DOI: 10.15514/ISPRAS-2023-35(4)-10

CC BY

Автоматическое определение сходства Javadoc-комментариев

[1] *Д.В. Кознов,* ORCID: 0000-0003-2632-3193 <*d.koznov@spbu.ru*>
[2] *Е.Ю. Леденева,* ORCID: 0009-0002-8907-143X <*ekiuled@gmail.com*>
[1] *Д.В. Луцив,* ORCID: 0000-0002-6332-2360 <*d.lutsiv@gmail.com*>
[3] *П.И. Браславский,* ORCID: 0000-0002-6964-458X <*pbras@yandex.ru*>

[1] *Санкт-Петербургский государственный университет,*
Россия, 199034, г. Санкт-Петербург, Университетская наб., д. 7–9
[2] *ООО Яндекс.Технологии,*
Россия, 11902, Москва, 1 ул. Льва Толстого, д. 7
[3] *НИУ «Высшая школа экономики»,*
Россия, 109028, Москва, Покровский бульвар, д. 11

Аннотация. Комментарии в исходном коде являются важной частью документации программного обеспечения. Многие программные проекты страдают от некачественных комментариев, которые часто создаются путем копирования и содержат многочисленные ошибки и неточности. В случае схожих методов, классов и т.п. копирование комментариев с небольшими изменениями оправдано, но и в этом случае разработчики делают ошибки. В этом исследовании мы решаем проблему обнаружения похожих комментариев к исходному коду, что позволяет улучшить комментариев к коду. Применительно к задаче определения сходства JavaDoc-комментариев мы провели оценку традиционных алгоритмов сходства строк и современных методов машинного обучения. В нашем эксперименте мы используем коллекцию комментариев Javadoc из четырех промышленных Java-проектов с открытым исходным кодом. Мы выяснили, что LCS (Longest Common Subsequence) является лучшим алгоритмом для решения нашей задачи, учитывая как качество (точность 94%, полнота 74%), так и производительность.

Ключевые слова: документация программного обеспечения; комментарии Javadoc; метрики схожести.

Для цитирования: Кознов Д.В., Леденева Е.Ю., Луцив Д.В., Браславский П.И. Вычисление схожести комментариев Javadoc. Труды ИСП РАН, том. 35, выпуск 4, 2023. с. 177-186. DOI: 10.15514/ISPRAS-2023-35(4)-10.

Evaluation of Similarity of Javadoc Comments

[1] *D. V. Koznov,* ORCID: 0000-0003-2632-3193 <*d.koznov@spbu.ru*>
[2] *E. Iu. Ledeneva,* ORCID: 0009-0002-8907-143X <*ekiuled@gmail.com*>
[1] *D. V. Luciv,* ORCID: 0000-0002-6332-2360 <*d.lutsiv@gmail.com*>
[3] *P. I. Braslavski,* ORCID: 0000-0002-6964-458X <*pbras@yandex.ru*>

[1] *Saint-Petersburg State Uinversity,*
7/9 Universitetskaya emb., St. Petersburg, 199034, Russia
[2] *Yandex LLC, 7 Lva Tolstogo st., Moscow, 119021, Russia*
[3] *HSE University, 11 Pokrovsky blvd., Moscow, 109028, Russia*

Abstract. Code comments are an essential part of software documentation. Many software projects suffer the problem of low-quality comments that are often produced by copy-paste. In case of similar methods, classes, etc. copy-pasted comments with minor modifications are justified. However, in many cases this approach leads to degraded documentation quality and, subsequently, to problematic maintenance and development of the project. In this study, we address the problem of near-duplicate code comments detection, which can potentially improve software documentation. We have conducted a thorough evaluation of traditional string similarity metrics and modern machine learning methods. In our experiment, we use a collection of Javadoc comments from four industrial open-source Java projects. We have found out that LCS (Longest Common Subsequence) is the best similarity algorithm taking into account both quality (Precision 94%, Recall 74%) and performance.

Keywords: software documentation; Javadoc comments; similarity measure.

For citation: Koznov D.V., Ledeneva E.Iu., Luciv D.V., Braslavski P.I. Calculating similarity of Javadoc comments. Trudy ISP RAN/Proc. ISP RAS, vol. 35, issue 4, 2023. pp. 177-186. DOI: 10.15514/ISPRAS-2023-35(4)-10.

1. Введение

Комментарии в исходном коде – один из наиболее важных видов документации программного обеспечения. Комментарии играет важную роль при сопровождении и поддержке программного обеспечения, особенно при разработке программных интерфейсов (API), которые создаются для использования другими разработчиками [1].

Существуют специальные инструменты (например, Javadoc [1], Doxygen [2]), которые по комментариям в исходном коде, представленным в специальном формате (а также некоторым другим артефактам – диаграммам в формате GraphViz [3], сигнатурам методов и пр.) генерируют html/pdf документацию. Для Java-приложений стандартом де-факто является технология Javadoc, которая определяет язык разметки и программный инструмент, интегрированный в большинство сред разработки на Java, таких как Eclipse, IntelliJ IDEA и другие.

Обычно в комментариях встречается множество дубликатов, поскольку при комментировании новых классов, методов и прочих программных сущностей разработчики зачастую копируют и вставляют в новые места уже существующие комментарии [2]. Эта практика является общепринятой, поскольку и сами новые сущности также очень часто функционально похожими на существующие. Скопированный комментарий при этом исправляется и дополняется, чтобы отразить специфику новой сущности, в результате чего порождается *неточный дубликат*. Однако скопированные комментарии могут содержать опечатки и ошибки: разработчик может скопировать старый комментарий, слегка изменить его, окончательную доработку отложить на потом, и в конце концов просто забыть о ней.

[1] https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html
[2] https://www.doxygen.nl/
[3] https://graphviz.org/

Также возможны ошибки в изменённых фрагментах комментария, например, когда разработчик забывает заменить в скопированном комментарии часть идентификаторов.

Таким образом, наличие неточных дубликатов считается нормальной практикой, но из-за вышеупомянутых проблем требуются специализированные инструменты для их обнаружения и анализа, чтобы выявить несоответствия между комментариями и исходным кодом, к которому они относятся [2].

Изучению различных дубликатов в комментариях посвящён ряд работ [2]. Тем не менее, исследователи не уделяют должного внимания ни автоматическому обнаружению дубликатов, ни поиску неточных повторов. Из перечисленных работ, лишь [3] предлагает инструментарий для поиска дубликатов, но и там оставлены в стороне неточные дубликаты.

Данная работа восполняет этот пробел, рассматривая неточные дубликаты в Javadoc-комментариях. Мы исследуем различные алгоритмы, которые могут использоваться для оценки попарного сходства комментариев (pairwise similarity), рассматривая как алгоритмы сравнения строк, так и современные алгоритмы машинного обучения. В ходе экспериментов мы проанализировали четыре широко известных Java-проекта с открытым исходным кодом: JSON, JUnit4, Mockito и Slf4J. В результате мы выяснили, что алгоритм LCS (поиск наибольшей общей подпоследовательности) является наилучшим при определении сходства, как с точки зрения качества результата (точность – 94%, полнота – 74%), так и с точки зрения производительности. Мы определили все неточные дубликаты в документации рассмотренных проектов, а также обнаружили некоторые ошибки в комментариях.

2. Background

2.1 Неточные дубликаты в Javadoc-комментариях

Рассмотрим два следующих метода из проекта JUnit4: `assertTrue` и `assertFalse`. Первый активирует исключение (exception), если его аргумент имеет значение «истина», второй делает тоже, если аргумент имеет значение «ложь». Очевидно, что функциональность методов очень похожа, следовательно, их комментарии также должны быть похожи. В самом деле, разница этих комментариев заключается в одном единственном слове – см. пример ниже, в котором различающиеся слова выделены жирным шрифтом и подчеркнуты. Таким образом, эти комментарии являются *неточными дубликатами* (рис.1).

```
/** Asserts that a condition is true.
    If it isn't it throws an AssertionError
    without a message.
    @param condition - condition to be checked */
public static void assertTrue(boolean condition)

/** Asserts that a condition is false.
    If it isn't it throws an AssertionError
    without a message.
    @param condition - condition to be checked */
public static void assertFalse(boolean condition)
```

Рис. 1. Неточные дубликаты комментариев к методам проекта JUnit4
Fig. 1. Near duplicates of JUni4 project methods' comments

2.2 Выбранные для экспериментов алгоритмы

Для определения неточных дубликатов необходима функция сходства (similarity function). Рассмотрим наиболее известные алгоритмы для вычисления функции сходства двух строк:

поиск наибольшей подпоследовательности (Longest Common Subsequence, LCS), косинусное сходство (Cosine Similarity, COS), локально-чувствительное хеширование (Locality-Sensitive Hashing, LSH), а также известные алгоритмы машинного обучения – Word Mover's Distance (WMD), doc2vec (D2V), and Siamese Neural Network (SNN). Краткий обзор этих алгоритмов представлен в табл. 1.

LCS, COS, и LSH являются широко известными алгоритмами сравнения строк и основаны на разных идеях. Мы использовали следующие реализации этих алгоритмов: LCS из библиотеки difflib [4], COS из библиотеки scipy [5], реализацию LSH мы адаптировали из библиотеки datasketch[6].

WMD, D2V и SNN являются ведущими на сегодняшний день алгоритмами машинного обучения, предназначенными для обработки текста. Мы использовали реализацию WDM and D2V из библиотеки gensim[7], реализацию SNN из проекта Keras[8].

Табл. 1. Краткий обзор алгоритмов вычисления сходства строк
Table 1. Brief Overview of Algorithm Selected

Алгоритм	Описание
LCS [6]	Измеряет длину наибольшей общей подпоследовательности двух строк в виде идентичных фрагментов текста, следующих в одном и том же порядке.
COS [7]	Преобразует значения частот вхождения слов в текст (term frequency, TF) в многомерные вектора. Каждая компонента вектора пропорциональна частоте употребления этого слова в тексте и обратно пропорциональна частоте употребления слова в совокупности анализируемых текстов. Сходство этих векторов вычисляется как косинус угла между ними.
LSH [8]	Представляет текст как мультимножество n-грамм (наборов из нескольких последовательно встречающихся слов), к которым применяет алгоритм minhash [9], результирующие битовые вектора разбивает на фрагменты и вычисляет долю совпадающих фрагментов векторов, получившихся из разных комментариев.
WMD [10]	Представляет два документа как множества точек в векторном пространстве (эмбеддинги, embeddings) и вычисляет их сходство как цену перемещения одного множества в другое. Использует алгоритм word2vec [11] для конструирования эмбеддингов для слов документа, основываясь на нейронной сети, обученной на большом корпусе текстов.
D2V [12]	Расширение word2vec, в котором строятся эмбеддинги документов, а не отдельных слов.
SNN [13]	Нейронная сеть с двумя подсетями, имеющими общую архитектуру и веса. Наша реализация использует LSTM (Long Short-Term Memory) компоненты.

[4] https://docs.python.org/3/library/difflib.html
[5] https://scipy.org
[6] http://ekzhu.com/datasketch/
[7] https://radimrehurek.com/gensim/
[8] https://keras.io/

3. Обзор сходных работ

Очень часто комментарии к коду имеют невысокое качество – они могут быть неполными, содержать ошибки, опечатки и противоречия [14]. Более того, если система достаточно велика, комментарии к коду написаны не в едином стиле, поскольку создавались различными разработчиками [16]. Имеется некоторое количество методов и инструментов для определения ошибок в комментариях к коду [5], [17]. Но данная задача всё ещё далека от разрешения и требуются новые подходы.

Имеется ряд исследований, фокусирующихся на дубликатах в программной документации – обзоры могут быть найдены в [16], [22]. В работах [16], [22] рассматриваются неточные дубликаты, но эти работы не имеют дела с JavaDoc-комментариями. Nosál и Porubän [4] рассматривают неточные дубликаты в Javadoc. Однако, они только представляют Javadoc-плагин для реализации повторного использования в комментариях, опуская вопрос поиска таких дубликатов в уже существующих комментариях. Oumaziz и др. [2], а также and Blasi и др. [3] исследовали связь между дубликатами в Javadoc-комментариях и дубликатами в коде, но они не рассматривали неточные дубликаты. Wagner и Fernández [25] писали о алгоритмах выявления повторов документации ПО, но они не рассматривали Javadoc. А между тем, Javadoc является широко используемым на практике форматом, применяясь в большинстве индустриальных Java-проектах.

Таким образом, существующие исследования Javadoc-дубликатов не рассматривают применение современных алгоритмов сопоставления сток и текстов, а также не рассматривают неточные дубликаты.

В своих экспериментах мы использовали известные Java-проекты, представленные в табл. 2. Кроме объёма кода мы также рассматриваем число классов в проектах, а также количество методов, и комментариев. Наконец, мы рассматриваем число пар неточных дубликатов, найденных в результате наших экспериментов.

4. Постановка экспериментов

В работах [16], [22] повторяющиеся фрагменты рассматривались безотносительно к структуре документа, что приводило к нахождению большого количества бессмысленных дубликатов. В работе [2] предлагается средство поиска повторяющихся тэгов в Javadoc, но предложенных подход также приводит к нахождению значительного количества очень «мелкозернистых» дубликатов, и полученную информацию в итоге слишком сложно анализировать.

В данной работе мы решили искать дубликаты в виде целых комментариев, относящиеся к различным сущностям исходного кода – классам, методам, аннотациям и т.д. Это позволило нам работать с заведомо осмысленными фрагментами текста.

Мы привлекли к эксперименту двух Java-разработчиков. Чтобы обеспечить корректность эксперимента, они независимо работали с одними и теми же данными, затем они обсудили результаты своей работы и пришли к общему мнению там, где их решения расходились.

Набор данных, размеченный экспертами, был создан следующим образом. Вначале мы объединили весь исходный код каждого проекта в один текстовый файл и оставили в этом файле лишь комментарии Javadoc и объявления соответствующих им программных сущностей. Затем мы запустили на полученных файлах Clone Miner (инструмент для поиска программных клонов на уровне лексем исходного кода) [4], получили набор групп клонов длинной не менее 5 слов в комментариях, подобно тому, как мы ранее делали это в [22].

Если некоторая группа G содержала два соседних комментария (а такие случаи встречались, поскольку Clone Miner не обращает внимания на структуру анализируемого текста), то мы разделяли каждый её клон на два и получали две группы. В нашем случае этого оказалось достаточно, поскольку более двух соседних комментариев в клоны одной группы не

попадало. Затем мы отсортировали группы по убыванию размера их клонов. Далее мы редактировали группы, заменяя каждую группу G на G' таким образом, что для всех клонов $g \in G$, текст которых являлся частью комментария g', в G' включался полный текст комментария g'. Таким образом, мы получили группы комментариев.

Далее, эксперты вручную анализировали каждую группу комментариев и выбирали пары комментариев, которые являлись неточными дубликатами. Для визуализации повторяющихся фрагментов комментариев одной группы мы использовали Duplicate Finder [24]. Полученный набор *положительных* пар был автоматически дополнен таким же количеством *отрицательных* пар комментариев из различных случайно выбранных групп. Отрицательные пары затем были дополнительно проверены экспертами. Помимо упомянутых инструментов в ходе эксперимента мы также использовали несколько разработанных вспомогательных скриптов на языке Python. В результате был получен размеченный экспертами набор данных, включающий 2600 пар комментариев (см. столбец «Количество пар» в табл. 2). Средняя длина Javadoc-комментария в рассматриваемых проектах равнялась 52 словам.

Табл. 2. Исследованные проекты
Table 2. Evaluated projects

Название проекта	Размер, Кб	Количество классов	Количество методов	Количество комментариев	Количество пар
GSON[9]	480	50	281	426	364
JUnit4[10]	588	103	433	602	744
Mockito[11]	901	110	471	669	786
Slf4J[12]	163	21	158	227	706

Для того, чтобы принять решение касательно того, является ли пара положительной или отрицательной, эксперты оценивали не только сходство текста комментариев, но и то, насколько близкую функциональность они описывали. Встречались ситуации, когда степень текстуального сходства была высока, но по смыслу комментарии были далеки. Такие пары учитывались как отрицательные. Если комментарии пары имели большой по объему сходный текст с одинаковым тегом (например, в них присутствовало одно и то же исключение), но при этом общая функциональность программных сущностей, к которым относились комментарии, значительно различалась, то такие пары учитывались как отрицательные. Наконец, некоторые короткие комментарии включали словосочетание или фразу, относящуюся к одному и тому же объекту (например, «tests to be run by the receiver»), но в целом комментарии описывали совершенно разную функциональность. Такие пары эксперты также считали отрицательными.

Нами было выполнено два эксперимента. Эксперимент 1 был проведён на размеченном экспертами наборе данных с разделением на обучающую и тестовую выборки в пропорции 70/30. Для каждого алгоритма мы нашли пороговое значение схожести, на котором достигалось максимальное значение меры F1. Вдобавок, для лучшей адаптации алгоритмов,

[9] https://github.com/google/gson
[10] https://github.com/junit-team/junit4
[11] https://github.com/mockito/mockito
[12] https://github.com/qos-ch/slf4j

основанных на машинном обучении (WMD, D2V, SNN), мы натренировали их эмбеддинги на большом наборе Javadoc-комментариев (приблизительно 2 миллиона), взятом с Kaggle[13].

Для того, чтобы получить более реалистичные результаты, мы провели эксперимент 2, в котором заместили тестовые данные *полным набором* Javadoc-комментариев из проекта JUnit4. При этом мы исключили пары JUunit, использованные в обучающей выборке. Поскольку в этом эксперименте тестовые данные не были полностью размечены экспертами, последние вручную проверили дополнительные положительные пары, найденные в ходе эксперимента 2. Для упрощения работы эксперты использовали инструмент Duplicate Finder [24], основанный на методе поиска точных повторов [26] и наглядно отображающий одинаковые фрагменты текста в отдельных элементах группы неточных дубликатов (в данном случае – в парах Javadoc-комментариев).

Поскольку наши исходные данные для обоих экспериментов не были сбалансированы в плане количества положительных и отрицательных пар, мы делали выводы о результатах наших экспериментов на основе трёх метрик – точности (P, Precision), полноты (R, Recall) и меры F1.

5. Результаты и дискуссия

Результаты наших экспериментов представлены в табл. 3. Столбцы 2-4 относятся к эксперименту 1, столбцы 5-8 – к эксперименту 2. В рамках эксперимента 1 мера F1 колеблется в пределах от 0,94 до 0,97. Строковые алгоритмы показывают постоянную точность (0,97), тогда как в случае алгоритмов на основе машинного обучения точность меняется от 0,92 (SNN) до 0,98 (WMD).

Табл. 3. Результаты экспериментов
Table 3. Experiment Results

Алгоритмы	Эксперимент 1			Эксперимент 2			
	P	R	F1	P	R	F1	Время, с.
LCS	0,97	0,95	0,96	0,96	0,74	0,84	15
COS	0,97	0,94	0,96	0,88	0,82	0,85	19
LSH	0,97	0,90	0,94	0,93	0,69	0,80	625
D2V	0,94	0,99	0,97	0,64	0,90	0,75	3157
WMD	0,98	0,95	0,96	0,97	0,73	0,84	1949
SNN	0,92	0,97	0,95	0,79	0,86	0,82	82

Полученные в ходе эксперимента 2 результаты ниже результатов, полученных в ходе эксперимента 1. Это объясняется тем, что в последнем случае был использован набор данных, вручную созданный экспертами. Как следствие, обучающие данные состояли из пар комментариев, хорошо разделённых на положительные и отрицательные, сходство комментариев в положительных парах близко к 1, сходство в отрицательных близко к 0. При постановке эксперимента 2 в нашем распоряжении было б*о*льшее количество пар, сходство в которых близк*о* к пороговому значению, отделяющему отрицательные и положительные пары. Их следовало бы использовать в качестве обучающих данных, но из-за большой

[13] https://www.kaggle.com/isofew/code-comment-pairs

трудоёмкости разметки новых данных этого не было сделано. Алгоритмы на основе машинного обучения, особенно D2V, оказались более чувствительны к данной специфике входных данных, нежели строковые алгоритмы.

Эксперимент 2 показывает, что неточных дубликатов среди комментариев очень много – 564 (неточные дубликаты) из 602 (общее количество дубликатов) в случае проекта JUnit4. Можно сделать вывод, большинство проектов в рассматриваемом проекте является неточными дубликатами, и лишь немногие из комментариев уникальны. Это свидетельствует о том, что при создании комментариев в реальных программных проектов копирование и вставка – нормальная практика, хотя для подтверждения этого вывода необходимо провести дополнительные эксперименты.

Время работы всех алгоритмов на данных проекта JUnit4 представлено в таблице 3 (столбец «время»). Хуже всех в этом смысле показали себя D2V и WMD (50 и 30 минут соответственно), при этом JUnit4 не является большим проектом. Таким образом, подобная производительность для практического применения неприемлема. D2V работал так медленно, поскольку по ходу работы он вычисляет свои весовые коэффициенты, в то время как остальные алгоритмы используют коэффициенты, вычисленные при тренировке. Использованный нами вариант алгоритма WMD тратил много времени для вычисления метрики Earth Mover's Distance, и в целом его сложность оценивается как $O(p^3 * \log p)$, где p – это количество уникальных слов в сравниваемых текстовых фрагментах. Время работы LCS, COS и SNN варьировалось от 15 секунд до полутора минут, что вполне приемлемо для использования на практике.

В нашем случае LSH не показал хороших результатов, поскольку Javadoc-комментарии короче топиков (topics) DITA, рассматриваемых в [23], поэтому хеш-сигнатуры работали недостаточно эффективно.

LCS, COS и WMD показали наилучшие результаты в обоих экспериментах. Но напомним, что WMD существенно проигрывает строковым алгоритмам в плане производительности. LCS же напротив, показывает наилучшую производительность, так что в итоге мы считаем его лучшим выбором для применения на практике. К такому же заключению пришли и авторы работы [23], хотя в своём исследовании они рассматривали другой набор алгоритмов.

Наконец, при анализе данных нам удалось найти несколько ошибок в комментариях, возникших вследствие копирования-вставки: в каких-то случаях различия комментариев в некоторых парах были излишними или некорректными, в других случаях различий не было там, где им следовало бы быть.

В качестве дальнейшего направления исследований мы предполагаем разработку подходов, инструментов и сервисов для поиска ошибок, вызванных копированием и вставкой. Также важно улучшить качество алгоритмов вычисления сходства комментариев, дополнив их специфичными для наших задач возможностями и обучая их на данных, лучше сбалансированных в плане количества положительных и отрицательных пар, и имеющих больший объём.

5. Заключение

В работе, применительно к вычислению схожести Javadoc-комментариев, были проанализированы различные алгоритмы вычисления схожести фрагментов текста. Мы установили, что существующие алгоритмы хорошо справляются с этой задачей. Проведённые эксперименты показали, что строковый алгоритм LCS (поиск наибольшей общей подпоследовательности) показал себя наилучшим образом. Также результаты говорят в пользу того, что интерактивные (с участием человека) методы, обладающие специфичными для решаемых задач возможностями, и классические строковые алгоритмы могут лучше показывать себя на практике по сравнению с методами, основанными на машинном обучении. Однако это требует дальнейшего исследования, поскольку основанные на

машинном обучении алгоритмы предположительно имеют значительный потенциал, но требуют и значительных усилий, чтобы его раскрыть. Также представляют интерес визуальные средства анализа экспериментов в духе [27-29], а также средства управления знаниями [30].

Литература

[1]. Spinellis D. Code Documentation // IEEE Softw. – 2010. – Vol. 27, no. 4. – pp. 18–19.
[2]. Oumaziz M. A. et al. Documentation Reuse: Hot or Not? An Empirical Study // Proc. of ICSR 2017. – 2017. – pp. 12–27.
[3]. Blasi A., Gorla A. Replicomment: identifying clones in code comments // Proc. of ICPC 2018, Gothenburg, Sweden. – ACM. – 2018. – pp. 320–323.
[4]. Nosál M., Porubän J. Reusable software documentation with phrase annotations // Central Eur. J. Comput. Sci. – 2014. – Vol. 4, no. 4. – pp. 242–258.
[5]. Corazza A. et al. On the Coherence between Comments and Implementations in Source Code // EUROMICRO-SEAA. – 2015. – pp. 76–83.
[6]. Chin F. Y. L., Poon C. K. Binary Codes Capable of Correcting Deletions, Insertions and Reversals // Afast algorithm for computing longest common subsequences of small alphabet size. – 1991. – Vol. 13(4). – pp. 463–469.
[7]. Manning C. D. et al. Introduction to information retrieval. – 2008. – Vol. 1.
[8]. Gionis A. et al. Similarity Search in High Dimensions via Hashing // Proc. of VLDB 1999, Edinburgh, Scotland, UK. – Morgan Kaufmann. – 1999. – pp. 518–529.
[9]. Broder A. Z. On the resemblance and containment of documents // Proc. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171). – IEEE, 1997. – c. 21-29.
[10]. Kusner M. J. et al. From Word Embeddings To Document Distances // Proc. of ICML 2015, Lille, France. – JMLR.org. – 2015. – Vol. 37. – pp. 957–966.
[11]. Mikolov T. et al. Efficient estimation of word representations in vector space //arXiv preprint arXiv:1301.3781. – 2013.
[12]. Le Q.V., Mikolov T. Distributed Representations of Sentences and Documents // ICML 2014. – 2014. – Vol. 32 of JMLR Workshop and Conference Proceedings. – pp. 1188–1196.
[13]. Mueller J., Thyagarajan A. Siamese Recurrent Architectures for Learning Sentence Similarity // Proc. AAAI, 2016. – AAAI Press. – 2016. – pp. 2786–2792.
[14]. Tan S. H. et al. @tComment: Testing Javadoc Comments to Detect Comment-Code Inconsistencies // ICST 2012. – IEEE Computer Society. – 2012. – pp. 260–269.
[15]. Fluri B., Würsch M., Gall H. C. Do Code and Comments Co-Evolve? On the Relation between Source Code and Comment Changes // Proc. of WCRE 2007, Vancouver, BC, Canada. – IEEE Computer Society. – 2007. – pp. 70–79.
[16]. Luciv D., Koznov D., Chernishev G., et al. Detecting Near Duplicates in Software Documentation // Programming and Computer Software. – 2018. – Vol. 44, no. 5. – pp. 335–343.
[17]. Wen F. et al. A large-scale empirical study on code-comment inconsistencies // Proc. of ICPC 2019 / ed. by Guéhéneuc Y. et al. – IEEE / ACM. – 2019. – pp. 53–64.
[18]. Wang D. et al. Deep Code-Comment Understanding and Assessment // IEEE Access. – 2019. – Vol. 7. – pp. 174200– 174209.
[19]. Zhou Y. et al. Analyzing APIs documentation and code to detect directive defects // Proc of the ICSE 2017, Buenos Aires, Argentina. – IEEE / ACM. – 2017. – pp. 27–37.
[20]. Ratol I. K., Robillard M. pp. Detecting fragile comments // Proc. of ASE 2017, Urbana, IL, USA. – IEEE Computer Society. – 2017. – pp. 112–122.
[21]. Otaibi J. A. et al. Machine Learning and Conceptual Reasoning for Inconsistency Detection // IEEE Access. – 2017. – Vol. 5. – pp. 338–346.
[22]. Koznov D. V. et al. Clone Detection in Reuse of Software Technical Documentation // 10th International Andrei Ershov Informatics Conference, PSI 2015. – Springer. – 2015. – Vol. 9609 of LNCS. – pp. 170–185.
[23]. Soto A. J. et al. Similarity-Based Support for Text Reuse in Technical Writing // Proc. of the ACM DocEng 2015, Lausanne, Switzerland / ed. by Vanoirbeek C., Genevès pp. – ACM. – 2015. – pp. 97–106.
[24]. Luciv D., Koznov D., Chernishev G., et al. Duplicate finder toolkit // Proc. of ICSE 2018: Companion Proceeedings. – 2018. – pp. 171–172.

[25]. Wagner S., Fernández D. M. Analyzing Text in Software Projects // The Art and Science of Analyzing Software Data / ed. by Bird Christian et al. – Morgan Kaufmann / Elsevier, 2015. – pp. 39–72.
[26]. Basit H. A. et al. Efficient token based clone detection with flexible tokenization // Proceedings of the ESEC/SIGSOFT FSE, 2007. – 2007. – pp. 513–516.
[27]. Кознов Д.В., Ольхович Л.Б.Визуальные языки проектов // Системное программирование. 2005. Т. 1. с. 148-167.
[28]. Кознов Д.В. Методология и инструментарий предметно-ориентированного моделирования // диссертация на соискание ученой степени доктора технических наук / Санкт-Петербургский государственный университет. Санкт-Петербург, 2016.
[29]. Гаврилова Т., Алсуфьев А., Янсон А.С. Современные нотации бизнес-моделей: визуальный тренд // Форсайт. 2014. Т. 8. № 2. с. 56-70.
[30]. Гаврилова Т.А. Логико-лингвистическое управление как введение в управление знаниями // Новости искусственного интеллекта. 2002. № 6. с. 36-40.

Информация об авторах / Information about authors

Дмитрий Владимирович КОЗНОВ - доктор технических наук, профессор кафедры системного программирования. Научные интересы: программная инженерия, модельно-ориентированная разработка программного обеспечения, программные данные, машинное обучение.

Dmitry Vladimirovich KOZNOV - Doctor of Technical Sciences, Professor of the System Programming Department. Research interests: software engineering, model-driven software development, program data, machine learning.

Дмитрий Вадимович ЛУЦИВ – кандидат физико-математических наук, доцент кафедры системного программирования Санкт-Петербургского государственного университета. Область научных интересов: программная инженерия, анализ данных программных проектов, анализ документации, системное программирование.

Dmitry Vadimovich LUCIV – PhD in computer science, associate professor of System Programming Department at Saint Petersburg State University, Russia. Research interests: software engineering, software data analysis, documentation analysis, systems programming.

Екатерина Юрьевна ЛЕДЕНЕВА – разработчик ООО Яндекс.Технологии, выпускница кафедры системного программирования Санкт-Петербургского государственного университета. Сфера научных интересов: анализ данных программных проектов, анализ технических текстов.

Ekaterina Iurevna LEDENEVA – Software engineer at Yandex LLC, Saint Petersburg State University alumni. Research interests: software data analysis, technical documentation analysis.

Павел Исаакович БРАСЛАВСКИЙ - кандидат технический наук, старший научный сотрудник научно-учебной лаборатории моделей и методов вычислительной прагматики НИУ «Высшая школа экономики». Научные интересы: ресурсы и методы для оценки моделей обработки естественного языка и информационного поиска, вычислительный юмор.

Pavel Isaakovich BRASLAVSKI - PhD in computer science, senior researcher at the Laboratory for Models and Methods of Computational Pragmatics, HSE University. Research interests: resources and methods for evaluation of NLP and IR models, computational humor.