

МЕТОД СЕМАНТИЧЕСКОГО АНАЛИЗА ИЗМЕНЕНИЙ ТРЕБОВАНИЙ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ В ВЕРСИЯХ ТЕХНИЧЕСКОЙ ДОКУМЕНТАЦИИ

А. С. Савинчева¹ [0009-0005-0756-6001], З. А. Еникеева² [0000-0002-2468-7455],

К. Ш. Еникеев³ [0009-0000-6020-9912]

¹⁻³Казанский (Приволжский) федеральный университет, г. Казань, Россия

¹asanvli@gmail.com, ²ZAEnikeeva@kpfu.ru, ³KSEnikееv@kpfu.ru

Аннотация

Представлен формализованный метод автоматического обнаружения изменений требований к программному обеспечению в процессе эволюции версий технической документации. Исходными данными служат множества требований, извлеченных и классифицированных на функциональные и нефункциональные, с дополнительной детализацией типов нефункциональных требований (производительность, безопасность, доступность и др.) для каждой версии документации. Задача сопоставления требований между версиями сведена к построению частичного отображения между множествами требований и присвоению каждому требованию статуса: неизмененное (Unchanged), измененное (Modified), новое (New) или удаленное (Deleted). Предложена композитная функция сходства пар требований, объединяющая семантическое сходство на основе векторных представлений (эмбеддингов), лексическое сходство, совпадение класса требования и структурный контекст документа. Представлены результаты применения метода на открытом наборе пар требований. Разработан прототип системы анализа изменений требований в технических документах.

Ключевые слова: требования к программному обеспечению, анализ изменений требований, сопоставление версий документации, прослеживаемость требований, семантическое сходство, метод взаимных ближайших соседей, метод Отсу.

ВВЕДЕНИЕ

Эволюция технической документации программных систем сопровождается итеративными правками: уточнением формулировок, добавлением новых ограничений, исключением устаревших требований, а также изменением числовых параметров. Для поддержки процессов управления конфигурацией, верификации и сопровождения необходимо не только зафиксировать факт изменения текста, но и интерпретировать его в терминах изменений требований к программному обеспечению (ПО) [1, 2]. Под требованием к ПО будем понимать формализуемое утверждение на естественном языке о свойствах программной системы, ее функциях, ограничениях или условиях эксплуатации, изменение которого влияет на стадии жизненного цикла системы (проектирование, тестирование, приемо-сдаточные испытания и др.).

В большинстве современных систем работы с документацией механизмы анализа изменений опираются на посимвольное или построчное сравнение текста [3, 4]. Такой подход фиксирует все правки, но не обеспечивает их семантическую интерпретацию. Исправление опечатки, перефразирование предложения и изменение критического числового параметра (например, времени отклика сервера) выглядят для подобного классического алгоритма как равнозначные правки. В результате специалист сталкивается с большим объемом изменений, где множество редакторских правок затрудняет процесс выделения именно тех, которые затрагивают содержание требований к ПО.

Альтернативным подходом является анализ изменений через призму уже выделенных требований. Если для каждой версии документации получены множества требований с присвоенными классами (функциональные, нефункциональные по категориям), то задача сводится к сопоставлению этих множеств и определению для каждого требования статуса по отношению к другой версии – неизмененное, измененное, новое или удаленное. Такой подход согласуется с практиками прослеживаемости требований (requirements traceability) и управления изменениями в инженерии требований [1, 5, 6].

В рамках отдельного исследования были разработаны и экспериментально оценены методы извлечения требований из технической документации и их последующей классификации [7]. Такое извлечение было реализовано на основе двунаправленной нейронной сети с долговременной краткосрочной памятью

(Bidirectional Long Short-Term Memory, BiLSTM) и слоя условных случайных полей (Conditional Random Field, CRF), что в совокупности обеспечивает учет контекстуальных зависимостей в последовательности предложений. Классификация требований устроена по таксономии из 13 классов (функциональные, 11 типов нефункциональных требований и второстепенный текст, не содержащий требований) на основе полносвязной нейронной сети. Для задач извлечения и последующей классификации достигнуты значения F1-score маско порядка 0.85. В настоящей работе указанные алгоритмы были использованы в готовом виде для получения множеств требований в различных версиях документации. Цель настоящего исследования состоит в проектировании и разработке метода автоматического обнаружения изменений требований к ПО между версиями технической документации, с присвоенным статусом изменения.

1. ФОРМАЛИЗАЦИЯ ЗАДАЧИ СОПОСТАВЛЕНИЯ ТРЕБОВАНИЙ МЕЖДУ ВЕРСИЯМИ

Пусть имеются две версии технической документации – предыдущая и новая, в общем случае не обязательно последовательные. В результате применения методов извлечения и классификации требований для каждой версии получено множество требований. Каждое требование задано текстовым фрагментом (формулировкой требования) и меткой из фиксированной таксономии классификации $C = \{\text{Functional, Availability, Legal, Look \& Feel, Maintainability, Operational, Performance, Scalability, Security, Usability, Fault tolerance, Portability, Other}\}$. Обозначим множество требований предыдущей версии как $R_{\text{old}} = \{r_1, r_2, \dots, r_n\}$, а новой версии – как $R_{\text{new}} = \{r'_1, r'_2, \dots, r'_m\}$. Под сопоставлением требований между версиями будем понимать установление соответствий между элементами R_{old} и R_{new} с целью выявления того, какое требование новой версии (если оно есть) соответствует данному требованию старой версии, и наоборот.

В такой постановке задача обнаружения изменений формулируется следующим образом. Для каждого требования r'_j из R_{new} необходимо определить его статус по отношению к R_{old} :

- неизмененное (Unchanged): в R_{old} существует требование r_i того же класса и с семантически эквивалентным содержанием, при этом допустимы различия в формулировке (например, пунктуация, перефразирование);
- измененное (Modified): в R_{old} существует требование-прообраз r_i , однако содержание требования претерпело изменения (например, изменились числовой параметр или тип требования);
- новое (New): соответствующего прообраза в R_{old} нет, требование добавлено в новую версию.

Для каждого требования r_i из R_{old} определим его актуальность в новой версии. Требование считается удаленным (Deleted), если в R_{new} не найдено требования, сопоставленного с r_i , что интерпретируется как исключение данной функциональности или ограничения из спецификации.

Таким образом, выходом метода является частичное отображение $M \subseteq R_{old} \times R_{new}$ – множество сопоставленных пар (r_i, r'_j) между версиями, с присвоенным каждому элементу статусом из множества

{Unchanged, Modified, New, Deleted}.

2. КОМПОЗИТНАЯ ФУНКЦИЯ СХОДСТВА ТРЕБОВАНИЙ

Ключевым элементом метода в предлагаемом подходе является композитный балл сходства $S(i, j)$, вычисляемый для каждой пары требований (r_i, r'_j) . Нами использован композитный балл, объединяющий несколько различных признаков, что позволяет учитывать смысловую близость формулировок, лексическое совпадение, совпадение типа требования и структурный контекст документа. Согласно проведенным исследованиям, наиболее эффективными признаются гибридные подходы, сочетающие в себе признаки различных уровней – от строкового сопоставления до учета структурной организации информации [8].

Для каждого требования вычислим векторное представление $e(r) = \text{SBERT}(\text{text}(r))$ с помощью модели SBERT, позволяющей получать семантические контекстуализированные эмбединги предложений [9]. Семантическое сходство двух требований определим как косинусное сходство их эмбедингов: $s_{\text{sem}}(i, j) = \cos(e(r_i), e(r'_j))$. Данный показатель отражает близость смыслового содержания требований и устойчив к перефразированию текста [9].

Для учета лексических изменений в формулировках введем коэффициент Жаккара по множествам токенов требований. Пусть $W(r)$ – множество токенов, полученных из текста требования после предобработки (приведение к нижнему регистру, удаление знаков пунктуации). Тогда показатель лексического сходства s_{lex} определяется как:

$$s_{\text{lex}}(i, j) = \frac{|W(r_i) \cap W(r'_j)|}{|W(r_i) \cup W(r'_j)|}.$$

Этот показатель отражает степень совпадения токенов в формулировках требований и позволяет различать требования со схожей семантикой, но с различной терминологией [10].

Дополнительным признаком является совпадение классов требований. Пусть $\text{class}(r)$ – метка класса требования из таксономии $\mathcal{C}$. Определим $s_{\text{class}}(i, j) = 1$, если $\text{class}(r_i) = \text{class}(r'_j)$, и 0 в противном случае. Совпадение класса усиливает вероятность того, что два требования являются одной и той же сущностью в двух версиях.

Учтем также структурный контекст документа. В технической документации требования, как правило, группируются по разделам (функциональным областям или качественным характеристикам). Учет структурного контекста важен в случаях кратких или универсальных формулировок, которые могут повторяться в различных разделах. Например, требование вида «сервер должен отвечать за 1 секунду» в новых версиях документа может быть помещено в различные разделы (например, в раздел, описывающий скорость поиска, или раздел, посвященный созданию функциональных элементов приложения). Тогда пусть h_i и h_j – заголовки разделов, в которых расположены требования r_i и r'_j соответственно. Сходство заголовков s_{hdr} аналогично определяется по эмбедингам SBERT как

$$s_{\text{hdr}}(i, j) = \cos(e(h_i), e(h_j)).$$

Нормализованную позицию требования в разделе зададим как $\text{pos}(r) = \frac{k_r}{N_r}$, где k_r – порядковый номер требования в разделе, а N_r – общее число требований в этом разделе. Значение $\text{pos}(r)$ таким образом лежит в отрезке $[0, 1]$. Близость расположения требований s_{pos} в соответствующих разделах равна

$$s_{\text{pos}}(i, j) = 1 - |\text{pos}(r_i) - \text{pos}(r'_j)|.$$

Чем ближе позиции, тем больше значение s_{pos} . Структурный контекст объединим в один признак:

$$s_{\text{ctx}}(i, j) = \lambda_1 s_{\text{hdr}}(i, j) + \lambda_2 s_{\text{pos}}(i, j).$$

В реализованном варианте были установлены значения $\lambda_1 = 0.7$ и $\lambda_2 = 0.3$. Более высокий вес показателя s_{hdr} позволяет снизить влияние случайных перестановок требований внутри раздела, которые могут возникнуть при редактировании документации.

Итоговую функцию сходства зададим взвешенной суммой с понижающим коэффициентом penalty при несовпадении числовых параметров:

$$S(i, j) = (\alpha s_{\text{sem}} + \beta s_{\text{lex}} + \gamma s_{\text{class}} + \delta s_{\text{ctx}}) \cdot \text{penalty}.$$

Использование penalty обусловлено тем, что изменение численного параметра (например, времени отклика с 3 до 1 с) изменяет суть требования, даже если семантический вектор остается почти неизменным. Веса зададим как $\alpha = 0.7$, $\beta = 0.1$, $\gamma = 0.1$, $\delta = 0.1$ из определения иерархии признаков, семантика рассмотрена в качестве базиса сопоставления [8].

3. ПОСТРОЕНИЕ ОТОБРАЖЕНИЯ И ПРИСВОЕНИЕ СТАТУСА СОПОСТАВЛЕННЫМ ТРЕБОВАНИЯМ

После вычисления матрицы сходства $S(i, j)$ для всех пар (r_i, r'_j) необходимо построить отображение M и присвоить статусы его элементам. Задача сопоставления допускает неполное покрытие: не каждое требование из R_{old} обязано иметь пару в R_{new} (статус Deleted), и не каждое из R_{new} – в R_{old} (статус New). Венгерский алгоритм решения задачи стремится к полному покрытию при создании пар, что в данном контексте может привести к сопоставлению семантически не связанных требований [11]. В связи с этим нами использован метод взаимных ближайших соседей (Mutual Nearest Neighbors, MNN). Пара (r_i, r'_j) включается в M , только если r'_j является ближайшим соседом r_i в R_{new} по баллу S , и одновременно r_i является ближайшим соседом r'_j в R_{old} , при этом $S(i, j) \geq \theta_{\text{match}}$. Под порогом сопоставления θ_{match} в дальнейшем понимаем минимальный балл сход-

ства, при котором пара требований допускается в отображение M . Таким образом, в M попадают исключительно взаимно наилучшие совпадения с достаточным уровнем сходства. Остальные требования не образуют пары и получают статус New (для R_{new}) или Deleted (для R_{old}).

Для разграничения неизмененных и измененных требований введем порог неизменности θ_{unch} . Это минимальный балл $S(i, j)$ для пар, уже включенных в M , при превышении которого сопоставленной паре присваивается статус Unchanged. При $S(i, j) < \theta_{\text{unch}}$ пара получает статус Modified.

Оба порога (θ_{match} и θ_{unch}) подбираем автоматически по распределению вычисленных баллов сходства. При наличии как новых, так и измененных/неизмененных пар требований распределение баллов $S(i, j)$, как правило, имеет бимодальный характер: одна мода сосредоточена в области высоких значений (сопоставленные требования), другая – в области низких (требования, не имеющие пары). Для разделения мод применим метод Отсу, позволяющий определить оптимальный порог бинарного разделения, максимизируя межклассовую дисперсию (и минимизируя внутриклассовую) [12]. Этот метод не требует априорных предположений о виде распределения и адаптируется к конкретным данным, что необходимо для учета разнообразия стилей документации [12]. Для θ_{match} анализируем распределение максимальных баллов сходства для каждого требования из R_{new} (максимум по r_i). Для θ_{unch} проанализируем распределение баллов по парам, уже включенным в M .

Итоговую классификацию по статусам выполним по следующим правилам:

- New: требование r_j' из R_{new} не имеет пары в M ;
- Deleted: требование r_i из R_{old} не имеет пары в M ;
- Unchanged: пара $(r_i, r_j') \in M$ и $S(i, j) \geq \theta_{\text{unch}}$;
- Modified: пара $(r_i, r_j') \in M$ и $\theta_{\text{match}} \leq S(i, j) < \theta_{\text{unch}}$.

4. ПРИМЕНЕНИЕ МЕТОДА К ТЕХНИЧЕСКОЙ ДОКУМЕНТАЦИИ

Для иллюстрации работы метода рассмотрим два сконструированных фрагмента технической документации в двух версиях. Первая версия имеет вид

Authentication

The system shall authenticate users using a username and password.

The system should process user requests within 3 seconds.

The system must log all authentication attempts.

Вторая версия содержит изменения по сравнению с первой:

User Authentication

The system shall authenticate users using a username and password.

The system should process user requests within 2 seconds.

The system should lock a user account after five failed login attempts.

Первая версия содержит раздел Authentication с тремя требованиями:

1) аутентификация по логину и паролю; 2) обработка запросов пользователя в течение 3 с; 3) логирование попыток аутентификации. Новая версия содержит раздел User Authentication с тремя требованиями: аутентификация по логину и паролю (формулировка без изменений); обработка запросов в течение 2 с (изменен числовой параметр); блокировка учетной записи после пяти неудачных попыток входа (новое требование). Третье требование предыдущей версии в новой версии отсутствует.

После извлечения и классификации требований для каждой версии алгоритм вычисляет компоненты S_{sem} , S_{lex} , S_{class} , S_{ctx} и итоговый $S(i, j)$ для всех пар (табл. 1).

- Для пары (r_2, r'_2) семантическое и лексическое сходства высоки, класс совпадает (нефункциональное требование, Performance), структурный контекст близок. Из-за различия числовых значений применен penalty. Итоговый балл оказался ниже порога неизменности, и паре присвоен статус Modified.

- Пара (r_1, r'_1) получила высокий балл $S(i, j)$ и соответствующий статус Unchanged благодаря семантическому сходству названий разделов.

- Требование r_3 не нашло прообраза в R_{old} , ему присвоен статус New. Аналогично требование r'_3 не нашло пары в R_{new} , его статусом стал Deleted.

Табл. 1. Вычисленные алгоритмом значения S_{sem} , S_{lex} , S_{class} , S_{ctx} в примере.

Номер требования из R_{new}	Номер требования из R_{old}		
	1	2	3
1	1.00; 1.00; 1.0; 0.95	0.31; 0.12; 0.0; 0.85	0.46; 0.16; 1.0; 0.75
2	0.30; 0.12; 0.0; 0.85	0.98; 0.80; 1.0; 0.95	0.32; 0.24; 0.0; 0.85
3	0.57; 0.13; 1.0; 0.75	0.27; 0.14; 0.0; 0.85	0.47; 0.19; 1.0; 0.95

На рис. 1 представлена вычисленная итоговая оценка сходства $S(i, j)$ для каждой пары требований рассмотренных версий. Таким образом, метод корректно выделил все четыре типа сопоставления из рассмотренного примера.

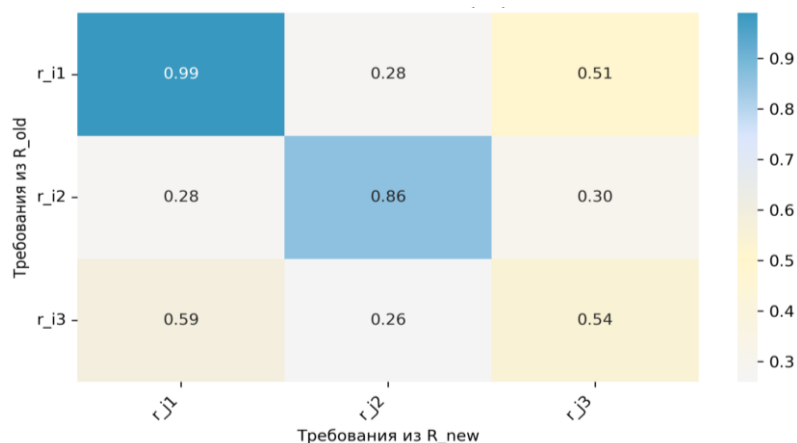


Рис. 1. Итоговая оценка сходства пар требований из примера.

Дополнительно был реализован прототип системы, объединяющий этапы обработки документации от извлечения требований до их сопоставления между версиями. На рис. 2 представлен фрагмент результата работы прототипа на приведенном примере с визуальным выделением статусов требований, а также измененных в них атрибутов между версиями.

Статус	Тип (V1 → V2)	Текст V1	Текст V2	Раздел	Сходство
UNCHANGED	Security (SE) → Security (SE)	The system shall authenticate users using a username and password.	The system shall authenticate users using a username and password.	User Authentication	0.99
MODIFIED	Performance (PE) → Performance (PE)	The system should process user requests within 3 seconds.	The system should process user requests within 2 seconds.	User Authentication	0.86
DELETED	Security (SE) → -	The system must log all authentication attempts.		Authentication	0.00
NEW	- → Security (SE)		The system should lock a user account after five failed login attempts.	User Authentication	0.00

Рис. 2. Фрагмент результата работы прототипа системы на примере.

Дополнительно для оценки точности метода был использован открытый набор TF_CDN, содержащий пары требований к ПО с экспертными метками о наличии или отсутствии связи между требованиями (категории дубликатов

(Duplicate), конфликтующих (Conflict) и пар, нейтральных по отношению друг к другу (Neutral)) [13]. В рамках эксперимента категории Duplicate и Conflict интерпретировались как наличие связи (Should Match), а Neutral – как отсутствие связи (No Relation). В табл. 2 представлены полученные значения метрик для предложенного метода.

Табл. 2. Результаты применения метода на датасете TF_CDN.

Наименование класса	Значение метрики		
	Precision	Recall	F1-score
No relation	0.97	0.84	0.90
Should Match	0.93	0.99	0.96

Для класса Should Match достигнуты показатели Precision порядка 0.93 и Recall порядка 0.99. Для No Relation метрика Precision составила около 0.97, Recall – 0.84. Общая точность (Accuracy) классификации пар равна 0.94. Высокое значение Recall для класса Should Match указывает на малое число пропущенных связей между требованиями, что является особенно важным показателем для задачи поиска пар изменений. Относительно высокое значение Precision (0.93) свидетельствует о низкой вероятности ложного сопоставления семантически несвязанных требований. Таким образом, полученные результаты подтверждают, что предложенная композитная функция сходства и процедура выбора порогов на основе метода Отсу адекватны для работы с парами требований из промышленных спецификаций.

ЗАКЛЮЧЕНИЕ

Предложен формализованный метод автоматического обнаружения изменений требований к ПО в версиях технической документации. Исходными данными служат множества требований, извлеченных и классифицированных для каждой версии. Задача сведена к построению частичного отображения между множествами требований и присвоению каждому требованию статуса: неизменное, измененное, новое или удаленное.

Ключевыми элементами метода является композитная функция сходства, объединяющая семантическое сходство (с помощью эмбедингов SBERT), лексическое сходство (коэффициент Жаккара), совпадение класса и структурный контекст (заголовки разделов и относительная позиция требований). Введен понижающий коэффициент при несовпадении числовых параметров в содержании требований. Построение отображения осуществлено по правилу взаимных ближайших соседей (MNN) с вычисляемыми с помощью метода Отсу динамическими порогами по распределениям баллов сходства.

Применение метода на открытом наборе пар требований показало достижение точности идентификации пар на уровне порядка 0.94 при высокой полноте (Recall = 0.99) по классу наличия связей между требованиями, что подтверждает применимость подхода для анализа версий технической документации в терминах изменений требований к ПО.

СПИСОК ЛИТЕРАТУРЫ

1. ISO/IEC/IEEE 29148:2018. Systems and software engineering – Life cycle processes – Requirements engineering.
URL: <https://www.iso.org/standard/72089.html>.
2. Sommerville I., Sawyer P. Requirements Engineering: A Good Practice Guide. Chichester: Wiley, 1997. 404 p.
3. Hunt J.W., MacIlroy M.D. An algorithm for differential file comparison. Murray Hill: Bell Laboratories, 1976. 9 p.
4. Myers E.W. An O (ND) difference algorithm and its variations // Algorithmica. 1986. No. 1. P. 251–266.
5. Gotel O.C.Z., Finkelstein C.W. An analysis of the requirements traceability problem // Труды Международной научной конференции «IEEE international conference on requirements engineering», Колорадо-Спрингс, США, 18–22 апреля 1994. IEEE: 1994, С. 94–101.
6. Cleland-Huang J., Gotel O. C., Huffman Hayes J., Mäder P., Zisman A. Software traceability: trends and future directions // Труды Международной научной конференции «Future of Software Engineering, FOSE», Хайдарабад, Индия, 31 мая – 7 июня 2014. ACM: 2014, С. 55–69.

7. Савинчева А.С. Разработка интеллектуальной системы семантического анализа изменений в версиях технической документации программного обеспечения: выпускная квалификационная работа магистра. Казань: КФУ, 2026. С.19-40. URL: https://www.kpfu.ru/student_diplom/10.160.178.20_RIDB8JMIV54UV013SE2J41C1TVBNFK-SUB2Z5YB281KKY836_2X_F_Savincheva.pdf.
8. Kumar L., Upadhye N.S., Piedy K. Advances and Challenges in Semantic Textual Similarity: A Comprehensive Survey. URL: <https://arxiv.org/abs/2601.03270>.
9. Reimers N., Gurevych I. Sentence-bert: Sentence embeddings using siamese bert-networks // Труды международной научной конференции «2019 Conference on empirical methods in natural language processing and the 9th International joint conference on natural language processing (EMNLP-IJCNLP)», Гонконг, Китай, 3–7 ноября 2019. ACL: 2019, С. 3982–3992.
10. Goma W.H., Fahmy A.A. A survey of text similarity approaches // International Journal of Computer Applications. 2013. No. 13. P. 13–18.
11. Kuhn H.W. The Hungarian method for the assignment problem // Naval research logistics quarterly. 1955. No. 1-2. P. 83–97.
12. Otsu N. et al. A threshold selection method from gray-level histograms // Automatica. 1979. No. 11. P. 285–296.
13. TF-CDN: Initial Release. URL: <https://zenodo.org/records/10830023>.

METHOD FOR SEMANTIC ANALYSIS OF SOFTWARE REQUIREMENTS CHANGES IN TECHNICAL DOCUMENTATION VERSIONS

A. S. Savincheva¹ [0009-0005-0756-6001], Z. A. Enikeeva² [0000-0002-2468-7455],
K. S. Enikeev³ [0009-0000-6020-9912]

¹⁻³*Institute of Information Technologies and Intelligent Systems, Kazan (Volga Region)
Federal University, Kazan, Russia*

¹asanvlit@gmail.com, ²ZAEnikeeva@kpfu.ru, ³KSEnikeev@kpfu.ru

Abstract

A formalized method for the automated detection of software requirement changes during the evolution of technical documentation versions is described. The input data consist of sets of requirements extracted and classified into functional and non-functional categories, with further granularity regarding non-functional requirement types (e.g., performance, security, availability, etc.) for each documentation version. The task of matching requirements between versions is reduced to constructing a partial mapping between the requirement sets and assigning one of the following statuses to each requirement: Unchanged, Modified, New, or Deleted.

A composite similarity function for requirement pairs is proposed, integrating semantic similarity based on vector representations (embeddings), lexical similarity, requirement class consistency, and the structural context of the document. The results of applying the proposed method to an open dataset of requirement pairs are presented. A prototype system for requirement change analysis in technical documentation has been developed.

Keywords: *software requirements, requirements change analysis, documentation version mapping, requirements traceability, semantic similarity, mutual nearest neighbors' method, Otsu's method.*

REFERENCES

1. ISO/IEC/IEEE 29148:2018. Systems and software engineering – Life cycle processes – Requirements engineering.
URL: <https://www.iso.org/standard/72089.html>.

2. *Sommerville I., Sawyer P.* Requirements Engineering: A Good Practice Guide. Chichester: Wiley, 1997. 404 p.
3. *Hunt J.W., MacIroy M.D.* An algorithm for differential file comparison. Murray Hill: Bell Laboratories, 1976. 9 p.
4. *Myers E.W.* An O (ND) difference algorithm and its variations // *Algorithmica*. 1986. No. 1. P. 251–266.
5. *Gotel O.C.Z., Finkelstein C.W.* An analysis of the requirements traceability problem // *Proceedings of the IEEE International Conference on Requirements Engineering*, Colorado Springs, USA, April 18–22, 1994. IEEE: 1994, pp. 94–101.
6. *Cleland-Huang J., Gotel O. C., Huffman Hayes J., Mäder P., Zisman A.* Software traceability: trends and future directions // *Proceedings of the Future of Software Engineering, FOSE*, Hyderabad, India, May 31 – June 7, 2014. ACM: 2014, pp. 55–69.
7. *Savincheva A.S.* Development of an intelligent system for semantic analysis of changes between versions of software technical documentation: Master's thesis. Kazan: KFU, 2026. pp. 19-40. URL: https://www.kpfu.ru/student_diplom/10.160.178.20_RIDB8JMIV54UV013SE2J41C1TVBNFK-SUB2Z5YB281KKY836_2X_F_Savincheva.pdf.
8. *Kumar L., Upadhye N.S., Piedy K.* Advances and Challenges in Semantic Textual Similarity: A Comprehensive Survey. URL: <https://arxiv.org/abs/2601.03270>.
9. *Reimers N., Gurevych I.* Sentence-bert: Sentence embeddings using siamese bert-networks // *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China, November 3–7, 2019. ACL: 2019, pp. 3982–3992.
10. *Gomaa W.H., Fahmy A.A.* A survey of text similarity approaches // *International Journal of Computer Applications*. 2013. No. 13. P. 13–18.
11. *Kuhn H.W.* The Hungarian method for the assignment problem // *Naval research logistics quarterly*. 1955. No. 1-2. P. 83–97.
12. *Otsu N. et al.* A threshold selection method from gray-level histograms // *Automatica*. 1979. No. 11. P. 285–296.
13. TF-CDN: Initial Release. URL: <https://zenodo.org/records/10830023>.

СВЕДЕНИЯ ОБ АВТОРАХ



САВИНЧЕВА Ангелина Сергеевна – магистрант Института информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета.

Angelina Sergeevna SAVINCHEVA – Master’s student of the Institute of Information Technologies and Intelligent Systems, Kazan (Volga region) Federal University.

email: asanvlit@gmail.com

ORCID: 0009-0005-0756-6001



ЕНИКЕЕВА Зульфура Аснафовна – к. ф.-м. н., доцент Института информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета.

Zulfira Asnafovna ENIKEEVA – Candidate of Physical and Mathematical Sciences, Docent at the Institute of Information Technologies and Intelligent Systems, Kazan (Volga Region) Federal University.

email: ZAEnikeeva@kpfu.ru

ORCID: 0000-0002-2468-7455



ЕНИКЕЕВ Камиль Шамилевич – старший преподаватель кафедры программной инженерии Института информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета.

Kamil Shamilevich ENIKEEV – Senior lecturer of the department of software engineering at the Institute of Information Technologies and Intelligent Systems, Kazan (Volga Region) Federal University.

email: KSEnikeev@kpfu.ru

ORCID: 0009-0000-6020-9912

Материал поступил в редакцию 17 мая 2026 года