

УДК 005.8+004.4

DOI: [10.26102/2310-6018/2026.60.9.021](https://doi.org/10.26102/2310-6018/2026.60.9.021)

Метод трансформации функциональных технических требований в задачи системы управления проектами

А.В. Ечин✉

*Волгоградский государственный технический университет, Волгоград,
Российская Федерация*

Резюме. В условиях глобальной информатизации и автоматизации бизнес-процессов пропорционально возрастают объемы разрабатываемого программного обеспечения и, как следствие, количество технических и функциональных требований. Это обуславливает острую необходимость автоматизации процесса трансформации проектной документации в задачи систем управления проектами. Ручная декомпозиция масштабных технических заданий и их перенос в трекеры задач не только требуют значительных временных затрат, но и критически зависят от человеческого фактора. Подобный подход неизбежно приводит к увеличению сроков планирования и рассинхронизации между изначальными бизнес-целями и итоговыми артефактами разработки, создавая системное ограничение на ранних этапах жизненного цикла ИТ-проекта. В данной статье представлен метод автоматической генерации структурированных данных в формате JSON для импорта в трекеры задач. Ключевой особенностью разработанного метода является внедрение механизма валидации JSON-схемы и цикла итеративного исправления синтаксических и логических ошибок. Экспериментальная апробация показала высокую эффективность предложенного подхода: доля валидных JSON-документов составила 95 %, что значительно превышает показатель прямого промптинга больших языковых моделей (LLM), равный 60 %. Точность воспроизведения иерархической структуры задач (Epic → Story → Subtask) достигла 88 %. Результаты исследования подтверждают, что метод снижает трудозатраты на рутинный перенос данных и минимизирует риски ошибок при планировании ИТ-проектов.

Ключевые слова: инженерия требований, RAG, большие языковые модели, управление проектами, векторизация текста, трансформация требований.

Благодарности: Исследование выполнено при поддержке «Центра цифровых научно-образовательных проектов и разработок в сфере промышленного искусственного интеллекта» Ц2RED-ИИ ВолгГТУ, созданного в рамках реализации образовательных программ топ-уровня в сфере искусственного интеллекта (Соглашение № 70-2025-000756).

Для цитирования: Ечин А.В. Метод трансформации функциональных технических требований в задачи системы управления проектами. *Моделирование, оптимизация и информационные технологии*. 2026;14(9). URL: <https://moitvvt.ru/ru/journal/article?id=2574> DOI: 10.26102/2310-6018/2026.60.9.021

Method for the functional requirements transformation into tasks of the project management system

A.V. Echin✉

Volgograd State Technical University, Volgograd, the Russian Federation

Abstract. As global computerization and business process automation continue to advance, the volume of software development and, consequently, the number of technical and functional requirements are increasing correspondingly. This growth creates an urgent need to automate the conversion of project documentation into tasks within project management systems (PMS). Manual decomposition of large

technical specifications and their subsequent transfer to issue trackers is not only time-consuming but also highly dependent on human factors. As a result, this approach extends planning timelines and creates a mismatch between initial business objectives and final development artifacts, thereby imposing a systemic constraint in the early stages of the IT project lifecycle. This paper presents a method to automate the generation of structured JSON data for import into issue trackers. A key feature of the developed method is the integration of a JSON Schema validation mechanism and an iterative error-correction loop to handle both syntactic and logical errors. Experimental validation demonstrated the high efficiency of the proposed approach: the share of valid JSON documents reached 95 %, significantly exceeding the 60 % baseline achieved by directly prompting large language models (LLMs). The replication accuracy of the hierarchical task structure (Epic → Story → Subtask) reached 88 %. Furthermore, the method ensures end-to-end requirements traceability, enabling the tracking of links from specific sections of the technical specification to the final development tasks. The research results confirm that the method reduces labor costs associated with routine data migration and minimizes error risks in IT project planning.

Keywords: requirements engineering, RAG, large language models, project management, text vectorization, requirements transformation.

Acknowledgments: The study was carried out with the support of the Center for Digital Scientific and Educational Projects and Developments in the Field of Industrial Artificial Intelligence (C2RED-AI) of Volgograd State Technical University, created as part of the implementation of top-level educational programs in the field of artificial intelligence (Agreement No. 70-2025-000756).

For citation: Echin A.V. Method for the functional requirements transformation into tasks of the project management system. *Modeling, Optimization and Information Technology*. 2026;14(9). (In Russ.). URL: <https://moitvvt.ru/ru/journal/article?id=2574> DOI: 10.26102/2310-6018/2026.60.9.021

Введение

Современные процессы промышленной разработки программного обеспечения характеризуются значительным разрывом между формальной документацией (Техническое Задание (ТЗ), Спецификация требований [1]) и инструментами оперативного управления проектами. Ручной перенос требований из документов в задачи трудоемок, подвержен ошибкам интерпретации и приводит к потере данных на протяжении жизненного цикла разработки [2, 3]. Согласно исследованиям в области инженерии требований, до 40 % дефектов программного обеспечения возникают на этапе передачи требований от аналитиков к разработчикам [4]. Основная причина заключается в том, что текстовые требования из технической документации не имеют прямой связи с объектами системы управления проектами (СУП, Система Управления Проектами), что затрудняет верификацию полноты реализации и поддержку актуальности документации [5, 6].

Прямое использование больших языковых моделей (LLM) для конвертации требований в задачи СУП сталкивается с рядом критических проблем [7]. При обработке документов целиком модель допускает ошибки валидации JSON в 40 % случаев, что делает последующий импорт в СУП технически невозможным. Также без явного указания структурного контекста корректное воспроизведение иерархических связей между задачами (Epic → Story → Subtask) оказывается нарушенным. Наконец, процесс усложняется подверженностью модели галлюцинациям, из-за чего она генерирует артефакты и требования, полностью отсутствующие в исходном документе.

Для решения указанных проблем в данной работе предлагается метод автоматической трансформации функциональных требований в объектную модель СУП на основе архитектуры RAG (Retrieval-Augmented Generation) [8].

Целью данной работы является исследование влияния стратегий трансформации функциональных требований на точность формирования структурированных объектов СУП и разработка метода, обеспечивающего сохранение иерархической структуры требований и корректность формируемых данных.

Для достижения поставленной цели в рамках исследования решается ряд последовательных задач. На начальном этапе формализуется процесс преобразования функциональных требований в объекты СУП. На основе данной формализации разрабатывается метод трансформации функциональных требований (МТФТ), функционирующий на базе архитектуры RAG. Важной составляющей предлагаемого решения является реализация механизма валидации выходных данных, включающего цикл итеративного исправления синтаксических и логических ошибок. Завершающим этапом исследования является сравнительная оценка базовой стратегии прямой генерации средствами LLM и предложенного метода по показателям корректности структурированных данных и точности воспроизведения иерархической структуры задач.

Материалы и методы

Для формализации задачи трансформации функциональных требований введем необходимые обозначения и определения.

Пусть $\mathcal{D}$ обозначает корпус ТЗ, а $d \in \mathcal{D}$ – отдельный документ. После этапа структурного анализа документ представляется в виде ориентированного графа:

$$G = (V, Q), \quad (1)$$

где V – множество структурных единиц документа (разделов, подразделов и элементов текста), а $Q \subseteq V \times V$ – множество ребер, отражающих иерархическую структуру документа.

Семантическое содержание документа описывается с использованием предметной онтологии:

$$O = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle, \quad (2)$$

где $\mathcal{T}$ – множество типов сущностей, $\mathcal{R}$ – множество отношений между сущностями, $\mathcal{A}$ – множество атрибутов. В результате семантического анализа формируется множество извлеченных сущностей:

$$\mathcal{E} = \{e_1, e_2, \dots, e_m\}. \quad (3)$$

Каждая сущность $e_i \in \mathcal{E}$ соответствует определенному фрагменту текста документа и характеризуется типом $\text{type}(e_i) \in \mathcal{T}$.

В рамках данной работы интерес представляет подмножество сущностей, соответствующих функциональным требованиям системы:

$$\mathcal{E}_{func} = \{e_i \in \mathcal{E} \mid \text{type}(e_i) = \text{FUNC_REQ}\}. \quad (4)$$

Выбор подмножества $\mathcal{E}_{func}$ обусловлен семантикой задачи трансформации: только функциональные требования описывают действия системы, подлежащие реализации в виде задач разработки. Другие типы сущностей (NON_FUNC_REQ, OPER_COND, TERM и др., полный перечень приведен в [9, 10]) могут использоваться как дополнительный контекст при генерации, но не являются самостоятельными объектами преобразования в M_{PM} .

Для обеспечения связи требований к структуре документа вводится отображение $\Phi: \mathcal{E} \rightarrow V$, связывающее сущность с вершиной графа G . Множество вершин, содержащих функциональные требования, определяется как:

$$V_{FR} = \{v \in V \mid \exists e \in \mathcal{E}_{func}: \Phi(e) = v\}. \quad (5)$$

Для обеспечения автоматической обработки каждое функциональное требование нормализуется – приводится к семантическому кортежу:

$$r_i = \langle \text{text}_i, \text{context}_i, \text{src}_i \rangle, \quad (6)$$

где text_i – исходный текст требования сущности $e_k \in \mathcal{E}_{func}$; context_i – дополнительные условия или ограничения выполнения требования; $\text{src}_i \in V_{FR}$ – ссылка на вершину графа структуры документа.

Множество нормализованных функциональных требований определяется как:

$$\mathcal{R} = \{r_i \mid e_i \in \mathcal{E}_{func}\}. \quad (7)$$

Для использования требований в архитектуре RAG вводится функция векторизации ϕ :

$$\phi: \mathcal{R} \rightarrow \mathbb{R}^n, \quad (8)$$

отображающая каждое нормализованное требование в векторное пространство размерности n (где n определяется архитектурой модели эмбедингов). Векторное представление требования определяется как:

$$\vec{v}_i = \phi(r_i). \quad (9)$$

На основе полученных векторных представлений формируется векторная база знаний функциональных требований:

$$\mathcal{V}_{DB} = \{\vec{v}_i \mid r_i \in \mathcal{R}\}. \quad (10)$$

При формировании задач разработки используется механизм поиска релевантного контекста. Для пользовательского запроса q процедура извлечения определяется как поиск k ближайших соседей по метрике косинусного сходства:

$$\text{Context}(q) = \text{TopK}_k(\mathcal{V}_{DB}, \cos(\phi(q), \vec{v}_i)), \quad (11)$$

где $\cos(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$. В качестве контекста для языковой модели используются нормализованные требования r_i , соответствующие найденным векторам.

Целевая модель СУП представляется в виде графа задач:

$$M_{PM} = \langle \mathcal{I}, \mathcal{L} \rangle, \quad (12)$$

где $\mathcal{I}$ – множество задач разработки (issues), $\mathcal{L} \subseteq \mathcal{I} \times \mathcal{I}$ – множество связей между задачами. Каждая задача $i \in \mathcal{I}$ характеризуется типом:

$$\tau(i) \in \{\text{Epic}, \text{Story}, \text{Subtask}\}. \quad (13)$$

Процесс преобразования функциональных требований в задачи разработки формально представляется функцией:

$$F_{trans}: \mathcal{P}(\mathcal{R}) \times \text{Schema}_{JSON} \rightarrow \mathcal{P}(M_{PM}), \quad (14)$$

где $\mathcal{P}(\cdot)$ обозначает булеан множества, Schema_{JSON} задает формальные ограничения на структуру выходных данных. Языковая модель генерирует JSON-представление задач на

основе переданного контекста требований и заданной схемы, после чего результат проходит этап валидации:

$$\text{Validate}(J, \text{Schema}_{JSON}) = \begin{cases} \text{true}, & \text{если } J \models \text{Schema}_{JSON} \\ \text{false}, & \text{иначе} \end{cases}. \quad (15)$$

При $\text{Validate}(J, \text{Schema}_{JSON}) = \text{false}$ применяется алгоритм итеративной валидации до достижения валидного состояния или исчерпания лимита итераций.

Описание метода трансформации функциональных требований. Предложенный МТФТ реализован в виде многоэтапного пайплайна, объединяющего структурный анализ, семантическую обработку и генерацию на основе RAG. Общая схема метода представлена на Рисунке 1.



Рисунок 1 – Общая схема МТФТ

Figure 1 – General scheme of the method for the transformation of functional requirements

МТФТ включает следующие этапы:

Этап 1 – структурный анализ, где исходный документ $d \in \mathcal{D}$ проходит верификацию на соответствие нормативному шаблону [11]. Результатом является граф (1). Модуль визуализации фиксирует процент соответствия шаблону, например, 66,7 % разделов и 83,3 % подразделов (Рисунок 2).

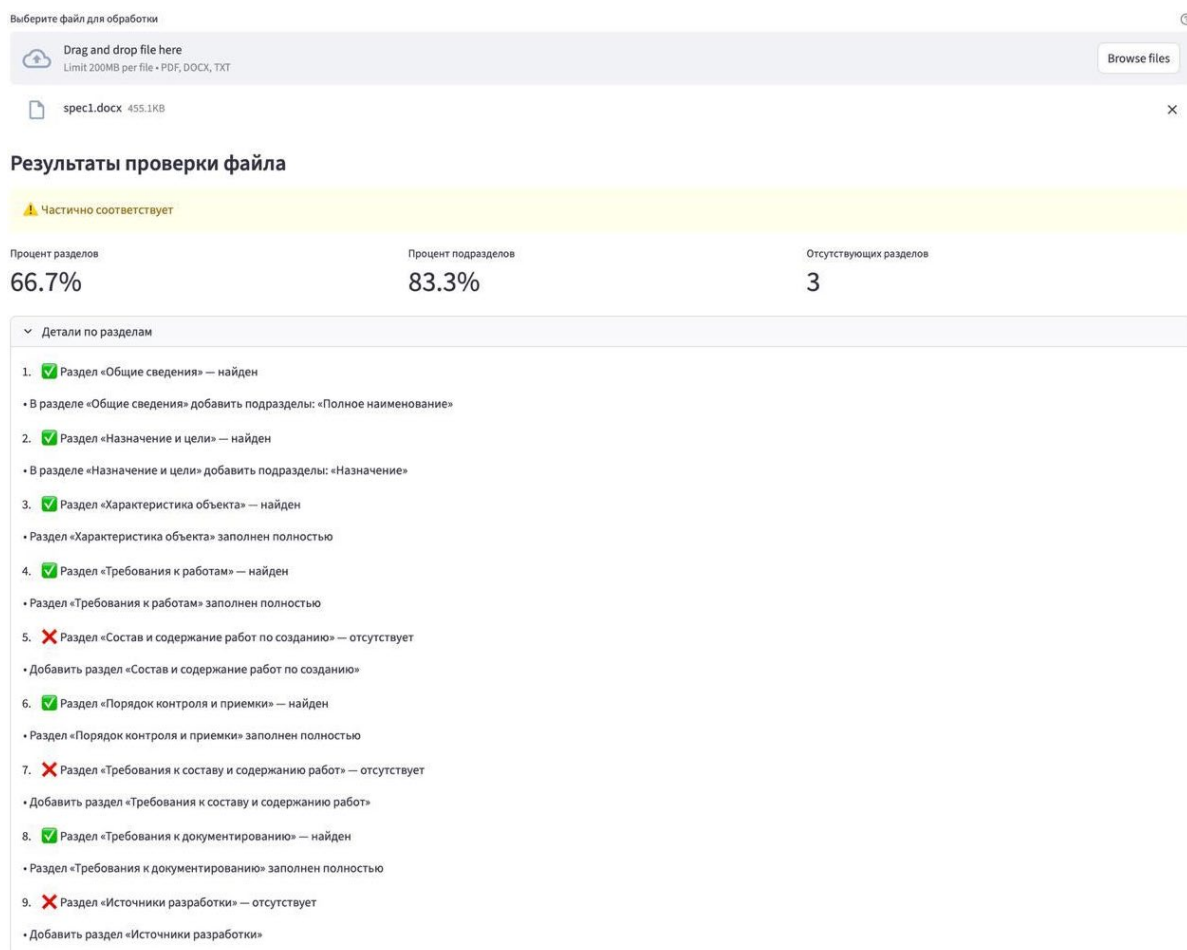


Рисунок 2 – Программная реализация этапа 1 МТФТ

Figure 1 – Software implementation of Stage 1 of the method for the transformation of functional requirements

Этап 2 – семантическое извлечение. На основе предметной онтологии (2) выполняется извлечение сущностей $\mathcal{E}$ всех типов (FUNC_REQ, NON_FUNC_REQ, COMP, SYS, OPER_COND и др.) с использованием NER-модели на базе ruRoBERTa-large [8]. Модель обучена с использованием библиотеки spaCy¹ на корпусе из 22 размеченных ТЗ и демонстрирует F1-score = 0,89 для класса FUNC_REQ. И для последующей трансформации в задачи СУП отбирается только подмножество функциональных требований $\mathcal{E}_{func}$ по формуле (4). Модуль мониторинга фиксирует количество извлеченных сущностей (Рисунок 3).

¹ Montani I., Honnibal M., Boyd A., Van Landeghem S., Peters H. *explosion/spaCy: v3.7.2: Fixes for APIs and requirements*. Zenodo. URL: <https://doi.org/10.5281/zenodo.1212303> (дата обращения: 19.06.2026).

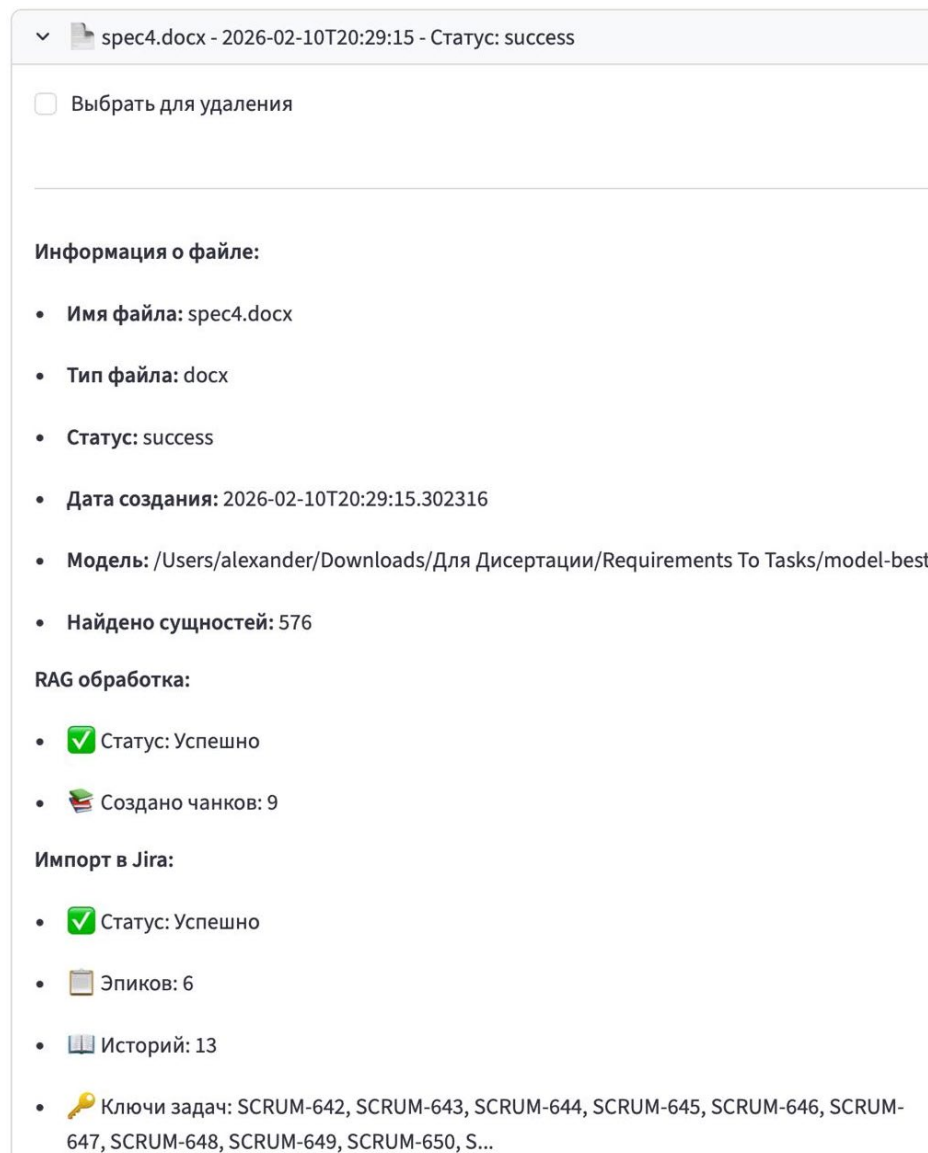


Рисунок 3 – Программная реализация этапа 2 МТФТ

Figure 3 – Software implementation of Stage 2 of the method for the transformation of functional requirements

На этапе 3 каждое требование $e_i \in \mathcal{E}_{func}$ приводится к нормализованному виду r_i по формуле (6).

На этапе 4 нормализованные требования $\mathcal{R}$ отображаются в векторное пространство $\mathbb{R}^n$ по формуле (8). Размерность вектора $n = 768$ определяется архитектурой модели эмбедингов ruRoBERTa.

Этап 5 – RAG-генерация. Формируется пользовательский запрос q . Механизм извлечения по формуле (11) выбирает релевантные требования из $\mathcal{V}_{DB}$ ($k = 5$ ближайших соседей, порог сходства $\theta = 0,75$), которые передаются языковой модели вместе со схемой $Schema_{JSON}$.

На этапе 6 сгенерированный JSON проходит проверку по формуле (15). При успешной валидации задачи импортируются в СУП с присвоением идентификаторов, например, SCRUM-642...SCRUM-650 (Рисунок 4). При ошибке валидации применяется механизм итеративного исправления (лимит – 3 итерации).

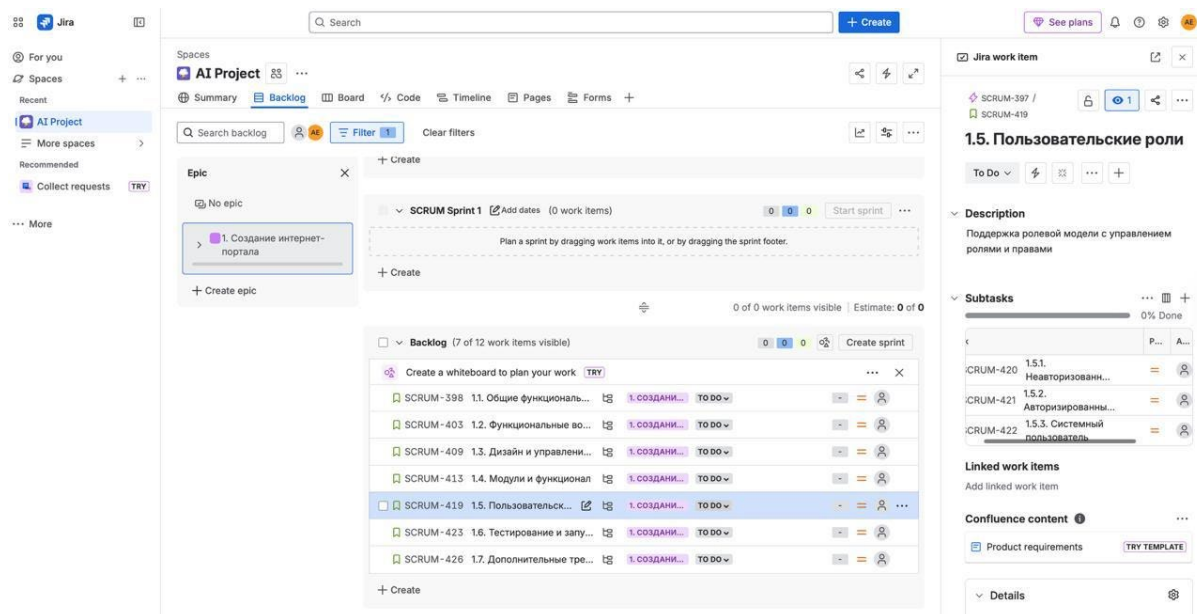


Рисунок 4 – Программная реализация этапа 6 МТФТ
Figure 4 – Software implementation of Stage 6 of the method for the transformation of functional requirements

Механизм валидации JSON с алгоритмом итеративной валидации. Прямая генерация JSON языковой моделью без валидации приводит к ошибкам в ~40 % случаев. Для обеспечения промышленной пригодности разработан механизм итеративной валидации:

Algorithm 1: JSON Validation with Self-Correction

Input: Context(q), Schema_JSON, LLM, max_iterations = 3

Output: Valid JSON or Error

1: $J_cand \leftarrow LLM.generate(Context(q), Schema_JSON)$

2: for $i = 1$ to $max_iterations$ do

3: if $Validate(J_cand, Schema_JSON) = true$ then

4: return J_cand

5: else

6: $error_message \leftarrow Parser.get_error(J_cand)$

7: $J_cand \leftarrow LLM.correct(J_cand, error_message)$

8: end if

9: end for

10: return $Error("Validation failed after \{max_iterations\} iterations")$

Валидатор выявляет синтаксические ошибки JSON, нарушение схемы (отсутствие обязательных полей, некорректные типы данных), нарушение иерархии (отсутствие связи parent_of между Subtask и Story), дублирование задач.

Визуализация результатов и мониторинг. Для обеспечения прозрачности процесса трансформации разработан модуль визуализации, фиксирующий все этапы пайплайна (Рисунок 4). Метрики, отображаемые модулем, представлены в Таблице 1.

Модуль обеспечивает возможность отладки пайплайна: при ошибке на любом этапе аналитик может определить, на каком этапе произошла деградация качества.

На Рисунке 2 представлен пример структурной верификации документа spec1.docx. Документ получил статус «Частично соответствует» (66,7 % разделов, 83,3 % подразделов). Отсутствуют разделы 5 «Состав и содержание работ по созданию», 7 «Требования к составу и содержанию работ» и 9 «Источники разработки». Несмотря на

неполное соответствие шаблону, документ был допущен к обработке, так как ключевые разделы с функциональными требованиями (раздел 4) присутствуют и заполнены полностью.

Таблица 1 – Метрики

Table 1 – Metrics

Метрика	Значение
Статус структурной верификации	66,7 % разделов
Количество извлечённых сущностей	576
Статус RAG-обработки	Успешно
Количество созданных Epic	6
Количество созданных Story	13
Идентификаторы	SCRUM-642...SCRUM-650
Статус импорта в СУП	Успешно

На Рисунке 4 показан интерфейс СУП после успешного импорта задач. Видна иерархическая структура:

Epic: «1. Создание интернет-портала»

Story: «1.5. Пользовательские роли» (SCRUM-419)

Subtasks: «1.5.1. Неавторизованные...», «1.5.2. Авторизованные...», «1.5.3. Системный пользователь».

На Рисунке 3 представлен лог обработки документа spec4.docx, где зафиксированы все этапы пайплайна:

- найдено сущностей: 576 (все типы онтологии);
- RAG обработка: успешно, создано 9 чанков;
- импорт в СУП: успешно;
- эпиков: 6, историй: 13;
- ключи задач: SCRUM-642...SCRUM-650.

Экспериментальное исследование. Базовый подход предполагает прямой запрос к LLM с просьбой извлечь функциональные требования из всего ТЗ целиком и оформить их в формате JSON для импорта в СУП.

В экспериментах использовались LLM-модели DeepSeek, GPT-4.0 mini.

К ограничениям базового подхода можно отнести обработку всего документа без ограничения контекста, отсутствие валидации выходных данных; отсутствие механизма исправления ошибок.

Для оценки эффективности предложенного метода был сформирован корпус из 5 ТЗ различной предметной направленности (информационные системы, автоматизированные комплексы, программные платформы). Документы прошли предварительную структурную верификацию, при этом процент соответствия нормативному шаблону составил от 60 % до 75 % (пример проверки представлен на Рисунке 2).

Характеристики корпуса: количество документов: 5 ТЗ (формат DOCX); общий объем: приблизительно 150 страниц; количество извлеченных сущностей: 2,400+ (все типы онтологии); количество функциональных требований: ~1,500 (тип FUNC_REQ); модель NER: ruRoBERTa-large (F1-score = 0.89 для FUNC_REQ).

Методы оценки и метрики. Для количественной оценки качества трансформации требований в задачи СУП были выбраны следующие метрики:

1. Valid JSON Rate (VJR) – доля ответов языковой модели, успешно прошедших валидацию на соответствие синтаксису JSON и схеме.

2. Hierarchy Accuracy (HA) – воспроизведения иерархической структуры задач (Epic → Story → Subtask), оцениваемая экспертом.

Результаты

Результаты сравнительного анализа базового метода и МТФТ представлены в Таблице 2.

Таблица 2 – Сравнение качества трансформации требований
Table 2 – Comparison of requirements transformation quality

Способ	Valid JSON Rate (%)	Hierarchy Accuracy (%)
Базовый метод (только LLM)	60	45
МТФТ	95	88

Valid JSON Rate. Базовый метод показал нестабильность генерации: в 40 % случаев модель допускала синтаксические ошибки JSON, что делало автоматический импорт в СУП невозможным. Предлагаемый метод обеспечил 95 % валидных результатов благодаря алгоритму итеративной валидации: 83 % ошибок исправлялись на первой итерации, 12 % – на второй, 5 % потребовали ручной корректировки.

Hierarchy Accuracy. При прямом запросе к LLM модель часто игнорировала иерархическую структуру: подзадачи не связывались с родительскими историями, эпика не группировались логически. Предлагаемый метод, использующий структурный граф G, обеспечил точность 88 % за счет явного указания связей L на основе вложенности разделов ТЗ.

Результаты эксперимента показывают влияние выбранной стратегии трансформации функциональных требований на качество формирования объектов СУП: применение структурного анализа, семантической фильтрации и RAG-генерации с итеративной валидацией повысило Valid JSON Rate с 60 до 95 %, а Hierarchy Accuracy – с 45 до 88 % по сравнению с прямой генерацией LLM.

Обсуждение

Несмотря на продемонстрированную эффективность, предложенный метод имеет ряд ограничений, которые необходимо учитывать при его промышленном внедрении. Качество итогового результата критически зависит от точности работы модуля распознавания именованных сущностей (NER): если функциональное требование не будет извлечено на этапе $\mathcal{E}_{func}$, его последующая трансформация в задачу окажется невозможной. При этом текущая NER-модель оптимизирована исключительно для русскоязычных ТЗ, что ограничивает применимость метода к англоязычным или многоязычным документам без дополнительного дообучения или замены базовой модели. Также метод предъявляет строгие требования к предварительной структурной верификации входных данных. Алгоритм предполагает, что документ d соответствует заданному шаблону; наличие структурных расхождений более чем на 50 % свидетельствует о некорректной иерархии разделов. Сохраняются объективные ограничения, связанные с размером контекстного окна LLM. Несмотря на применение архитектуры RAG, генерация сложных многоуровневых иерархий задач может превышать лимиты контекста, что в ряде случаев требует привлечения дополнительной постобработки.

Перспективными направлениями дальнейшего развития предложенного метода являются расширение функциональности за счет обработки мультимодальных данных и повышение универсальности решения. В частности, планируется интеграция

vision-language моделей (VLM) для извлечения требований из таблиц, диаграмм и схем, присутствующих в ТЗ, что позволит охватить нетекстовые элементы проектной документации. Параллельно предполагается адаптация метода под различные СУП путем расширения поддерживаемой JSON-схемы, что обеспечит совместимость с разнообразными форматами импорта и повысит практическую применимость разработанного подхода в гетерогенных ИТ-средах.

В качестве дополнительного уровня обеспечения структурной целостности в перспективе предлагается развитие интерактивной постобработки. Данный подход предполагает возможность корректировки сгенерированных артефактов аналитиком в режиме, близком к реальному времени, с последующим автоматическим обновлением векторной базы $\mathcal{V}_{DB}$.

Заключение

В данной работе предложен метод автоматической трансформации функциональных требований ТЗ в объекты СУП на основе архитектуры RAG.

Научная новизна работы заключается в предложенном методе трансформации функциональных требований, совместно учитывающем структурное представление исходного документа, семантическую выборку релевантных требований и итеративную валидацию формируемой иерархии объектов СУП. Во-первых, адаптация архитектуры RAG с предварительной выборкой релевантных требований из векторной базы данных позволяет существенно снизить нагрузку на контекстное окно модели и повысить точность итоговой генерации. Во-вторых, новизна состоит во внедрении механизма строгой постобработки: генерируемые данные проходят проверку на соответствие заданной JSON-схеме с применением цикла итеративного исправления ошибок. Такое архитектурное решение обеспечивает высокую надежность метода, гарантируя получение 95 % валидных документов, пригодных для бесшовного импорта в СУП.

Разработанный метод позволяет сократить время переноса требований из ТЗ в СУП на 90 %, повысить качество импортируемых данных и обеспечить автоматическую генерацию отчетов о покрытии требований.

Модульная архитектура пайплайна (структурный анализ → семантическое извлечение → нормализация → RAG → валидация) позволяет адаптировать метод под различные предметные области путем замены онтологии и дообучения NER-модели.

СПИСОК ИСТОЧНИКОВ / REFERENCES

1. Wiegers K., Beatty J. *Software Requirements*. Redmond: Microsoft Press; 2013. 672 p.
2. Nikiforova O., Miľune M.K., Babris K., Pastor O. Generation of IT Project Documentation Elements from a Model Transformation Chain. In: *Proceedings of the 20th International Conference on Software Technologies: Volume 1, 10–12 June 2025, Bilbao, Spain*. SciTePress; 2025. P. 336–345. <https://doi.org/10.5220/0013568300003964>
3. Khurana D., Koli A., Khatter K., Singh S. Natural language processing: state of the art, current trends and challenges. *Multimedia Tools and Applications*. 2023;82(3):3713–3744. <https://doi.org/10.1007/s11042-022-13428-4>
4. Vaswani A., Shazeer N., Parmar N., et al. Attention is all you need. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, 04–09 December 2017, Long Beach, CA, USA*. 2017. P. 5998–6008.
5. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human*

- Language Technologies (NAACL-HLT 2019): Volume 1, 02–07 June 2019, Minneapolis, MN, USA*. ACL; 2019. P. 4171–4186.
6. Cleland-Huang J., Gotel O.C.Z., Hayes J.H., Mäder P., Zisman A. Software Traceability: Trends and Future Directions. In: *ICSE '14: Proceedings of the 36th International Conference on Software Engineering, 31 May – 07 June 2014, Hyderabad, India*. New York: ACM; 2014. P. 55–69. <https://doi.org/10.1145/2593882.2593891>
 7. Hou X., Zhao Y., Liu Y., et al. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Transactions on Software Engineering and Methodology*. 2024;33(8):220. <https://doi.org/10.1145/3695988>
 8. Lewis P., Perez E., Piktus A., et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems (NeurIPS 2020), 06–12 December 2020, Virtual Event*. 2020. URL: <https://arxiv.org/abs/2005.11401>
 9. Echin A., Kravets A.G., Safonova E., Skorobogatchenko D.A., Karasev D. Semantic Analysis of Technical Documentation: Systematic Review, Formal Task Definition, and Transformer-Based NER Implementation. *Big Data and Cognitive Computing*. 2026;10(7):199. <https://doi.org/10.3390/bdcc10070199>
 10. Коробкин Д.М. Извлечение физико-технической информации из текстовых документов. *Информационные технологии*. 2026;32(1):28–36. <https://doi.org/10.17587/it.32.28-36>
Korobkin D.M. Extraction of physical and technical information from text documents. *Informacionnye Tehnologii*. 2026;32(1):28–36. (In Russ.). <https://doi.org/10.17587/it.32.28-36>
 11. Ечин А.В., Алиева Н.Д., Садыков А.М., Кравец А.Г., Сафонова Е.В. Адаптивный шаблон технического задания: графовая модель, структурный анализ и алгоритм автоматизированной верификации. *Моделирование, оптимизация и информационные технологии*. 2026;14(4). <https://doi.org/10.26102/2310-6018/2026.55.4.017>
Echin A.V., Alieva N.D., Sadykov A.M., Kravets A.G., Safonova E.V. Adaptive technical specification template: graph-based model, structural analysis, and automated verification algorithm. *Modeling, Optimization and Information Technology*. 2026;14(4). (In Russ.). <https://doi.org/10.26102/2310-6018/2026.55.4.017>

ИНФОРМАЦИЯ ОБ АВТОРЕ / INFORMATION ABOUT THE AUTHOR

Ечин Александр Васильевич, аспирант, **Alexander V. Echin**, Postgraduate, Volgograd Волгоградский государственный технический State Technical University, Volgograd, the университет, Волгоград, Российская Федерация. Russian Federation.
e-mail: alexander_echin@mail.ru
ORCID: [0009-0006-1419-934X](https://orcid.org/0009-0006-1419-934X)

Статья поступила в редакцию 03.08.2026; одобрена после рецензирования 18.09.2026; принята к публикации 25.09.2026.

The article was submitted 03.08.2026; approved after reviewing 18.09.2026; accepted for publication 25.09.2026.