

ПОДХОДЫ К ДОКУМЕНТИРОВАНИЮ В МАСШТАБНЫХ ЦИФРОВЫХ СИСТЕМАХ

Быков С.А., Дробкова О.С.

Московский государственный технический университет им. Н.Э. Баумана, Москва, Россия
sabkvq@yandex.ru, drobkova.os@bmstu.ru

Аннотация. С ростом сложности цифровых и мультидисциплинарных систем поддержание актуальной документации, прозрачности и согласованности архитектурных решений становится критически важным. Исследование охватывает современные цифровые подходы к документированию.

Ключевые слова: документация как код, docs-as-code, архитектура как код, architecture-as-code, автоматизация документирования, непрерывная документация, DocOps, ArchOps.

Введение

Современные ИТ-системы становятся все более сложными: микросервисная архитектура, облачные инфраструктуры и DevOps требуют постоянного обновления и поддержки технической документации [1]. Традиционные подходы становятся экономически неэффективными: возрастают затраты на поддержку, увеличивается риск ошибок. Одной из ключевых проблем является «проклятие знания» – ситуация, когда опытные сотрудники ошибочно предполагают, что знания очевидны для других, что затрудняет передачу информации и увеличивает стоимость адаптации новых участников [2]. Особенно остро эта проблема проявляется в крупных мультикомандных проектах, где рассогласование между кодом и документацией ведет к сбоям в коммуникации, увеличивает технический долг и снижает скорость вывода продукта на рынок. Кроме того, передача знаний новым участникам проекта без актуальной документации требует дополнительных ресурсов, удлинняя цикл адаптации. Так, автоматизация процессов документирования приобретает не только технологическую, но и стратегическую значимость, влияя на эффективность команд и стоимость сопровождения цифровых продуктов. Современные подходы интегрируют документирование в общий жизненный цикл разработки, снижая транзакционные издержки, повышая прозрачность процессов и обеспечивая воспроизводимость знаний. Таким образом, автоматизация документации становится неотъемлемой частью устойчивого управления цифровыми активами, что подтверждает ее актуальность как научной [3], так и прикладной проблемы в экономике цифровых систем [4-7].

Целью данного исследования является анализ и систематизация современных подходов к автоматизации документирования в масштабных цифровых системах, а также выявление их экономических преимуществ для повышения эффективности процессов разработки и сопровождения программного обеспечения. Для достижения поставленной цели были поставлены следующие задачи:

1. Провести обзор и анализ отечественных и зарубежных научных и прикладных источников по тематике автоматизации технической документации в цифровых системах.
2. Систематизировать основные методологические подходы к автоматизации документирования.
3. Оценить потенциальные экономические эффекты внедрения автоматизированных подходов к документированию в масштабных цифровых проектах.
4. Обобщить результаты анализа и представить выводы.

1. Методология

В работе использован метод «снежного кома», который позволил поэтапно выявлять ключевые концепции и инструменты для автоматизации документирования. Были изучены научные публикации, отраслевые и технические руководства, open-source-проекты. В фокусе оказались концепции: docs-as-code, diagram-as-code, architecture-as-code, а также DocOps и ArchOps как практики интеграции документации в жизненный цикл разработки.

2. Результаты

В результате изучения имеющихся данных по теме, были выделены подходы к документации, которые могут быть объединены с помощью docs-as-code.

2.1. Docs-as-code

Философия Documentation-as-Code (DaC) представляет собой трансформационный методологический сдвиг в области программной инженерии и технической коммуникации. Этот подход предполагает управление технической документацией с той же строгостью, инструментами и

рабочими процессами, которые применяются к исходному коду программного обеспечения [8]. Это позволяет сократить издержки за счет автоматизации публикации и проверки, устранить ручной труд в управлении версиями, повысить прозрачность за счет включения документации в цикл code-review (процесс проверки чужого кода членом команды или с использованием автоматизированных инструментов). Экономический эффект в данном случае достигается за счет снижения затрат на сопровождение, минимизации простоев и повышения качества пользовательской и технической документации.

Данный подход возник в ответ на постоянные проблемы устаревшего контента и ненадежной информации, присущие традиционным методам документации. Применение DaC особенно распространено в системах Content Operations (ContentOps), предназначенных исключительно для текстовых форматов, где он предоставляет основную инфраструктуру для управления жизненным циклом контента.

Суть DaC заключается в совместном размещении документационных артефактов с исходным кодом в экосистемах контроля версий, преимущественно на платформах, основанных на Git. Эта парадигма возникла из синергетической интеграции практик Continuous Integration (CI) и легковесных языков разметки, таких как Markdown, что позволяет разработчикам создавать документацию в привычных интегрированных средах разработки (IDEs). Основная техническая реализация следует автоматизированной последовательности: коммиты в системе контроля версий инициируют совместный обзор через pull requests, что запускает пайплайны, преобразующие исходники разметки в публикуемые форматы с помощью статических генераторов сайтов, выполняя проверки на соответствие (например, целостность ссылок, верификация кода) и развертывая результаты с минимальным ручным вмешательством. Этот рабочий процесс напрямую решает две системные проблемы: отсутствие авторинга, ориентированных на разработчиков, и механизмов обеспечения соответствия документации [9, 10].

Хотя DaC акцентирует внимание на интеграции инструментов – использовании систем контроля версий, трекеров задач и CI/CD параллельно с разработкой кода – его ограничения в захвате всей сложности документации способствовали концептуальной эволюции. Появляющаяся перспектива "docs-as-ecosystem" рассматривает документацию не просто как файлы Markdown или фрагменты кода, а как целостную экосистему, требующую централизованного управления сообществом. Эта структура признает многомерный характер документации, охватывающий разнообразные артефакты, сети участников, процессы рецензирования и механизмы доставки, которые выходят за рамки технических инструментов. Она соответствует отраслевым тенденциям, направленным на развитие совместных знаний, где общее владение позволяет различным участникам – разработчикам, авторам, экспертам в области – совместно использовать специализированные навыки [11].

Культурный аспект остается важным в обеих структурах. DaC намеренно создает среды, в которых авторы и разработчики чувствуют свою ответственность за документацию и работают вместе, чтобы сделать ее как можно лучше. Эта совместная этика трансформирует документацию из изолированной задачи в интегрированную ответственность за продукт, что подкрепляется общими рабочими процессами и прозрачностью контроля версий [12]. Взгляд на экосистему дополнительно расширяет этот принцип, подчеркивая необходимость поддержки сообщества и каналов кросс-функционального вклада, которые поддерживают жизнеспособность документации.

Тем не менее, остаются проблемы реализации, особенно в отношении требований к технической квалификации для участников, не являющихся разработчиками, и постоянного обслуживания пайплайнов [13]. По мере того, как терминология эволюционирует от "кода" к "экосистеме", область продолжает примирять инженерную строгость с человеческими и системными аспектами устойчивого создания знаний, продвигаясь к более адаптивным и инклюзивным структурам для технической коммуникации. Также следует отметить, что хоть сближение процессов разработки и документации приводит к ускорению вывода продукта на рынок, но также требует изменения процессов CI/CD и DevOps-практик. Требуется переобучение технических писателей, адаптация документационных пайплайнов, затраты на поддержание редакторов и инструментов (например, Markdown-совместимых систем, Git, генераторов документации).

Основные языки и инструменты docs-as-code представлены в таблицах 1 и 2 соответственно.

Таблица 1. Сравнение форматов docs-as-code

Формат	Год введения	Особенности	Сценарии использования
TeX	1978	Продвинутая верстка, математическая нотация, точный контроль	Академические статьи, техническое издательство, книги
Wiki	1994	Легковесный, совместный, ориентированный на ссылки	Wikipedia, базы знаний, внутренние вики
XML	1998	Расширяемый, строгая вложенность, ориентирован на данные	Веб-сервисы, конфигурационные файлы, хранение данных
reStructuredText	2001	Мощный, расширяемый, строгий синтаксис	Документация Python, Sphinx, ядро Linux
AsciiDoc	2002	Сложные структуры, экспорт в несколько форматов	Документация Git, Red Hat, O'Reilly Media
Markdown	2004	Простой синтаксис, широкая поддержка, ограниченное форматирование	Блоги, GitHub, документация проектов

Источник: составлено автором

Таблица 2. Сравнение инструментов docs-as-code

Инструмент	Назначение	Преимущества	Ограничения
Asciidoctor	Генерация документации из AsciiDoc-файлов	Поддержка PDF/HTML, мощная разметка, расширяемость	Требует знания синтаксиса AsciiDoc
Pandoc	Конвертация между форматами	Поддержка десятков форматов, мощные фильтры и шаблоны	Настройка шаблонов может быть сложной
Doxygen	Генерация API-документации из кода	Автоматическое извлечение, поддержка C++, Java, Python и др.	Ограниченные стилистические настройки
Swagger (OpenAPI)	Документация RESTful API	Интерактивность, поддержка автогенерации, UI	Требует строгого соблюдения OpenAPI спецификации
Sphinx	Статическая генерация из reStructuredText	Поддержка расширений, интеграция с Read the Docs	Использует специфичный синтаксис
Jekyll	Статический сайт из Markdown	Простота, поддержка GitHub Pages	Медленная пересборка на больших проектах
Hugo	Быстрый генератор сайтов на Go	Мгновенная сборка, гибкость шаблонов	Язык шаблонов специфичен, кривая обучения
MkDocs	Статический сайт из Markdown	Простая настройка, поддержка тем	Менее гибкий для кастомной логики
Confluence	Корпоративный wiki-инструмент	Интеграция с Jira, визуальный редактор	Проприетарный, слабая интеграция с Git
LaTeX	Профессиональная вёрстка документов	Полный контроль над оформлением, математическая нотация	Сложность синтаксиса, требует компиляции
Diplodoc	Внутренняя разработка Яндекса для API-документации	Актуальная структура, используется в больших проектах	Недоступен как open source (на момент анализа)
Docusaurus	React-движок для документации	Поддержка версионирования, Markdown + JSX, интеграция с GitHub	Требует знания JavaScript/React

Источник: составлено автором

Следует подчеркнуть, что Pandoc демонстрирует наибольшую универсальность, выделяющую его среди инструментов docs-as-code. Его архитектура, основанная на преобразовании абстрактного синтаксического дерева (AST), обеспечивает поддержку свыше 40 форматов ввода-вывода, включая LaTeX, HTML, Markdown, DOCX и PDF, что позволяет применять его в мультидисциплинарных исследовательских и инженерных контекстах. Интеграция с системами сборки, генераторами статических сайтов (Jekyll, Sphinx) и pipelines CI/CD обуславливает его роль как инфраструктурного компонента, связывающего этапы создания, обработки и публикации контента.

Docs-as-Code снижает транзакционные издержки на взаимодействие между разработчиками и техписателями. Внедрение этого подхода позволяет исключить традиционную фазовую модель создания документации, где участие специалистов по документированию начинается уже после завершения основной разработки. Это приводит к снижению временных и финансовых затрат на актуализацию устаревших материалов. Более того, организация совместной работы в едином репозитории обеспечивает прозрачность версий, что минимизирует риски дублирования и ошибок, особенно при масштабировании команды. Среди крупных технологических компаний (Google, Amazon, GitLab) использование Docs-as-Code стало стандартом де-факто. Особенно активно он используется в проектах с открытым исходным кодом, где прозрачность и вовлеченность сообщества являются ключевыми факторами успеха. Развитие инструментов, таких как MkDocs, Docusaurus и Hugo, способствует распространению подхода среди малого и среднего бизнеса.

Инновационность данного подхода заключается в слиянии процессов разработки и создания знаний в одном канале. Docs-as-Code позволяет выстраивать динамические контуры знаний, которые обновляются синхронно с продуктом. Это открывает перспективы для внедрения систем управления знаниями на основе искусственного интеллекта, где документация становится обучающей средой для машинного анализа и последующей генерации решений.

2.2. Literal programming

В рамках парадигмы docs-as-code, предполагающей создание артефактов, интерпретируемых как человеком, так и машинами, концепция Literate Programming (LP), предложенная Дональдом Кнудом в 1984, представляет фундамент для других подходов к цифровому документированию [14]. LP использует естественно-языковые для объяснения логики программы и её структуры, трансформируя программный код в нарратив. Отнесение к Literate programming инструментов, генерирующих документацию из кода с комментариями, противоречиво у разных исследователей [9, 15].

Literate Programming – это подход, при котором разработка программного обеспечения происходит в парадигме «человек читает, машина исполняет». Код пишется как объяснительный текст, в который встроены исполняемые фрагменты. Главная экономическая ценность данного подхода заключается в сокращении затрат на понимание и сопровождение сложных участков кода. Особенно важно это для отраслей с высокой стоимостью ошибок – банковский сектор, телекоммуникации, авиация. Качественно оформленный и объясненный код снижает стоимость ревью, позволяет существенно сократить фазы тестирования и быстрее адаптировать новых сотрудников. Таким образом, происходит экономическая капитализация знаний: знания не теряются с уходом разработчиков, а сохраняются в продуктах. Literate Programming, в свою очередь, стимулирует создание самообучающихся и самообъясняющихся систем. В контексте развития ИИ и машинного обучения, где объяснимость моделей критична, этот подход позволяет объединить результат и логику его получения в едином артефакте. Это обеспечивает прорывные возможности для аудита ИИ-систем, сертификации решений и взаимодействия с регуляторами.

Основные принципы literal programming могут сформулированы следующим образом:

- Совмещение кода и документации в одном артефакте (не путать с генерацией доков из комментариев)
- Логика повествования следует за мыслью программиста, а не требованиями компилятора
- Программирование как акт осмысления – явная формулировка идей перед реализацией.

Сам термин literate programming наиболее популярен в контексте методологии научных вычислений, нежели в промышленном программировании. Более того, создатель концепции LP, Дональд Кнут, является автором системы верстки TeX, которая используется в основном для научного письма [15]. Развитие концепции Literate Programming привело к появлению интерактивных блокнотов, отличительной чертой которых является воспроизводимость, а также появлению подхода [16].

Для интерактивных блокнотов характерны форматы разметки, используемые в docs-as-code. Сравнение наиболее популярных инструментов подхода представлено в таблице 3.

Таблица 3. Сравнение интерактивных блокнотов

Параметр	Quarto	RMarkdown	Jupyter Notebook	Apache Zeppelin
Поддерживаемые языки программирования	R, SQL, Julia, Python. Можно использовать kernels Jupyter Notebook	R, Python, SQL, Bash, Rcpp, Stan, JavaScript	Python, R, Julia, and Scala. Множество сторонних kernels	Apache Spark, Apache Flink, Python, R, JDBC, Markdown, Shell. Сторонние интерпретаторы
Поддерживаемые форматы разметки	Markdown, LaTeX, HTML	Markdown, LaTeX, HTML	Markdown, LaTeX, HTML	Markdown, LaTeX, HTML
Интеграция с Git	Да	Да	Усложненная	Нет
Встроенная автоматизация	Да	Да	Да	Усложненная
Генератор сайта	MkDocs/Hugo	bookdown/blogdown	Jupyter Book/Sphinx	Jekyll
Тестирование кода	Да	Да	Да	Отсутствует
Модульность	Включение файлов, параметры	Дочерние документы, параметры	"Магические команды"	Отсутствует
Управление зависимостями	Да	Да	Да	Зависит от интерпретатора
Интерактивные визуальные элементы	ObservableJS/Shiny	Shiny/htmlwidgets	ipywidgets/Voilà	Динамические формы и дашборды

Источник: составлено автором в ходе исследования

Из таблицы 3 видно, что многие интерактивные блокноты используют форматы разметки и средства генерации сайтов, поддерживаемые docs-as-code, что снова показывает близость этих подходов. Не все инструменты имеют функции, которые свойственны инструментам docs-as-code. С точки зрения возможностей, лучшим выбором может быть Quarto, который использует движки knitr (на котором построен RMarkdown) и Jupyter [17].

Интеграция literate programming позволяет повысить качество архитектурного мышления, особенно в научных и исследовательских проектах, улучшить прослеживаемость решений. При этом данный подход характеризуется рядом вызовов: высокие временные инвестиции на подготовку документации, возможна перегрузка при применении в масштабных проектах, что повлияет на скорость принятия решений, низкая совместимость с Agile и CI/CD-подходами, требуется высокий уровень дисциплины от разработчиков и специфические знания, которые редко применяются в индустрии программирования.

2.3. DocOps и Archops

DocOps и ArchOps представляют собой ключевые практики, которые интегрируют процессы документации и архитектуры в жизненный цикл программного обеспечения. DocOps (иногда именуется DevDocOps), или операции с документацией, фокусируется на автоматизации создания, управления и публикации документации, применяя принципы DevOps [18]. Это включает использование CI/CD для автоматизации сборки и развертывания документации, а также концепцию "Docs-as-Code", где документация создается в форматах, привычных разработчикам, и хранится в системах контроля версий [9, 13]. ArchOps, в свою очередь, расширяет DevOps на уровень проектирования системы, акцентируя внимание на управлении архитектурой как важным элементом на всех этапах жизненного цикла. Это включает интеграцию архитектурной верификации в CI/CD, использование моделей архитектуры в качестве исходных точек для развертывания и автоматическую генерацию артефактов из актуальных спецификаций [19].

Автоматизация процессов документирования и архитектурного моделирования ведет к значительному снижению затрат на управление изменениями. Стандартизированные пайплайны DocOps позволяют избежать ошибок, связанных с человеческим фактором, минимизируя издержки на ручную проверку, валидацию и синхронизацию артефактов. В контексте корпоративных ИТ-инфраструктур это дает экономию как на прямых операционных расходах, так и на рисках

документации: она становится частью инфраструктурного кода. Это открывает перспективы для внедрения политики «инфраструктура как знание» (Infrastructure-as-Knowledge), где архитектура, документация и процессы проектирования формируют единый цифровой двойник ИТ-системы. Таким образом, интеграция данных концепций позволяет снизить стоимость владения документацией, сформировать быстрое реагирование на изменение требований, оптимизировать процесс согласования через использование стандартных пайплайнов.

Синергия между DocOps и ArchOps создает комплементарную систему, где изменения в архитектуре автоматически актуализируют документацию, обеспечивая единый источник истины для всех заинтересованных сторон. Формирование автоматизированной архитектуры замкнутого цикла способствует сокращению рисков рассогласования и повышению скорости поставки. Таким образом, интеграция DocOps и ArchOps обеспечивает сквозную автоматизацию как кода и инфраструктуры, так и проектных решений с их документационным сопровождением, что критически важно для сложных и быстро эволюционирующих систем. Наибольшее распространение ArchOps и DocOps получили в высокорегулируемых отраслях: финтех, государственные информационные системы, фармацевтика. Здесь необходима строгая прослеживаемость артефактов и наличие процедур валидации. Компании, ориентированные на облачные решения и мультиоблачную архитектуру, также активно интегрируют ArchOps как способ динамической оркестрации архитектурных решений.

2.4. Architecture-as-code

Architecture-as-code представляет собой методологическую парадигму в области инженерии программного обеспечения, ориентированную на преобразование процессов определения, управления и эволюции архитектуры системы. Основная идея заключается в формализации архитектурных определений через машинно-интерпретируемые, версионизируемые спецификации, которые интегрированы непосредственно в жизненный цикл разработки. Этот подход рассматривает архитектуру не как набор статических артефактов, а как динамическую, исполняемую модель, которая постоянно взаимодействует с процессом реализации [20].

Четкая формализация архитектурных решений снижает издержки на проектирование, согласование и сопровождение инфраструктур. Это особенно важно при горизонтальном масштабировании, где стоимость ошибок архитектуры возрастает экспоненциально. Кроме того, автоматизация обновления архитектурных моделей снижает затраты на аудит и внутренние проверки, особенно в распределенных командах. Архитектура как код открывает путь к созданию самоадаптирующихся систем, где архитектурные решения принимаются автоматически на основе мониторинга и анализа. Это особенно актуально в условиях быстроменяющихся требований и архитектурной гибкости, необходимой для внедрения ИИ и обработки больших данных.

Таблица 4 содержит основные характеристики architecture-as-code.

Таблица 4. Характеристики architecture-as-code

Характеристики	Аспекты
(1) Кодоцентричный подход к определению и управлению архитектурой	Диаграмма как код (моделирование)
	Согласование между архитектурой и кодом
	Версионирование
	Прослеживаемость
(2) Эволюция архитектуры, основанная на автоматизации	Автоматическая генерация архитектурных диаграмм, документации и т.д.
	Автоматический анализ и оценка качества
	Интеграция с практиками разработки
(3) Доступность и вовлечение заинтересованных сторон	Доступные языки (DSLs)
	Структурированный словарь
	Универсальный язык
	Модели жизненного цикла разработки программного обеспечения, способствующие вовлечению заинтересованных сторон
	Модели жизненного цикла разработки программного обеспечения, поддерживающие эмерджентную архитектуру

Источник: [20]

Следует отметить, что концепция "Архитектура как Код" (AaC) принципиально отличается от парадигмы Infrastructure-as-code. Некоторые исследователи ошибочно используют эти термины в качестве синонимов [21]. В то время как IaC, реализуемая с помощью инструментов таких как Terraform и Ansible, сосредоточена на автоматизации развертывания и управления вычислительной инфраструктурой (платформой исполнения), AaC функционирует на уровне логической структуры и проектных решений самого прикладного программного обеспечения. Она определяет внутреннюю организацию, компонентный состав и правила взаимодействия приложения. Таким образом, IaC отвечает за среду исполнения (где), тогда как AaC обеспечивает архитектурную целостность приложения (что и как). Обе парадигмы используют код и автоматизацию, но ориентируются на различные уровни абстракции в технологическом стеке.

Глобально можно выделить следующие экономические эффекты:

- уменьшение затрат на изменение архитектуры;
- снижение рисков ошибок при внедрении новых решений и затрат на исправление багов;
- оптимизация планирования ресурсов через четкое описание архитектуры.

2.5. Content first

Подход Content First утверждает фундаментальный принцип проектирования документации: структура и дизайн должны определяться реальным содержанием, а не наоборот. Он требует отказа от макетов с заполнителями на ранних стадиях в пользу использования прототипного или финального контента. Это гарантирует, что итоговый продукт максимально соответствует пользовательским потребностям в доступной и хорошо структурированной информации. Ключевой шаг в рамках Content First – раннее определение типов, объема и организации контента на основе анализа целевой аудитории, до детальной проработки визуального представления.

Ставка на контент как основной актив позволяет повысить вовлеченность пользователей, сократить затраты на поддержку и повысить показатель self-service. Компании, применяющие Content First, фиксируют снижение нагрузки на техническую поддержку до 40% за счет проактивного информирования пользователей. Это особенно важно в условиях масштабируемых SaaS-продуктов, где рост пользовательской базы не должен приводить к линейному росту издержек. Данный подход активно используется в образовательных платформах, медицинских информационных системах и цифровых маркетплейсах. Он особенно актуален для продуктов, ориентированных на пользовательскую автономность, таких как low-code и no-code решения, где понятность интерфейсов и документации критична для экономической жизнеспособности модели.

Одним из дополнений Content First подхода является Topic-Based Authoring. Он предлагает разбивать информацию на минимальные, самодостаточные модули – темы. Каждая тема фокусируется на одной микроцели (объяснении концепта, выполнении задачи, предоставлении справки) и строится по принципам минимализма и пользовательской центрированности, исключая избыточную информацию. Автономность тем достигается за счет гипертекстовых связей.

Инновационность Content First выражается в тесной интеграции контентной стратегии и разработки продукта. Это позволяет строить персонализированные знания, адаптируемые под конкретного пользователя или рынок. В перспективе присутствует возможность генерации пользовательского опыта на основе анализа поведения в реальном времени и адаптивного контента.

2.6. Living documentation

В контексте обсуждения концепций DocOps и подхода "content first" стоит упомянуть понятие living documentation, которое представляет собой прогрессивный подход к системной документации, характеризующийся своей динамичной природой и соответствием автоматизированным тестам приемки, особенно при использовании методологии Behavior-Driven Development (BDD) [22]. Ключевым отличием от традиционных статических документов является наиболее точное и актуальное отражение текущего состояния системы, что способствует не только снижению рисков, связанных с регрессией, но и способствует эффективной передаче знаний в крупных проектах. Актуальность документации в living documentation обеспечивается за счет использования динамичных артефактов, которые часто создаются и обновляются при помощи автоматизации. Это и обеспечивает соответствие документации фактическому поведению системы [15].

В рамках данного подхода документация трансформируется в динамический элемент управления качеством и знаниями. Она становится интерфейсом между командами, арбитром соответствия реализации требованиям и средством обучения новых сотрудников без участия менторов. В условиях

развития искусственного интеллекта и цифровых помощников Living Documentation может стать обучающей средой для систем поддержки принятия решений.

2.7. Diagram-as-code

Подход diagram-as-code революционно трансформирует создание технических визуализаций. Вместо ручного рисования в графических редакторах (например, Visio) диаграммы описываются кодом с использованием DSL, который затем интерпретируется специальными инструментами, что обеспечивает согласованность, версионность и автоматизацию [23]. Некоторые исследователи рассматривают этот подход как составную часть Architecture-as-code [20]. Этот метод интегрируется в общую экосистему разработки: исходники диаграмм хранятся в репозиториях Git вместе с Markdown-документацией, что позволяет применять практики code review, CI/CD и ветвления. Инструменты diagram-as-code и соответствующие им нотации представлены в таблице 5. Инструмент Draw.io рассматривался с точки зрения возможности редактирования кода, а не с точки зрения интерактивного WYSIWYG редактора.

Таблица 5. Сравнение инструментов diagram-as-code

Инструмент	Архитектурная нотация	Язык разметки
Diagrams (Python)	C4, UML, Graphviz	Python-код
Mermaid	Mermaid	Mermaid-синтаксис
PlantUML	UML	PlantUML-синтаксис
Structurizr	C4	Structurizr DSL
Graphviz	DOT	DOT-язык
Kroki	C4, UML, Mermaid и др.	Зависит от нотации
Draw.io	Произвольная	XML
D2	D2	O2-синтаксис
Asciidoctor Diagram	UML, Graphviz и др.	AsciiDoc + встроенные нотации

Источник: составлено автором

Стоит отметить, что сравниваемые инструменты подхода diagram-as-code требуют кастомизации при документировании крупных архитектур. При работе с архитектурами информационных систем, они могут уступать проприетарным инструментам [20]. Стоит отметить также, что diagram-as-code, как и многие другие инструменты документирования с помощью кодом, может автоматизировано при помощи использования системных дескрипторов элементов информационной инфраструктуры [23].

Использование диаграмм как кода снижает затраты на архитектурное проектирование, коммуникацию между командами и визуальную верификацию решений. При этом обеспечивается консистентность визуализаций, что критично для масштабируемых проектов с большим количеством участников. Кроме того, Diagram-as-Code позволяет анализировать изменения не только в коде, но и в архитектуре, что снижает стоимость технического долга. Данный подход дает возможность формировать визуальные цифровые двойники, динамически обновляющиеся в ответ на изменения архитектуры. Это открывает путь к автоматическому архитектурному анализу, прогнозированию влияния изменений и применению визуальной аналитики как элемента стратегии управления ИТ-ландшафтом.

3. Вызовы и последствия

Несмотря на очевидные преимущества описанных подходов, внедрение современных практик документирования в масштабных цифровых системах сопровождается рядом вызовов и потенциальных последствий, которые необходимо учитывать при разработке стратегии цифровой трансформации.

Многие подходы, такие как Docs-as-Code, Living Documentation или Architecture-as-Code, требуют существенного изменения рабочих процессов. Организациям необходимо инвестировать в обучение сотрудников, адаптацию инфраструктуры, настройку CI/CD-пайплайнов и интеграцию новых инструментов. Эти затраты часто оказываются выше первоначальных ожиданий, особенно в компаниях с устоявшимися процессами. Переход к новым форматам документирования требует не только технических изменений, но и трансформации корпоративной культуры. Сотрудники могут проявлять сопротивление новым подходам, воспринимая их как дополнительную нагрузку или угрозу своей квалификации. Без эффективного управления изменениями такие инициативы могут

пробуксовывать или не приносить ожидаемого эффекта. Также при отсутствии должной координации подходы типа Content First или Diagram-as-Code могут привести к дублированию информации, расхождению версий и нарушению целостности знаний. Особенно это критично в масштабных распределенных командах и при активном использовании микросервисной архитектуры. Стоит упомянуть, что инструменты, поддерживающие современные практики документирования, часто являются open-source или нишевыми. Это создает определенные сложности при масштабировании решений, особенно в корпоративной среде с требованиями по безопасности, интеграции с внутренними системами и долгосрочной поддержке.

Хотя современные подходы направлены на уменьшение технического долга, при неправильном внедрении они могут его усугубить. Например, автоматизированные пайплайны документации, не поддерживаемые в актуальном состоянии, могут создавать иллюзию достоверности, вводя разработчиков и менеджеров в заблуждение. В условиях высокой скорости изменений автоматизация не отменяет необходимость экспертного контроля качества документации. Это требует внедрения ролей и практик, ориентированных не только на написание, но и на архитектурную валидацию и структурный аудит документов. Важным моментом является регуляторный аспект. В отраслях с высоким уровнем регулирования (медицина, финансы, государственные услуги) автоматизированные подходы к документированию должны соответствовать требованиям к хранению, обновлению и контролю версий. Невыполнение этих требований может повлечь за собой юридические последствия, штрафы и репутационные потери.

4. Заключение

Автоматизация процессов документирования в цифровых системах представляет собой важнейшее направление повышения эффективности современных ИТ-практик. Результаты исследования показывают, что переход к концепциям docs-as-code, diagram-as-code и architecture-as-code позволяет организациям не только упростить процессы создания и сопровождения документации, но и существенно снизить связанные с этим издержки. Эти подходы обеспечивают интеграцию документации в единый контур жизненного цикла разработки, минимизируя рассогласования между кодом и описаниями, ускоряя принятие архитектурных решений и улучшая воспроизводимость знаний внутри команды. Практики DocOps и ArchOps формируют инфраструктуру сквозной автоматизации, в которой документация синхронизируется с кодом, архитектурой и тестами, что снижает технический долг и повышает устойчивость систем. Инструментальные средства, применяемые в этих подходах, оказывают значительное влияние на метрики экономической эффективности – от уменьшения времени вывода продукта на рынок до оптимизации затрат на сопровождение. В дополнение к этому, внедрение технологий искусственного интеллекта (в частности, LLM) в документационные пайплайны открывает новые возможности для адаптивного и масштабируемого управления знаниями [24]. Таким образом, автоматизация документирования рассматривается как системный, мультидисциплинарный механизм, объединяющий инженерные и экономические принципы в рамках цифрового управления.

Внедрение современных подходов к документированию – это не только путь к инновациям и экономической эффективности, но и вызов, требующий стратегического управления, оценки рисков и зрелости организационной культуры. Компании, осознающие эту двойственность, получают конкурентное преимущество за счет способности адаптироваться к изменениям, формировать устойчивые потоки знаний и ускорять инновационный цикл с минимальными потерями.

Литература

1. Azad N., Hyrynsalmi S. DevOps Challenges in Organizations: Through Professional Lens // Software Business / ed. Carroll N. et al. Cham: Springer International Publishing, 2022. – Vol. 463. – P. 260–277.
2. Borowa K., Zalewski A., Kijas S. The Influence of Cognitive Biases on Architectural Technical Debt // 2021 IEEE 18th International Conference on Software Architecture (ICSA). Stuttgart, Germany: IEEE, 2021. – P. 115–125.
3. Oliveira S.M., Lucrédio D. Guidelines for Data Engineering Documentation in a DevDocOps Approach // Anais do XVIII Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software (SBCARS 2024). Brasil: Sociedade Brasileira de Computação, 2024. – P. 31–40.
4. Announcing TechDocs: Spotify's docs-like-code plugin for Backstage | Backstage Software Catalog and Developer Platform [Электронный ресурс]. 2020. URL: <https://backstage.io/blog/2020/09/08/announcing-tech-docs/> (дата обращения: 05.06.2025).

5. Why we use a ‘docs as code’ approach for technical documentation – Technology in government [Электронный ресурс]. 2017. URL: <https://technology.blog.gov.uk/2017/08/25/why-we-use-a-docs-as-code-approach-for-technical-documentation/> (дата обращения: 05.06.2025).
6. iaaras / gostdown [Электронный ресурс] // GitLab. 2024. URL: <https://gitlab.iaaras.ru/iaaras/gostdown> (дата обращения: 29.05.2025).
7. Pochet B. Markdown & vous: L’écriture académique au format texte avec Markdown et Pandoc. ULiège Library, 2023. – 68 p.
8. Cadavid H., Andrikopoulos V., Avgeriou P. Improving hardware/software interface management in systems of systems through documentation as code // Empir Software Eng. – 2023. – Vol. 28, № 4. – P. 100.
9. Andel B. Continuous Documentation: Automating Document Preparation with your DevSecOps Pipeline // 2022 IEEE 29th Annual Software Technology Conference (STC). Gaithersburg, MD, USA: IEEE, 2022. – P. 156–165.
10. Slaughter A.E. et al. Continuous Integration, In-Code Documentation, and Automation for Nuclear Quality Assurance Conformance // Nuclear Technology. Taylor & Francis, 2021.
11. Quetzalli A. Docs-as-Ecosystem: The Community Approach to Engineering Documentation. Berkeley, CA: Apress, 2023.
12. Chiu K.K., Marquez S., Asundi S. Model based systems engineering with a docs-as-code approach for the SeaLion CubeSat project // Syst. 2023.
13. Content operations from start to scale: perspectives from industry experts / ed. Evia C. Blacksburg, VA: Virginia Tech Publishing for the Virginia Tech Academy of Transdisciplinary Studies, 2024.
14. Knuth D.E. Literate Programming // The Computer Journal. – 1984. – Vol. 27, № 2. – P. 97–111.
15. Sotomayor-Beltran C., Barriaes A.L.F., Lara-Herrera J. Work in progress: The impact of using LATEX for academic writing: A Peruvian engineering students’ perspective // 2021 IEEE World Conference on Engineering Education (EDUNINE). Guatemala City, Guatemala: IEEE, 2021. – P. 1–4.
16. Birkenkrahe M. Teaching Data Science with Literate Programming Tools // Digital. – 2023. – Vol. 3, № 3. – P. 232–250.
17. Bauer P.C., Landesvatter C. Writing a reproducible paper with RStudio and Quarto. Open Science Framework, 2023.
18. Silva Cardoso Rodrigues J.M., Ferreira Ribeiro J.E., Aguiar A. Improving Documentation Agility in Safety-Critical Software Systems Development For Aerospace // 2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). Charlotte, NC, USA: IEEE, 2022. – P. 222–229.
19. Castellanos C., Correal D., Rodriguez J.-D. Executing Architectural Models for Big Data Analytics // Software Architecture / ed. Cuesta C.E., Garlan D., Pérez J. Cham: Springer International Publishing, 2018. – Vol. 11048. – P. 364–371.
20. Bucaioni A. et al. Architecture as code // International Conference on Software Architecture, 2025.
21. Shapkin P. Automation of Configuration, Initialization and Deployment of Applications Based on an Algebraic Approach // Procedia Computer Science. 2022. – Vol. 213. – P. 785–792.
22. Myklebust T., Stålhane T., Vatn D.M.K. Documentation, Information, and Work Products // The AI Act and The Agile Safety Plan. Cham: Springer Nature Switzerland, 2025. – P. 105–108.
23. Nicacio J., Petrillo F. Towards improving architectural diagram consistency using system descriptors // 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC). Madrid, Spain: IEEE, 2021. – P. 401–405.
24. Hou X. et al. Large Language Models for Software Engineering: A Systematic Literature Review // ACM Trans. Softw. Eng. Methodol, 2024. – Vol. 33, № 8. – P. 1–79.