из оставшихся слов выделяются основы с помощью алгоритма Snowball[13]. Это необходимо, чтобы учесть слова, схожие по смыслу, но различные по написанию.

Вес ключевого слова для данного фрагмента вычисляется с помощью меры TF-IDF [14, глава 6, раздел 2].

$$w_t = tf(t,d) \cdot idf(t,D) = (1 + \log(f_{t,d} + 1)) \cdot \log \frac{|D|}{|\{d \in D : t \in d\}|}, \qquad (4)$$

где t – ключевое слово, $d \in D$ – фрагмент кода, $f_{t,d}$ – количество вхождений ключевого слова t во фрагмент d.

Вычислив данную меру для всех ключевых слов, мы получаем вектор, представляющий фрагмент кода в векторном пространстве модели. Степень схожести двух векторов определяется косинусом угла между ними.

$$similarity(A,B) = \frac{A \cdot B}{||A|| \cdot ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2}\sqrt{\sum_{i=1}^{n} (B_i)^2}}, \qquad (5)$$

Последним шагом является сортировка найденных фрагментов кода по убыванию схожести с открытым документом пользователя.

[12] `http://lucene.apache.org/`

[13] `http://snowballstem.org/`

2.4.2. Ранжирование обсуждений

Stack Overflow – это интернет-портал, где разработчики могут публиковать свои вопросы и отвечать на уже опубликованные. Помимо текстового описания каждый вопрос содержит несколько тегов, которые облегчают понимание проблемы. Участники сообщества могут оценивать вопросы и ответы, тем самым указывая степень их полезности. Оценки бывают положительными и отрицательными, а их сумма составляет общую оценку вопроса/ответа. Автор вопроса также может отметить лучший ответ. Кроме того, каждый пользователь имеет репутацию – числовое значение, определяемое его активностью на сайте. Чем выше репутация, тем больше степень доверия к пользователю.

Таким образом, обсуждение содержит множество метаинформации, доступной для анализа. При этом каждый параметр может оказывать разное влияние на точность рекомендации, поэтому необходимо установить для них подходящие веса. Была использована конфигурация весов, эмпирически подобранная в ходе исследования [10]. Наиболее важными параметрами являются схожесть тегов между вопросом и кодом, репутация автора вопроса, а также общая оценка вопроса. Итоговая оценка дискуссии равна взвешенной сумме параметров.

Оценка схожести тегов производится на основе информации об импортированных классах в текущем открытом документе. Из собранных данных выделяются ключевые слова так же, как было описано в предыдущей главе, а затем для каждой дискуссии вычисляется доля совпадения тегов с ключевыми словами.

Оставшиеся параметры являются числовыми. Для их нормализации используется следующая логистическая функция.

$$f(x) = \frac{1}{1 + e^{x_{ave} - x}}, \tag{6}$$

где x_{ave} – среднее значение параметра x.

3. Архитектура инструмента

В данной работе была разработана и реализована архитектура подключаемого модуля к IntelliJ IDEA по схеме Presentation Model [16], которая позволяет отделить бизнес-логику от пользовательского интерфейса. Архитектура инструмента представлена на рис. 2.

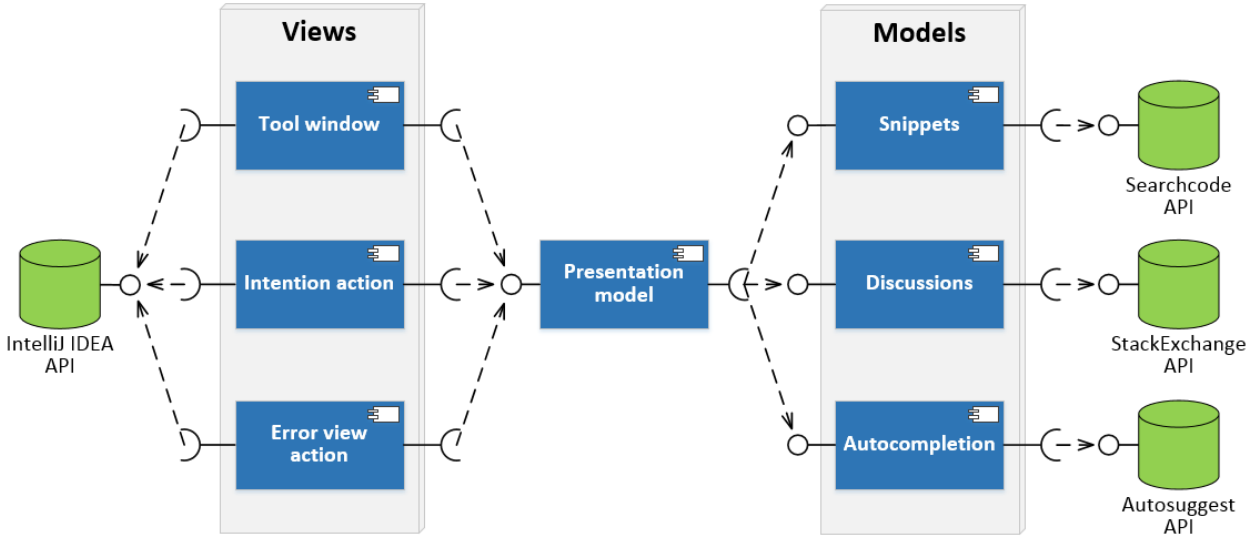


Рис. 2: Диаграмма компонентов инструмента.

Код системы разделён на три основных компонента: модель (model), модель представления (presentation model) и представление (view). Модель содержит данные и функции, отвечающие за их получение и обработку. Представление – это пользовательский интерфейс системы; оно включает формы с элементами управления. Модель представления является связующим звеном между данными и пользовательским интерфейсом. Этот компонент содержит логику и текущее состояние пользовательского интерфейса, выполняет дополнительную обработку данных модели, а также отвечает за обновление её состояния.

Каждый компонент также разделён на несколько независимых модулей, реализующих определённую часть функциональности. Благодаря модульности обеспечивается дополнительный уровень инкапсуляции, что упрощает отладку и тестирование.

4. Пользовательский интерфейс

Инструмент написан на языках Kotlin и Java. Пользовательский интерфейс реализован с использованием библиотеки Swing, расширенной собственными классами.

4.1. Функциональность

Основным способом взаимодействия пользователя с расширением является диалоговое окно, показанное на рис. 3. Оно располагается справа от редактора кода, позволяет ввести запрос и отобразить результаты поиска. Для удобства реализовано автодополнение, предлагающее несколько вариантов продолжения введённой фразы.

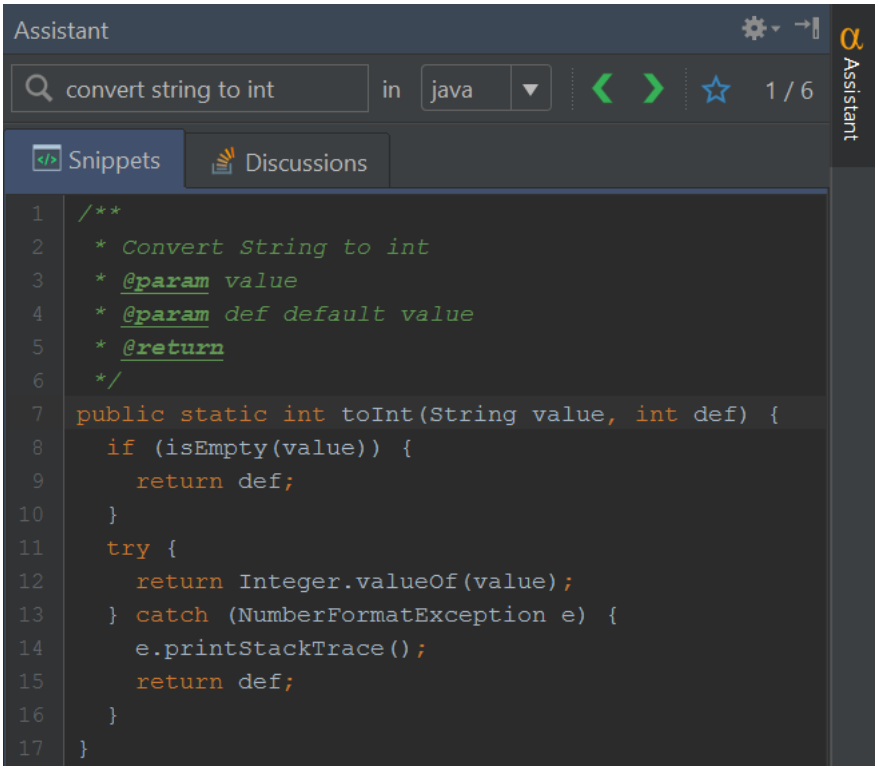


Рис. 3: Главное окно инструмента.

Вкладка “Snippets” содержит найденные фрагменты кода. На вкладке “Discussions” отображаются дискуссии с помощью мобильной версии сайта Stack Overflow.

Также расширение позволяет получить помощь с ошибками компиляции. Пользователь может вызвать подсказку непосредственно из редактора, либо из списка ошибок, как показано на рис. 4. При этом автоматически сформируется запрос, запустится поиск и отобразится окно с результатами.

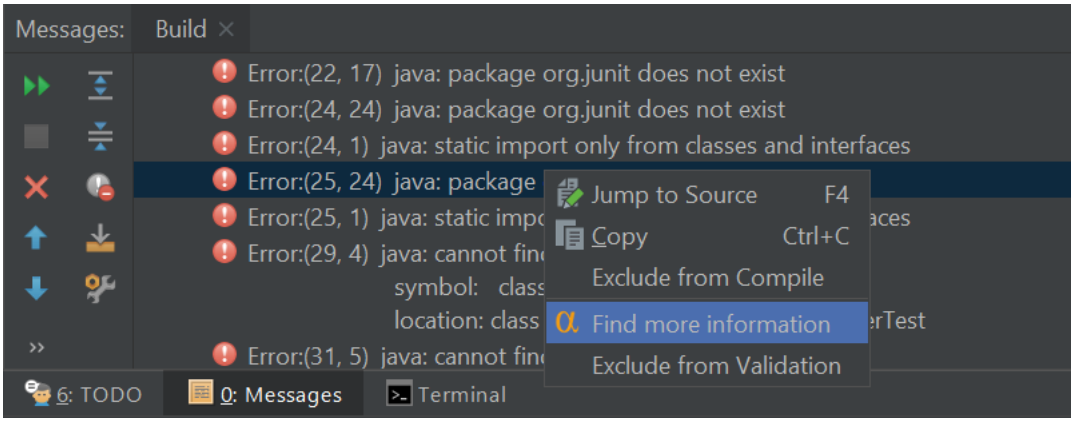


Рис. 4: Вызов подсказки из контекстного меню в списке ошибок.

5. Апробация

В ходе апробации было проведено сравнение разработанного решения с аналогами. Для этого несколько привлечённых экспертов оценили точность рекомендации инструментов из обзора, имеющих открытые реализации, а также разработанного решения. Оценивание производилось слепым методом с использованием сервиса Labelbox[14].

Поскольку большинство решений разработаны для Java, этот язык был выбран целевым. Из списка наиболее часто задаваемых вопросов в Stack Overflow были сформированы 10 запросов, которые затем подавались на вход инструментам. Первые 5 фрагментов кода из поисковой выдачи каждого решения добавлялись в тестовый набор данных. В качестве контекста использовался код авторов вопросов, схожих по формулировке с запросом.

5.1. Анализ результатов

Полученные результаты представлены в таблице 2.

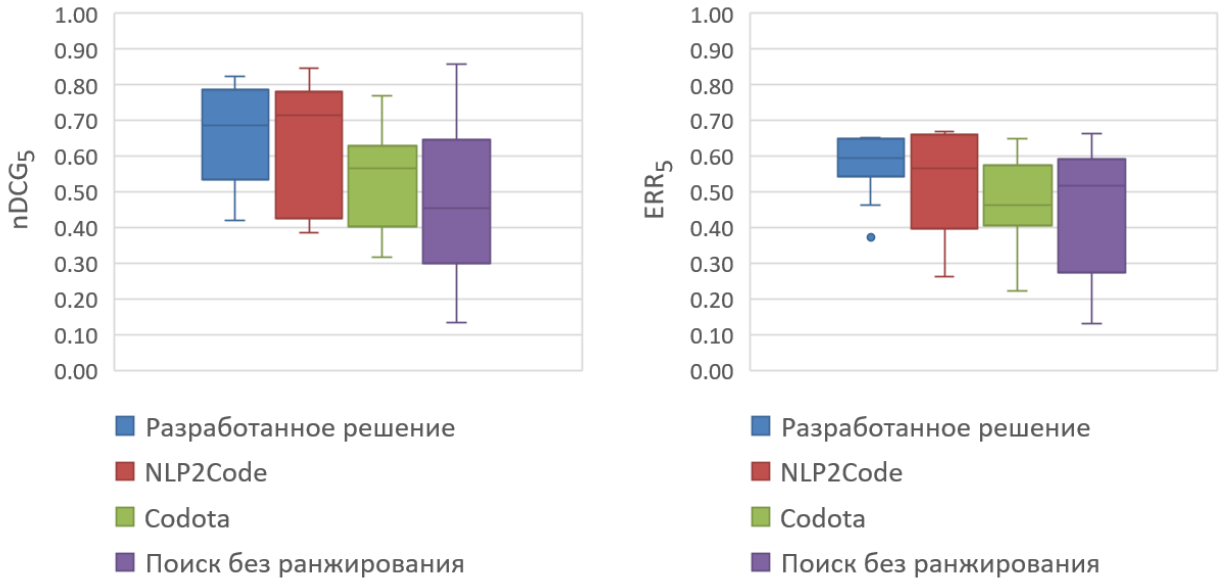
	$nDCG_5$	ERR_5
Разработанное решение	0.6572 ± 0.068	0.6003 ± 0.032
NLP2Code	0.6466 ± 0.088	0.5274 ± 0.069
Codota	0.5351 ± 0.070	0.4718 ± 0.062
Поиск без ранжирования	0.4689 ± 0.116	0.4564 ± 0.091

Таблица 2: Результаты сравнения точности рекомендации.

Значения метрик указаны с доверительным интервалом для уровня доверия 95%. Разработанный в рамках данной работы инструмент показывает лучшие результаты для обеих метрик, что говорит об эффективности применённого подхода.

[14] `https://labelbox.io/`

На рис. 5 отражено распределение значений метрик для инструментов, участвовавших в сравнении. Как показано на графиках, исходные данные являются достаточно однородными, а значит, полученные результаты можно считать надёжными.



(a) Для метрики nDCG.

(b) Для метрики ERR.

Рис. 5: Распределение значений метрик.

5.2. Границы применимости

В ходе апробации было выявлено, что все представленные решения эффективно работают только для достаточно простых запросов, таких как конвертация строки в число или сортировка массива. Также следует заметить, что данные из открытых источников зачастую отличаются низким качеством. Эти факторы являются общими для всех рассмотренных инструментов и ограничивают их применимость в коммерческой разработке.

Заключение

В ходе данной работы были получены следующие результаты.

- Проведён анализ предметной области. Сделан обзор функциональности аналогичных решений, используемых в них алгоритмов, а также метрик для оценки качества их работы.
- Реализован алгоритм рекомендации полезной информации. Он основан на техниках обработки естественного языка и простом анализе кода, благодаря чему может быть использован для разных языков программирования.
- Разработана архитектура инструмента, в основе которой лежит шаблон проектирования Presentation Model.
- Реализован пользовательский интерфейс. Инструмент позволяет находить релевантные данные по текстовому запросу, а также предлагает помощь с ошибками компиляции.
- Выполнена апробация разработанного решения. В ходе проведённого эксперимента алгоритм показал высокую точность рекомендации.

Таким образом, в данной работе представлен инструмент для помощи с распространёнными проблемами, возникающими в ходе разработки программного обеспечения. Ассистент будет полезен в первую очередь студентам, а также начинающим разработчикам, которые работают над проектами с открытым исходным кодом. Код инструмента доступен по ссылке[15].

[15] `https://github.com/MaKToff/Assistant`

Список литературы

[1] Monsell S. Task switching // Trends in Cognitive Sciences 7 (3). 2003. P. 134 – 140.

[2] Proksch S., Bauer V., Murphy G. C. How to build a recommendation system for software engineering // Springer International Publishing. 2015.

[3] Kersten M., Murphy G. C. Using task context to improve programmer productivity // FSE 2014: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering. ACM Press, 2006. P. 1 – 11.

[4] Neural Sketch Learning for Conditional Program Generation / V. Murali, L. Qi, S. Chaudhuri et al. // arXiv. 2017.

[5] Prompter: A Self-Confident Recommender System / L. Ponzanelli, G. Bavota, M. D. Penta et al. IEEE, 2014. P. 577 – 580.

[6] Bing developer assistant: improving developer productivity by recommending sample code / H. Zhang, A. Jain, G. Khandelwal et al. // FSE 2016: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2016. P. 956 – 961.

[7] Takuya W., Masuhara H. A spontaneous code recommendation tool based on associative search // SUITE '11: Proceedings of the 3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation. ACM, 2011. P. 17 – 20.

[8] Campbell B. A., Treude C. NLP2Code: code snippet content assist via natural language tasks // arXiv. 2017.

[9] Ponzanelli L., Bacchelli A., Lanza M. Seahawk: stack overflow in the IDE // ICSE '13: Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013. P. 1295 – 1298.

[10] Mining StackOverflow to turn the IDE into a self-confident programming prompter / L. Ponzanelli, G. Bavota, M. D. Penta et al. // MSR 2014: Proceedings of the 11th Working Conference on Mining Software Repositories. ACM, 2014. P. 102 – 111.

[11] Expected reciprocal rank for graded relevance / O. Chapelle, D. Metlzer, Y. Zhang et al. // Proceedings of the 18th ACM conference on Information and knowledge management. ACM, 2009. P. 621 – 630.

[12] What makes a good code example? A study of programming Q&A in StackOverflow / S. M. Nasehi, J. Sillito, F. Maurer et al. IEEE, 2012. P. 25 – 34.

[13] Zilberstein M., Yahav E. Leveraging a corpus of natural language descriptions for program similarity // Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software. ACM, 2016. P. 197 – 211.

[14] Manning C. D., Raghavan P., Schütze H. Introduction to Information Retrieval. Cambridge University Press, 2008.

[15] IntelliJ Platform SDK DevGuide. PSI Elements. URL: `http://www.jetbrains.org/intellij/sdk/docs/basics/architectural_overview/psi_elements.html`.

[16] Martin Fowler. Presentation Model. URL: `https://martinfowler.com/eaaDev/PresentationModel.html`.