Труды Кольского научного центра РАН. Серия: Технические науки. 2026. Т. 17, № 1. С. 115–128.
Transactions of the Kola Science Centre of RAS. Series: Engineering Sciences. 2026. Vol. 17, No. 1. P. 115–128.

Научная статья
УДК 004.89
doi:10.37614/2949-1215.2026.17.1.012

АВТОМАТИЗИРОВАННАЯ ГИПЕРТЕКСТОВАЯ ДОКУМЕНТАЦИЯ УНАСЛЕДОВАННЫХ СИСТЕМ НА ОСНОВЕ СТАТИЧЕСКОГО АНАЛИЗА РЕПОЗИТОРИЕВ ПРОГРАММНОГО КОДА И ГЕНЕРАТИВНОГО ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Андрей Михайлович Федоров[1], Игорь Олегович Датьев[2], Михаил Олегович Илясов[3], Иван Геннадьевич Вишняков[4]
[1–4]Институт информатики и математического моделирования имени В. А. Путилова Кольского научного центра Российской академии наук, Апатиты, Россия
[1]a.fedorov@ksc.ru, https://orcid.org/0000-0002-2862-7994
[2]i.datyev@ksc.ru, https://orcid.org/0000-0002-8372-8704
[3]m.ilyasov@ksc.ru, https://orcid.org/0009-0006-9065-2396
[4]i.vishnyakov@ksc.ru, https://orcid.org/0009-0003-4938-5693

Аннотация

Автоматизация процессов разработки и сопровождения программного обеспечения является ключевым фактором повышения эффективности и скорости создания информационных систем. В условиях роста сложности проектов и требований к качеству документации особую актуальность приобретают инструменты, позволяющие автоматизировать анализ исходного кода и генерацию структурированных описаний. В статье представлено решение для автоматизированного документирования унаследованного кода, основанное на интеграции систем управления версиями и большой языковой модели. Разработанный конвейер производит последовательную обработку исходных модулей, извлекает метаданные программных сущностей (классов, методов, свойств), генерирует описания с помощью языковой модели при их отсутствии и автоматически публикует результат в формате вики-документации в среде коллективной разработки. Предложенный подход позволяет сократить трудозатраты на создание и поддержку документации, обеспечивает ее постоянную синхронизацию с кодом и способствует повышению качества проектной документации за счет интеллектуального дополнения. Решение предназначено для проектов на различных языках программирования, где необходимо автоматическое документирование унаследованного кода, интеграция описаний в системы коллективной разработки и обеспечение доступности документации через веб-интерфейс.

Ключевые слова:

автоматизация документирования, искусственный интеллект, большая языковая модель, генерирование описаний, статический анализ кода, поддержка унаследованных систем

Благодарности:

исследование выполнено в рамках государственного задания Института информатики и математического моделирования имени В. А. Путилова Кольского научного центра Российской академии наук от Министерства науки и высшего образования РФ, тема научно-исследовательской работы: «Методы и технологии создания интеллектуальных информационных систем для поддержки развития сложных динамических систем с региональной спецификой в условиях неопределенности и риска» (шифр темы FMEZ-2025-0053).

Для цитирования:

Автоматизированная гипертекстовая документация унаследованных систем на основе статического анализа репозиториев программного кода и генеративного искусственного интеллекта / А. М. Федоров [и др.] // Труды Кольского научного центра РАН. Серия: Технические науки. 2026. Т. 17, № 1. С. 115–128. doi:10.37614/2949-1215.2026.17.1.012.

Original article

AUTOMATED HYPERTEXT DOCUMENTATION OF LEGACY SYSTEMS BASED ON STATIC ANALYSIS OF PROGRAM CODE REPOSITORIES AND GENERATIVE ARTIFICIAL INTELLIGENCE

Andrey M. Fedorov[1], Igor O. Datyev[2], Mikhail O. Ilyasov[3], Ivan G. Vishnyakov[4]
[1–4]Putilov Institute for Informatics and Mathematical Modeling of the Kola Science Centre of the Russian Academy of Sciences, Apatity, Russia
[1]a.fedorov@ksc.ru, https://orcid.org/0000-0002-2862-7994
[2]i.datyev@ksc.ru, https://orcid.org/0000-0002-8372-8704
[3]m.ilyasov@ksc.ru, https://orcid.org/0009-0006-9065-2396
[4]i.vishnyakov@ksc.ru, https://orcid.org/0009-0003-4938-5693

© Федоров А. М., Датьев И. О., Илясов М. О., Вишняков И. Г., 2026

Abstract

Automating software development and maintenance processes is a key factor in increasing the efficiency and speed of information system creation. As projects become increasingly complex and documentation quality requirements increase, tools that automate source code analysis and the generation of structured descriptions are becoming increasingly important. This article presents a solution for automated documentation of legacy code based on the integration of version control systems and a large language model. The developed pipeline sequentially processes source modules, extracts metadata for software entities (classes, methods, properties), generates descriptions using large language model when they are missing, and automatically publishes the results as wiki documentation in a collaborative development environment. The proposed approach reduces the labor costs of creating and maintaining documentation, ensures its constant synchronization with the code, and improves the quality of project documentation through artificial intelligent descriptions. The solution is designed for projects in various programming languages that require automatic documentation of legacy code, integration of descriptions into collaborative development systems, and accessibility of documentation via a web interface.

Keywords:

documentation automation, artificial intelligence, large language model, description generation, static program analysis, legacy project support

Acknowledgments:

The study was carried out within the framework of the state assignment of the Putilov Institute for Informatics and Mathematical Modeling of the Kola Science Center of the Russian Academy of Sciences from the Ministry of Science and Higher Education of the Russian Federation, research topic: "Methods and technologies for creating intelligent information systems to support the development of complex dynamic systems with regional specificity in conditions of uncertainty and risk" (topic code FMEZ-2025-0053).

For citation:

Automated hypertext documentation of legacy systems based on static analysis of program code repositories and generative artificial intelligence / A. M. Fedorov [et al.] // Transactions of the Kola Science Centre of RAS. Series: Engineering Sciences. 2026. Vol. 17, No. 1. P. 115–128. doi:10.37614/2949-1215.2026.17.1.012.

Введение

Прогресс в сфере информационных технологий во многом основан на автоматизации различных процессов, ключевыми из которых являются разработка и поддержка информационных систем. С появлением больших языковых моделей подходы к этим процессам претерпели существенные изменения. Устаревание существующих информационных систем происходит тем быстрее, чем быстрее развиваются информационные технологии. Современные информационные системы необходимо поддерживать и развивать. Значительная часть используемых сегодня систем строится на комбинации уже функционирующих (унаследованных, устаревших) и новых модулей. Зачастую унаследованные программные модули не имеют должным образом оформленной документации, что приводит к сложностям поддержки и развития всей системы. Извлечение и обогащение данных на естественном языке из унаследованных систем приобрели еще большую актуальность в процессе научно-исследовательской работы коллектива авторов по теме «Методы и технологии создания интеллектуальных информационных систем для поддержки развития сложных динамических систем с региональной спецификой в условиях неопределенности и риска» [Информационная...]. В частности, развиваемое направление интеллектуализированного фокусированного сбора данных [Датьев и др., 2024; Федоров и др., 2025] потребовало углубленной концептуальной и практической проработки вопросов, вынесенных в заголовок настоящей статьи.

В данной работе решается задача разработки технологии автоматизированного анализа и документирования унаследованной кодовой базы с использованием элементов искусственного интеллекта для облегчения ее сопровождения и модернизации. Подход ориентирован на работу с любыми проектами, обладающими структурированным синтаксисом и поддерживаемыми инструментами автоматизированной документации.

Формально задача поставлена следующим образом: дана унаследованная кодовая база, представленная набором исходных файлов, часто с недостаточной или устаревшей документацией. Необходимо разработать метод и инструмент для автоматического преобразования данной кодовой базы в структурированную и связанную документацию в формате, пригодном для коллективного использования разработчиками (вики-система на базе Git).

В качестве интеллектуального компонента используются большие языковые модели, технические характеристики которых позволяют им анализировать исходный код,и на его основе генерировать краткие содержательные описания программных сущностей (модулей, классов, методов) на естественном языке.

© Федоров А. М., Датьев И. О., Илясов М. О., Вишняков И. Г., 2026

Автоматизация документирования унаследованных систем

Проблема документирования унаследованного кода является одной из наиболее актуальных в инженерии программного обеспечения. Сопровождение таких систем часто затруднено из-за устаревших, неполных или отсутствующих описаний, что ведет к увеличению временных затрат и риску внесения ошибок [Aghajani et al., 2019]. Автоматизация процесса создания документации рассматривается как ключевое направление для снижения этих издержек.

Традиционно для извлечения структуры исходного кода и генерирования документации используются инструменты статического анализа. Например, для экосистемы Delphi и Object Pascal одним из немногих специализированных решений является PasDoc. Этот инструмент анализирует исходные файлы, комментарии в специальном формате и создает документацию в различных форматах (HTML, LaTeX, SimpleXML) [PasDoc: Documentation...]. PasDoc служит основой для многих проектов по документированию, однако его возможности ограничены присутствующими в коде комментариями, и он не предназначен для семантического анализа или генерирования описаний на естественном языке. Среди альтернативных систем, используемых в других экосистемах, можно выделить Doxygen (для C++, Java и других языков), Javadoc (для Java) и Sphinx (для Python), которые также используют комментарии, встроенные в код [van Deursen, van Vliet, 2000].

В последние годы появились исследования и инструменты, которые интегрируют ИИ-компоненты в процесс документирования. Например, проект "CodeT5" [Wang et al., 2021] и "Codex" [Chen et al., 2021] демонстрируют возможности языковых моделей для генерирования комментариев к коду, но они фокусируются на уровне отдельных функций, а не на создании комплексной проектной документации. Коммерческие решения вроде Swimm [Swimm] предлагают синхронизацию документации с кодом, но требуют ручной настройки и интеграции в рабочий процесс разработки. Инструменты типа Mintlify [Mintlify] используют ИИ для автоматического создания README-файлов, однако они ограничиваются описанием репозитория в целом, не предоставляя детализированной документации по отдельным модулям и классам.

Существуют также научные работы по автоматическому документированию. Исследование [Wang et al., 2022] предлагает подход к генерированию документации с использованием нейросетей, но он требует предварительного обучения на специфичных датасетах и не обеспечивает интеграции с системами контроля версий. Работа [Chen et al., 2020] исследует автоматическое извлечение знаний из кода, но не фокусируется на создании связанной вики-документации. В отличие от этих решений, предложенное в данной работе решение предлагает комплексный подход, который сочетает статический анализ, интеллектуальное обогащение в виде генерирования аннотаций на основе локальных языковых моделей, автоматическое связывание сущностей и публикацию в вики-системах коллективной разработки.

Для организации коллективной работы над документацией широкое распространение получили вики-системы, интегрированные с платформами контроля версий. Например, Gitea — самостоятельная, легковесная платформа для хостинга Git-репозиториев, которая включает встроенный механизм вики, основанный на Git [Gitea]. Это позволяет хранить документацию в виде Markdown-файлов в отдельном репозитории, обеспечивая историю изменений, ветвление и возможность совместного редактирования. Идея интеграции автоматически генерируемой технической документации с подобными вики-системами не нова, однако существующие подходы, как правило, предполагают односторонний экспорт сгенерированных статических страниц, а не создание живой, связанной базы знаний [Ayer et al., 2016].

Параллельно развивается концепция персональных баз знаний (Personal Knowledge Management, PKM). Инструменты вроде Obsidian используют локальные хранилища Markdown-файлов с двунаправленными ссылками, что позволяет создавать сложные семантические сети из заметок [Obsidian]. Этот подход, доказавший свою эффективность для организации личных знаний, может быть адаптирован и для структурирования технической документации, где связи между модулями, классами и функциями имеют первостепенное значение.

В последние годы для решения проблемы недостаточной документации активно исследуется применение технологий искусственного интеллекта, в частности больших языковых моделей (Large Language Models, LLM). Исследования показывают, что LLM способны анализировать синтаксис и в определенной степени семантику исходного кода, генерируя релевантные текстовые описания

© Федоров А. М., Датьев И. О., Илясов М. О., Вишняков И. Г., 2026

функций, классов и их взаимодействий [Husain et al., 2019; Obsidian]. Применение искусственного интеллекта для дополнения существующей документации или ее создания «с нуля» рассматривается как прорывное направление, способное кардинально снизить трудозатраты [Li et al., 2019]. Ключевой задачей при этом становится интеграция подобного интеллектуального компонента в автоматизированные конвейеры сборки и развертывания.

Таким образом, несмотря на наличие отдельных инструментов для автодокументации (PasDoc), систем для совместной работы над документацией (Gitea Wiki) и перспективных исследований по применению ИИ, отсутствует целостное решение, которое интегрирует эти компоненты в единый, полностью автоматизированный конвейер, предназначенный специально для документирования унаследованных систем, обеспечивая не только извлечение структуры, но и интеллектуальное обогащение контента с последующей публикацией в формате связанной, поддерживаемой сообществом базы знаний. Разработке основных компонентов такого конвейера посвящена данная работа.

Архитектура системы автоматизированного документирования

Предлагаемое решение представляет собой последовательный конвейер, реализованный в виде bash-скрипта (ai-descr.sh). Архитектура системы (рис. 1) построена по модульному принципу и включает этапы: подготовки данных, статического анализа, интеллектуального обогащения, генерирования документации и публикации. Конвейер разработан для идемпотентной обработки большого количества файлов с минимальным потреблением ресурсов за счет последовательного, а не параллельного выполнения.

1. Подготовка окружения и извлечение исходного кода

Первоначальный этап обеспечивает изоляцию и актуальность обрабатываемых данных. Система создает уникальную временную рабочую директорию с меткой времени (TMP_GIT_DIR) для предотвращения конфликтов между запусками. Затем выполняются операции, соответствующие функциям check_dependencies и get_file_list:

```
# Создание уникальной временной директории
NOW=$(date +%FT%H=%M)
TMP_GIT_DIR=~/prog/tmp/pasdoc_git_tmp_${NOW}

# Проверка зависимостей
check_dependencies() {
  if ! command -v xmlstarlet >/dev/null 2>&1; then
    echo "Ошибка: xmlstarlet не установлен!"; exit 1
  fi
  # ... проверка других инструментов
}

# Получение списка файлов
get_file_list() {
  git clone "$repo_url" "$REPO_DIR" 2>/dev/null || cd "$REPO_DIR" && git pull
  git ls-tree -r --name-only HEAD | \
    grep -E '\.(pas|pp)$' | grep -E "^($dirs_pattern)/" | sort
}
```

Листинг 1. Подготовка окружения и парсинг исходного кода

— Проверяется наличие обязательных инструментов: xmlstarlet, git, а также опциональных curl, jq, base64 для работы с ИИ-компонентом.

— Происходит клонирование или обновление основного репозитория с исходным кодом ($GITEA_PRJ) и целевого вики-репозитория ($GITEA_WIKI). Это гарантирует работу с последней версией кода и документации.

© Федоров А. М., Датьев И. О., Илясов М. О., Вишняков И. Г., 2026

— Формируется полный список файлов для обработки с использованием команды git ls-tree, что обеспечивает учет только файлов, находящихся под версионным контролем. Список фильтруется по заданным директориям (source, Src) и расширениям (*.pas, *.pp).

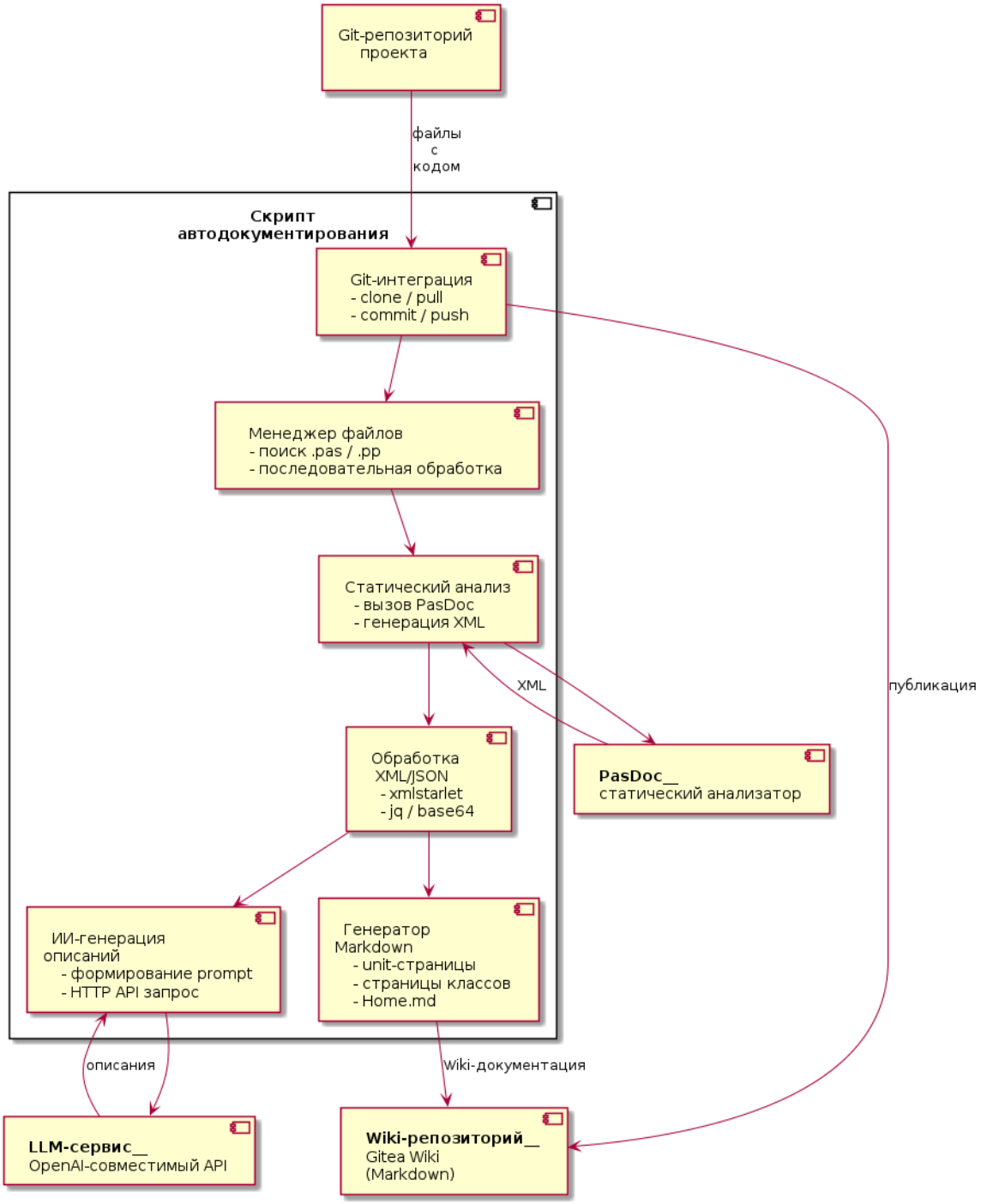


Рис. 1. Компоненты системы интеллектуального генерирования документации

Fig. 1. Components of the intelligent documentation generation system

© Федоров А. М., Датьев И. О., Илясов М. О., Вишняков И. Г., 2026

2. Статический анализ кода и извлечение метаданных

Ядро процесса анализа реализовано в функциях process_single_file и process_xml_file. Для каждого файла из списка выполняется:

```
# Обработка одного файла
process_single_file() {
  local pasdoc_list="$doc_dir/temp_${base_name}.inc"
  echo "$file_path" > "$pasdoc_list"

  # Генерирование XML через PasDoc
  $PASDOC_PROG --output "$xml_output_dir" \
    --format=simplexml \
    --source "$pasdoc_list" >/dev/null 2>&1

  # Извлечение описания модуля
  unit_desc=$(xmlstarlet sel -t -v "/unit/description/detailed" "$xml_file" 2>/dev/null | xargs)

  # Резервный поиск имени модуля
  [ -z "$unit_name" ] && unit_name=$(grep -i "^unit " "$file_path" | head -1 | sed 's/unit //' | sed
's/;.*//' | xargs)
                                                    }
```

Листинг 2. Статический анализ и извлечение метаданных

— Генерирование промежуточного XML-представления: инструмент PasDoc запускается в режиме simplexml для одного файла, что минимизирует потребление памяти. Команда имеет вид: $PASDOC_PROG --output "$xml_output_dir" --format=simplexml --source "$pasdoc_list". Полученный XML-файл содержит структурированные метаданные: имена модулей (unit), классы (structure[@type='class']), их методы (routine), свойства (property), переменные (variable) и связанные с ними комментарии.

— Парсинг и нормализация метаданных: для извлечения данных из XML используется утилита xmlstarlet. Например, извлечение описания модуля выполняется командой xmlstarlet sel -t -v "/unit/description/detailed" "$xml_file". Если имя модуля не удается определить из XML, проводится его поиск в исходном коде по ключевому слову unit. Этап обеспечивает переход от сырых данных к структурированному объектному представлению проекта.

3. Генерация описаний с помощью большой языковой модели

Данный этап, реализованный в функциях generate_ai_unit_description и generate_ai_class_description, является опциональным и активируется при наличии описанных инструментов (USE_AI=true). Его цель — восполнить пробелы в существующей документации.

— Вызов LLM: запрос к языковой модели отправляется только если описание сущности (модуля или класса) отсутствует, пусто или является шаблонным (равно "..."). Это реализует принцип точечного вмешательства для экономии вычислительных ресурсов.

— Формирование контекстного запроса: исходный код файла кодируется в base64 и включается в JSON-запрос к OpenAI-совместимому API ($AI_URL). Промпт содержит строгую инструкцию: «Дай краткое описание... один абзац, без форматирования», что направляет модель на генерирование лаконичных технических текстов. Модель по умолчанию — TeichAI/Qwen3-14B-Claude-Sonnet-4.5-Reasoning-Distill-GGUF:Q6_K.

— Идемпотентность: сгенерированное описание немедленно записывается обратно в промежуточный XML-файл с помощью xmlstarlet ed. Это делает процесс устойчивым к повторным запускам: при следующем выполнении скрипта ранее сгенерированные описания будут восприняты как существующие и повторный запрос к LLM не последует.

© Федоров А. М., Датьев И. О., Илясов М. О., Вишняков И. Г., 2026

```
# Универсальная функция для запроса к ИИ
generate_ai_description() {
   [ "$USE_AI" != true ] && return

   local file_content=$(base64 -w0 "$file_path")
   local json_request=$(jq -n --arg model "$AI_MODEL" \
      --arg content "$prompt$file_content" '{
         model: $model,
         messages: [{role: "user", content: $content}],
         temperature: 0.1,
         max_tokens: 250
      }')

   local response=$(curl -s -X POST "$AI_URL" \
      -H "Content-Type: application/json" -d "$json_request")

   echo "$response" | jq -r '.choices[0].message.content'
}

# Условный вызов LLM только при отсутствии описания
if [ -z "$unit_desc" ] || [ "$unit_desc" = "..." ]; then
   unit_desc=$(generate_ai_unit_description "$unit_name" "$file")
   [ -n "$unit_desc" ] && xmlstarlet ed -u "/unit/description/detailed" \
      -v "$unit_desc" "$xml_file" > "${xml_file}.tmp"
```

Листинг 3. Запрос к LLM для генерирования описаний

4. Генерирование и связывание вики-документации

На этом этапе структурированные и обогащенные метаданные преобразуются в набор связанных Markdown-файлов, формирующих вики-базу знаний. Логика распределена между функциями create_unit_markdown, create_class_markdown, add_properties и generate_markdown_section.

```
# Создание страницы модуля
create_unit_markdown() {
   cat > "$md_file" << EOF
# Unit: ${unit_name}
## Общее описание
${unit_desc}
---
## Структуры (классы)
EOF
}

# Создание страницы класса с перекрёстными ссылками
create_class_markdown() {
   cat > "$class_md" << EOF
# Класс: ${class_name}
**Наследник:** ${ancestor_name:-нет}
**Источник:** [[${unit_name}]]
## Описание
${class_desc}
EOF
}

# Добавление свойства к существующей странице
if ! grep -q "\[\[${unit_name}\]\]" "$class_md"; then
   echo -e "\n**Также встречается в:** [[${unit_name}]]" >> "$class_md"
fi
```

Листинг 4. Создание вики-документации

© Федоров А. М., Датьев И. О., Илясов М. О., Вишняков И. Г., 2026

— Создание страниц модулей (Units): для каждого модуля создается файл [UnitName].md. Страница включает техническое описание (исходное или сгенерированное ИИ), списки типов (//type) и глобальных переменных (//variable[not(parent::structure)]), извлеченные из XML, а также автоматически сформированный перечень всех классов модуля с гиперссылками.

— Создание страниц классов (Classes): для каждого класса создается отдельная страница [ClassName].md. Страница содержит описание, иерархию наследования, секции методов, полей и свойств, сформированные на основе XPath-запросов к XML. Важным элементом является перекрестная ссылка «Источник: [[UnitName]]», устанавливающая связь между классом и его модулем.

— Обеспечение целостности: при обработке файла, страница для которого уже существует, скрипт не перезаписывает ее полностью, а дополняет (например, добавляя ссылку на новый модуль, где встречается класс). Это позволяет накапливать знания о связях в проекте.

5. Публикация и поддержка актуальности вики-репозитория

Финальная стадия, управляемая функциями publish_single_file и create_index_file, обеспечивает интеграцию результата в процесс коллективной разработки.

```
# Инкрементальный коммит после обработки каждого файла
publish_single_file() {
   cd "$wiki_clone_dir"
   git add .
   if ! git diff --cached --quiet; then
      git commit -m "Добавлен/обновлён файл: $processed_file ($NOW)"
      git push origin main 2>/dev/null
   fi
}

# Создание навигационного индекса
create_index_file() {
   cat > "$wiki_dir/Home.md" << EOF
# Документация проекта
## Unit'ы (модули)
EOF
   ls *.md | grep -v Home.md | sort | while read file; do
      echo "- [[${file%.md}]]" >> Home.md
   done
}

# Очистка временных данных
rm -rf "$WIKI_CLONE_DIR" "$TMP_GIT_DIR"
```

Листинг 5. Публикация в вики-репозиторий

— Инкрементальные коммиты: после обработки каждого файла изменения в вики-директории ($WIKI_CLONE_DIR) сразу добавляются, коммитятся и пушатся в удаленный репозиторий. Такой подход минимизирует риск конфликтов и предоставляет разработчикам актуальную документацию в режиме, близком к реальному времени.

— Генерирование навигационного индекса: после обработки всех файлов автоматически создается или обновляется главная страница Home.md. Она содержит два структурированных списка с гиперссылками: все документированные модули и все классы, что добавляет в уже сформированную вики возможность для оперативного поиска и обзора.

— Очистка: по завершении работы все временные директории ($WIKI_CLONE_DIR, $TMP_GIT_DIR) рекурсивно удаляются.

Последовательность взаимодействия компонентов системы визуализирована на диаграмме последовательностей (рис. 2). Предложенная архитектура обеспечивает высокую степень автоматизации, сохраняя гибкость за счет параметризации (URL репозиториев, пути, модель LLM) и устойчивость за счет обработки ошибок и идемпотентности операций.

© Федоров А. М., Датьев И. О., Илясов М. О., Вишняков И. Г., 2026

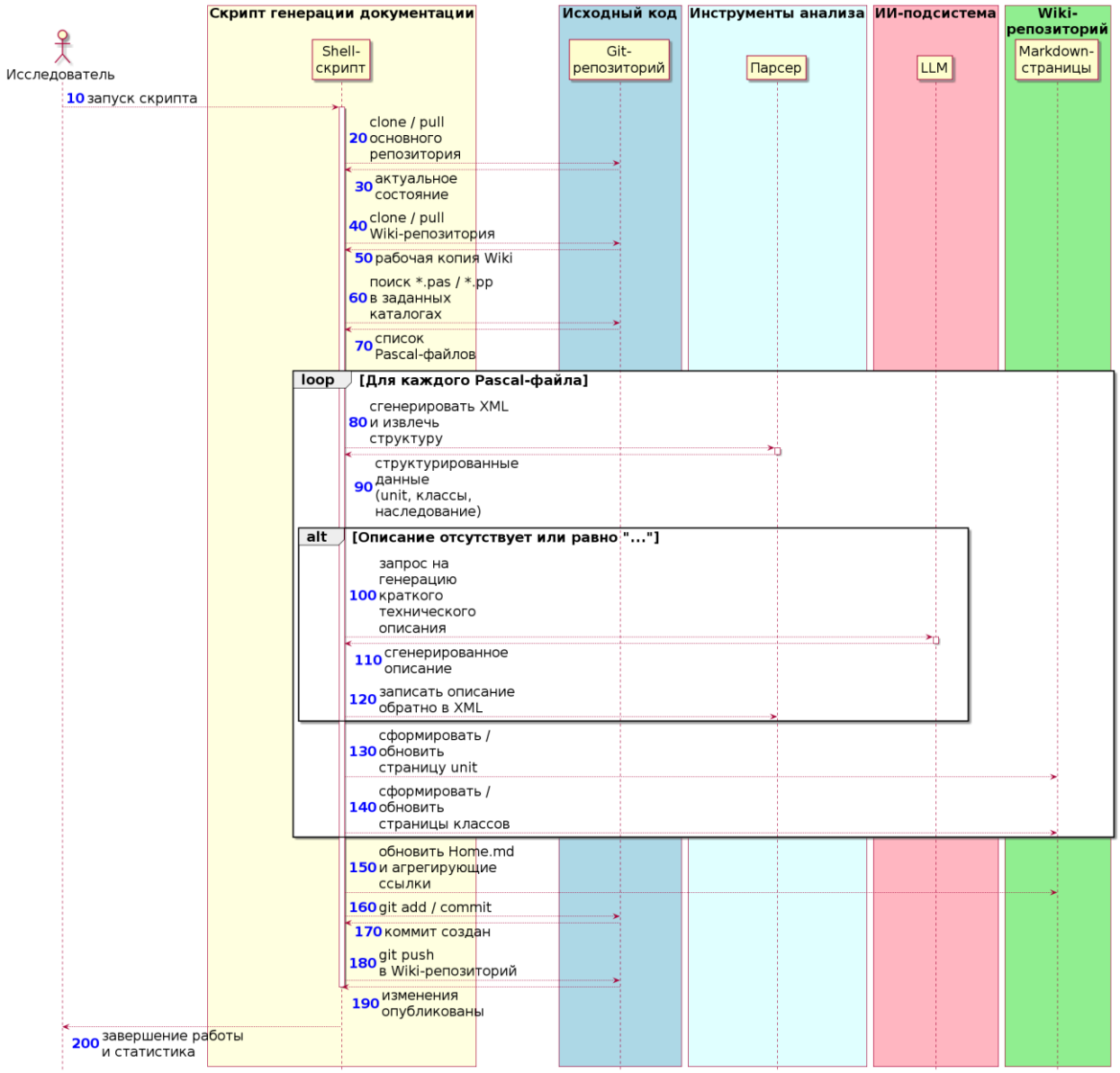


Рис. 2. Последовательность взаимодействия компонентов системы

Fig. 2. Sequence of interaction of system components

Анализ полученных результатов

Для оценки практической применимости и экономической эффективности разработанного конвейера было проведено экспериментальное тестирование на реальном унаследованном проекте «Dragon Logger», размещенном в репозитории Gitea. Целью эксперимента являлось сравнение временных и финансовых затрат при традиционном (ручном) документировании и при использовании автоматизированного конвейера.

Тестовая конфигурация и методика. Использовалась локально развернутая LLM TeichAI/Qwen3-14B-Claude-Sonnet-4.5-Reasoning-Distill-GGUF:Q6_K, запущенная через сервер с открытым API (Ollama). Обработка выполнялась разработанным скриптом (ai-descr.sh) с настройками по умолчанию:

© Федоров А. М., Датьев И. О., Илясов М. О., Вишняков И. Г., 2026

AI_TEMPERATURE=0, AI_MAX_TOKENS=40960. Весь процесс — от клонирования репозитория до публикации финальной вики — был выполнен одним запуском скрипта.

Основной результат работы системы — вики-документация проекта, состоящая из взаимосвязанных гипертекстовых страниц (с использованием разметки markdown). Всего было обработано 47 файлов исходного кода, содержащих 23 модуля (unit) и 38 классов. Система извлекла структурную информацию: иерархию наследования классов, списки методов, свойств и глобальных переменных.

Из 23 модулей только 7 содержали исходные описания в комментариях. Для остальных 16 модулей описания были сгенерированы языковой моделью. Аналогично: из 38 классов лишь 12 имели описания в коде, а для 26 классов текст был создан LLM. Таким образом, если значительная часть документации либо отсутствует, либо представлена шаблонными заглушками, то система эффективно восполняет эти пробелы.

Пример фрагмента извлеченных данных и сгенерированных описаний для модуля UConfig.pas и его методов в вики:

Класс: TConfig

Наследник: TObject

Видимость: public

Источник: [[UConfig]]

Описание

Класс TConfig управляет всеми настройками приложения через INI-файл. Содержит свойства для хранения параметров окон (MainScreen, InventoryScreen, AnalyzeScreen), настроек сортировки в различных представлениях данных, глобальных флагов (DevMode), учетных данных (UserName, Password) и обширных параметров кастомизации интерфейса (отображение элементов, фильтры анализа, поведение компонентов). Предоставляет методы для инициализации, чтения и записи настроек с поддержкой как полного обновления, так и сохранения только параметров анализатора.

Иерархия наследования

TConfig ← TObject

Методы

Initialize

Объявление: procedure Initialize();

Метод создает экземпляр TIniFile для работы с файлом конфигурации приложения. Инициализирует объект dataIni, используя путь к исполняемому файлу с расширением .ini. Вызывается при создании объекта для подготовки к операциям чтения/записи настроек.

Cleanup

Объявление: procedure Cleanup();

Освобождает ресурсы объекта dataIni, созданного в методе Initialize. Вызывается при завершении работы с объектом TConfig для корректного освобождения памяти и закрытия файла конфигурации.

ReadValues

Объявление: procedure ReadValues();

Загружает все настройки из INI-файла в свойства объекта. Читает параметры окон, глобальные настройки, учетные данные и параметры кастомизации, используя значения по умолчанию для отсутствующих записей. Обрабатывает различные типы данных (целые числа, строки, логические значения) и группирует настройки по секциям файла.

WriteValues

Объявление: procedure WriteValues(OnlyWriteAnalyzer: Boolean);

Видимость: public

Сохраняет значения свойств в INI-файл. При OnlyWriteAnalyzer=True записывает только настройки, связанные с окном анализатора (AnalyzeScreen) и его фильтрами. При OnlyWriteAnalyzer=False сохраняет все настройки: параметры окон, глобальные флаги, учетные данные и параметры кастомизации (за исключением некоторых временных или внутренних настроек, отмеченных как закомментированные).

Поля

Поля класса являются приватными и используются для хранения значений свойств.

© Федоров А. М., Датьев И. О., Илясов М. О., Вишняков И. Г., 2026

Свойства
MainScreen_Top
Тип: Integer
Видимость: public
Содержит верхнюю координату главного окна приложения.
MainScreen_Left
Тип: Integer
Видимость: public
Содержит левую координату главного окна приложения.
MainScreen_Width
Тип: Integer
Видимость: public
Определяет ширину главного окна приложения.
...
(еще 45 свойств для управления окнами, сортировкой, глобальными настройками, учетными данными и параметрами кастомизации интерфейса).

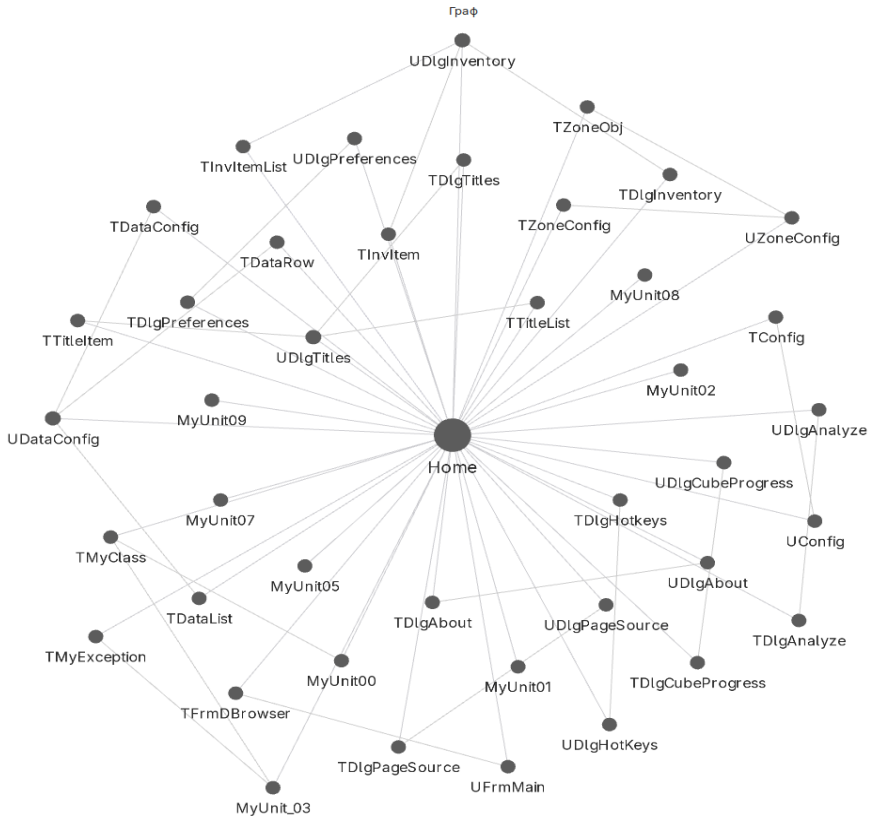


Рис. 3. Представление проекта в виде графов в базе знаний Obsidian

Fig. 3. Project representation as graphs in the Obsidian knowledge base

Автоматически созданные перекрестные ссылки между страницами образовали полноценную гипертекстовую базу знаний. Например, страница модуля UConfig.md содержит ссылку на класс "[[TConfig]]", а страница класса TConfig.md содержит обратную ссылку "Источник: [[UConfig]]". Аналогично: страница модуля MyUnit07.md содержит информацию о типе TMyEnumType и его элементах. На главной странице Home.md сформирован полный алфавитный указатель всех модулей и классов проекта с активными гиперссылками, включая ссылки на UConfig, MyUnit07 и TConfig.

© Федоров А. М., Датьев И. О., Илясов М. О., Вишняков И. Г., 2026

Это обеспечивает навигацию по всей структуре проекта — от общего списка модулей к конкретным страницам с описаниями классов, методов и свойств и обратно через ссылки "Источник".

Повторный запуск на том же репозитории подтвердил корректность реализации принципа идемпотентности: для файлов, уже обработанных в предыдущем запуске, повторных запросов к большой языковой модели не происходило. Система корректно определяла существующие описания и пропускала эти сущности. Обработка каждого файла занимала в среднем 12–15 секунд (включая время генерирования XML и потенциальный запрос к LLM), что для всего проекта составило около 10 минут.

Все сгенерированные Markdown-файлы были успешно опубликованы в вики-репозиторий проекта. Инкрементальные коммиты после обработки каждого файла обеспечили детальную историю изменений. Полученная вики полностью функциональна: поддерживает навигацию по ссылкам, отображает форматирование и доступна через веб-интерфейс Gitea для всей команды разработки.

Гипертекстовая структура автоматически сгенерированной документации не только обеспечивает удобство навигации через веб-интерфейс Gitea Wiki, но и открывает дополнительные возможности при локальной работе с файлами. Поскольку документация сохраняется в виде стандартных Markdown-файлов с двунаправленными ссылками, ее можно напрямую открыть в инструментах для управления персональными базами знаний, таких как Obsidian. В Obsidian эти ссылки автоматически строят визуальный граф зависимостей, наглядно отображающий архитектурные связи между модулями и классами проекта. Разработчик видит не просто список файлов, а карту взаимосвязей: какие модули содержат какие классы, как типы используются в разных частях системы.

Заключение

В ходе выполнения исследования разработана и апробирована автоматизированная система документирования унаследованных программных проектов, интегрирующая методы статического анализа исходного кода и генеративные возможности больших языковых моделей. Предложенное решение реализует полный конвейер обработки — от извлечения метаданных программных сущностей до публикации структурированной гипертекстовой документации в вики-репозиторий.

Принципиальной особенностью предложенного подхода является не просто автоматическое создание технической документации, а формирование связанной гипертекстовой структуры. Двунаправленные перекрестные ссылки между страницами модулей и классов образуют полноценную базу знаний, пригодную как для навигации через стандартный веб-интерфейс, так и для визуального анализа архитектурных связей в инструментах персональных баз знаний. Это качественно меняет восприятие унаследованного кода, превращая разрозненные исходные файлы в исследовательское пространство, где разработчик может отслеживать зависимости и прослеживать логику взаимодействия компонентов.

Несмотря на возможность оперативного создания документации, в текущей реализации генерирование описаний осуществляется на основе контента одного обрабатываемого файла без учета семантического контекста связанных модулей, классов или общей архитектуры проекта. Данное ограничение приводит к тому, что полученные описания могут носить локальный характер и не всегда отражать роль сущности в масштабе всей системы.

В связи с этим в качестве дальнейшего развития предполагается интеграция механизмов генерации с дополненным извлечением (RAG, Retrieval-Augmented Generation). Предполагается создание индексного хранилища векторных представлений всех документированных программных сущностей и их метаданных. При обработке нового модуля или класса система сможет извлекать релевантный контекст: описания родительских классов, интерфейсов, типов данных, а также примеры использования анализируемой сущности в смежных модулях.

Таким образом, предложенная система автоматизированного документирования на основе статического анализа и генеративного искусственного интеллекта представляет собой действенный инструмент снижения порога входа в унаследованные проекты и поддержания актуальности проектной документации. Дальнейшая эволюция системы в направлении контекстно-осведомленной генерации с применением RAG-архитектуры позволит перейти от автоматического документирования к интеллектуальному моделированию предметной области программной системы с формированием полноценного семантического описания ее структуры и поведения.

© Федоров А. М., Датьев И. О., Илясов М. О., Вишняков И. Г., 2026

Список источников

Датьев И. О., Федоров А. М., Ревякин А. А. Фокусированный сбор и обработка открытых данных социальных медиа // Онтология проектирования. 2024. Т. 14, № 4 (54). С. 569–581. DOI: 10.18287/2223-9537-2024-14-4-569-581.

Информационная карта НИОКТР. URL: https://gisnauka.ru/nioktr/detail/3A4F1NQ3Q3O68L885FV6A9XO (дата обращения: 20.01.2026).

Федоров А. М., Датьев И. О., Вишняков И. Г. Метод интеграции больших языковых моделей в алгоритмы фокусированного мониторинга открытых данных социальных медиа // Информатика и автоматизация. 2025. № 6 (24). С. 1623–1648.

Automatic Documentation Generation via Source Code Summarization / Y. Wang [et al.] // IEEE Transactions on Software Engineering. 2022. Vol. 48, № 5. P. 1528–1543.

Ayer S. K., Messner J. I., Anumba C. J. A Framework for Automating the Creation of a Building Information Model // Proceedings of the 16th International Conference on Computing in Civil and Building Engineering (ICCCBE 2016). Osaka, Japan, 2016. P. 1831–1838.

Chen M., Tworek J., Jun H., Yuan Q., Pinto H. P. D. O., Kaplan J. [et al.]. Evaluating Large Language Models Trained on Code [Электронный ресурс]: препринт. 2021. arXiv:2107.03374. URL: https://arxiv.org/abs/2107.03374 (дата обращения: 20.10.2025).

Chen Z., Kommrusch S., Monperrus M. Neural transfer learning for repairing security vulnerabilities in C code // IEEE Transactions on Software Engineering. 2020. Vol. 47, № 11. P. 2589–2610.

CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation / Y. Wang [et al.] // Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2021. P. 8696–8708.

Gitea: A painless self-hosted Git service [Электронный ресурс]: официальная документация. URL: https://docs.gitea.com (дата обращения: 20.10.2025).

Husain H., Wu H., Gazit T., Allamanis M., Brockschmidt M. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search [Электронный ресурс]: препринт. 2019. arXiv:1909.09436. URL: https://arxiv.org/abs/1909.09436 (дата обращения: 20.10.2025).

Li Y., Choi D., Chung J. [et al.]. Competition-Level Code Generation with AlphaCode / Y. Li [et al.] // Science. 2022. Vol. 378, № 6624. P. 1092–1097.

Mintlify—AI powered documentation [Электронный ресурс]: официальный сайт. URL: https://mintlify.com/ (дата обращения: 20.10.2025).

Obsidian [Электронный ресурс]: официальная документация. URL: https://help.obsidian.md (дата обращения: 13.02.2026).

PasDoc: Documentation Tool for Pascal Source Code [Электронный ресурс]: официальный репозиторий проекта. URL: https://github.com/pasdoc/pasdoc (дата обращения: 20.10.2025).

Software Documentation Issues Unveiled / E. Aghajani [et al.] // Proceedings of the 41st International Conference on Software Engineering (ICSE 2019). Montreal, Canada : IEEE/ACM, 2019. P. 1199–1210. DOI: 10.1109/ICSE.2019.00122.

Swimm—Continuous documentation that flows with your work [Электронный ресурс]: официальный сайт. URL: https://swimm.io/ (дата обращения: 20.10.2025).

van Deursen A., van Vliet H. Documenting and Analyzing a Legacy System // Journal of Software Maintenance and Evolution: Research and Practice. 2000. Vol. 12, № 3. P. 143–172.

References

Aghajani E., Nagy C., Vega-Márquez O. L., Linares-Vásquez M., Moreno L., Bavota G., Lanza M. Software Documentation Issues Unveiled. *Proceedings of the 41st IEEE/ACM International Conference on Software Engineering (ICSE 2019)*, Montreal, Canada, 2019, pp. 1199–1210. DOI: 10.1109/ICSE.2019.00122.

Ayer S. K., Messner J. I., Anumba C. J. A Framework for Automating the Creation of a Building Information Model. *Proceedings of the 16th International Conference on Computing in Civil and Building Engineering (ICCCBE 2016)*, Osaka, Japan, 2016, pp. 1831–1838.

Chen M., Tworek J., Jun H., Yuan Q., Pinto H. P. D. O., Kaplan J., et al. Evaluating Large Language Models Trained on Code. 2021. arXiv:2107.03374. Available at: https://arxiv.org/abs/2107.03374 (accessed 20.10.2025).

Chen Z., Kommrusch S., Monperrus M. Neural transfer learning for repairing security vulnerabilities in C code. *IEEE Transactions on Software Engineering*, 2020, Vol. 47, No. 11, pp. 2589–2610.

Datyev I. O., Fedorov A. M., Revyakin A. A. Focused collection and processing of open social media data. *Ontology of Designing*, 2024, vol. 14, no. 4 (54), pp. 569–581. (In Russ.). doi:10.18287/2223-9537-2024-14-4-569-581.

© Федоров А. М., Датьев И. О., Илясов М. О., Вишняков И. Г., 2026

Fedorov A. M., Datyev I. O., Vishnyakov I. G. The Method for Integrating Large Language Models into Algorithms for Focused Monitoring of Open Social Media Data. *Informatics and Automation*, 2025, no. 6 (24), pp. 1623–1648. (In Russ.).

Gitea: A painless self-hosted Git service. Official documentation. Available at: https://docs.gitea.com (accessed 20.10.2025).

Husain H., Wu H., Gazit T., Allamanis M., Brockschmidt M. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. 2019. arXiv:1909.09436. Available at: https://arxiv.org/abs/1909.09436 (accessed 20.10.2025).

Li Y., Choi D., Chung J., Kushman N. Competition-Level Code Generation with AlphaCode. *Science*, 2022, Vol. 378, No. 6624, pp. 1092–1097.

Mintlify—AI powered documentation. Official website. Available at: https://mintlify.com/ (accessed 20.10.2025).

Obsidian. Official documentation. Available at: https://help.obsidian.md (accessed 13.02.2026).

PasDoc: Documentation Tool for Pascal Source Code. Official repository. Available at: https://github.com/pasdoc/pasdoc (accessed 20.10.2025).

R&D Information Card. Available at: https://gisnauka.ru/nioktr/detail/3A4F1NQ3Q3O68L885FV6A9XO (accessed 20.10.2025).

Swimm—Continuous documentation that flows with your work. Official website. Available at: https://swimm.io/ (accessed 20.10.2025).

van Deursen A., van Vliet H. Documenting and Analyzing a Legacy System. *Journal of Software Maintenance and Evolution: Research and Practice*, 2000, Vol. 12, No. 3, pp. 143–172.

Wang Y., Liu S., Tan L., Zhou Y. Automatic Documentation Generation via Source Code Summarization. *IEEE Transactions on Software Engineering*, 2022, Vol. 48, No. 5, pp. 1528–1543.

Wang Y., Wang W., Joty S., Hoi S. C. H. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021, pp. 8696–8708.

Информация об авторах

А. М. Федоров — кандидат технических наук, ведущий научный сотрудник;
И. О. Датьев — кандидат технических наук, старший научный сотрудник;
М. О. Илясов — программист;
И. Г. Вишняков — аспирант ФИЦ КНЦ РАН, системный администратор ИИММ КНЦ РАН.

Information about the authors

A. M. Fedorov — PhD (Engineering), Leading Researcher;
I. O. Datyev — PhD (Engineering), Senior Researcher;
M. O. Ilyasov — Programmer;
I. G. Vishnyakov — Postgraduate Student of the Federal Research Center of the Kola Science Center of the Russian Academy of Sciences, System Administrator of the IIMM KSC RAS.

Статья поступила в редакцию 10.02.2026; одобрена после рецензирования 17.02.2026; принята к публикации 19.02.2026.
The article was submitted 10.02.2026; approved after reviewing 17.02.2026; accepted for publication 19.02.2026.

© Федоров А. М., Датьев И. О., Илясов М. О., Вишняков И. Г., 2026