УДК 004.01

Брайловский Андрей Валерьевич

Brailovskii Andrei Valerevich

Ассистент

Assistant

МИРЭА - Российский технологический университет

MIREA - Russian University of Technology

Москва, Россия

Moscow, Russia

АНАЛИЗ ПРИМЕНЕНИЯ СИСТЕМ КОНТРОЛЯ ВЕРСИЙ ДЛЯ УПРАВЛЕНИЯ ЖИЗНЕННЫМ ЦИКЛОМ ТЕХНИЧЕСКОЙ ДОКУМЕНТАЦИИ

Аннотация. Статья посвящена выявлению и анализу ключевых недостатков традиционных подходов к управлению жизненным циклом технической документации. Показано, что в условиях современной Agile- и DevOps-разработки существующие практики не способны обеспечить необходимую скорость, актуальность и прозрачность работы с документацией. В качестве решения рассматривается парадигма «Документация как код» (Docs as Code, DaC), основанная на применении методов и инструментов, зарекомендовавших себя в разработке программного обеспечения. Особое внимание уделено системам контроля версий (VCS), в частности Git, которые позволяют эффективно решать задачи отслеживания изменений, организации совместной работы и автоматизации процессов. Обосновано, что внедрение DaC способствует не только повышению качества и надёжности документации, но и улучшению взаимодействия между техническими писателями, разработчиками и другими специалистами, что существенно повышает производительность команд и сокращает издержки.

Ключевые слова

Документация как код, DaC, системы контроля версий, VCS, Git, жизненный цикл документации, управление версиями, совместная работа,

автоматизация, техническая документация, Agile, DevOps, управление конфигурациями.

Abstract

This article identifies and analyzes the key shortcomings of traditional approaches to managing the technical documentation lifecycle. It demonstrates that in the context of modern Agile and DevOps development, existing practices fail to provide the necessary speed, relevance, and transparency for working with documentation. The "Docs as Code" (DaC) paradigm, based on applying methods and tools proven in software development, is examined as a solution. Special attention is given to version control systems (VCS), particularly Git, which effectively solve tasks of change tracking, collaboration, and process automation. It is substantiated that implementing DaC not only enhances the quality and reliability of documentation but also improves collaboration between technical writers, developers, and other specialists, significantly boosting team productivity and reducing costs.

Keywords

Docs as code, DaC, version control systems, VCS, Git, documentation lifecycle, version management, collaboration, automation, technical documentation, Agile, DevOps, configuration management.

Введение

Стремительное усложнение программных продуктов и одновременное сокращение времени их вывода на рынок ставят перед IT-индустрией новые вызовы. В условиях глобальной цифровизации требования к скорости, надёжности и безопасности информационных систем постоянно растут. На этом фоне кардинально меняется и роль технической документации: из сопроводительного материала она превращается в критически важный компонент, напрямую влияющий на внедрение, эксплуатацию и развитие продукта. Однако традиционные подходы, при которых документация представляет собой напечатанную инструкцию, создаваемую отдельно от разработки и постфактум, продемонстрировали свою полную несостоятельность в реалиях гибких (Agile) и DevOps-методологий [2].

Ответом на эти вызовы стала парадигма «Документация как код» (Docs as Code, DaC). Её основной принцип — применять к работе с документацией те же инженерные практики и инструменты, которые десятилетиями доказывали свою эффективность в разработке программного обеспечения [6]. Центральный элемент этой парадигмы — системы контроля версий (VCS), которые предоставляют для неё технологическую основу. Важно понимать, что переход на DaC — это не просто техническая модернизация, а серьёзное изменение в культуре и организации работы [3]. Оно направлено на то, чтобы устранить барьеры между командами и тесно связать процессы разработки ПО и создания документации в единый, непрерывный и синхронизированный поток. Цель данного исследования — рассмотреть проблемы традиционного жизненного цикла документации и показать, как их решают системы контроля версий.

Проблематика традиционных подходов к управлению документацией

Классические модели управления документацией, несмотря на свою привычность, сопряжены с целым набором системных проблем, которые лежат в основном в плоскости организации процессов [4]. Эти недостатки напрямую снижают качество конечного продукта, приводят к избыточным трудозатратам и создают существенные риски для проектов.

Поддержание точности и актуальности контента становится по-настоящему острой проблемой. В условиях итеративной разработки, когда функциональность продукта может меняться ежедневно, обеспечение соответствия документации продукту становится практически невыполнимой задачей. Из-за отсутствия прямой автоматизированной связи между исходным кодом и текстом документации, их расхождение становится практически неизбежным. Технические писатели зачастую получают информацию об изменениях с опозданием, что приводит к устареванию документации ещё до её публикации. Последствия могут быть очень серьёзными: от введения пользователей в заблуждение до значительного роста нагрузки на службы технической поддержки, вынужденные компенсировать пробелы в документации.

Управление версиями файлов документации также крайне сложная задача. При отсутствии централизованной системы контроля версий, члены команды неизбежно скатываются к «творчеству». К именам файлов дописывают суффиксы версий, даты, фамилии авторов и так далее. Это быстро превращает любую общую папку в «кладбище» документов с названиями вроде «Инструкция_v1.2_финал.docx», «Инструкция_v1.2_правки_Иванова.docx» и, наконец, «Инструкция_v1.2_САМЫЙ_ФИНАЛ_ПРОВЕРЕНО.docx». В такой среде невозможно быстро понять, какая из версий является действительно последней, кто, когда и зачем внёс ту или иную правку, и тем более сравнить документы между собой [5]. При этом невозможность быстро откатиться к предыдущему стабильному состоянию создаёт серьёзные риски, особенно при необходимости срочного восстановления информации.

Организация эффективной совместной работы представляет собой ещё один значительный вызов. Качественная документация всегда является результатом тесного взаимодействия между техническими писателями, разработчиками, тестировщиками и профильными экспертами [7]. В традиционных процессах, основанных на пересылке файлов по почте, эта работа превращается в хаос. Рецензирование отнимает массу времени и сил на сбор и сведение комментариев из разных источников. Часто возникают конфликтующие правки, разрешение которых требует лишних совещаний и переписок, что замедляет процесс и повышает вероятность потери ценных замечаний.

Наконец, обеспечение единообразия стиля и терминологии в больших массивах документации, создаваемых несколькими авторами, становится почти невыполнимой задачей без автоматизированного контроля [4]. Несогласованность не только выглядит непрофессионально, но и мешает понимать документ, что может привести к неверному толкованию важных инструкций. Ручная вычитка и приведение документации к единому стандарту отнимают массу времени и сил.

Все перечисленные проблемы носят системный характер и ясно говорят,

что пора переходить к более зрелым и инженерным подходам. Наглядный пример сравнения подходов ведения документации (традиционный и «Docs as Code» (DaC)) представлен на Рисунке 1.

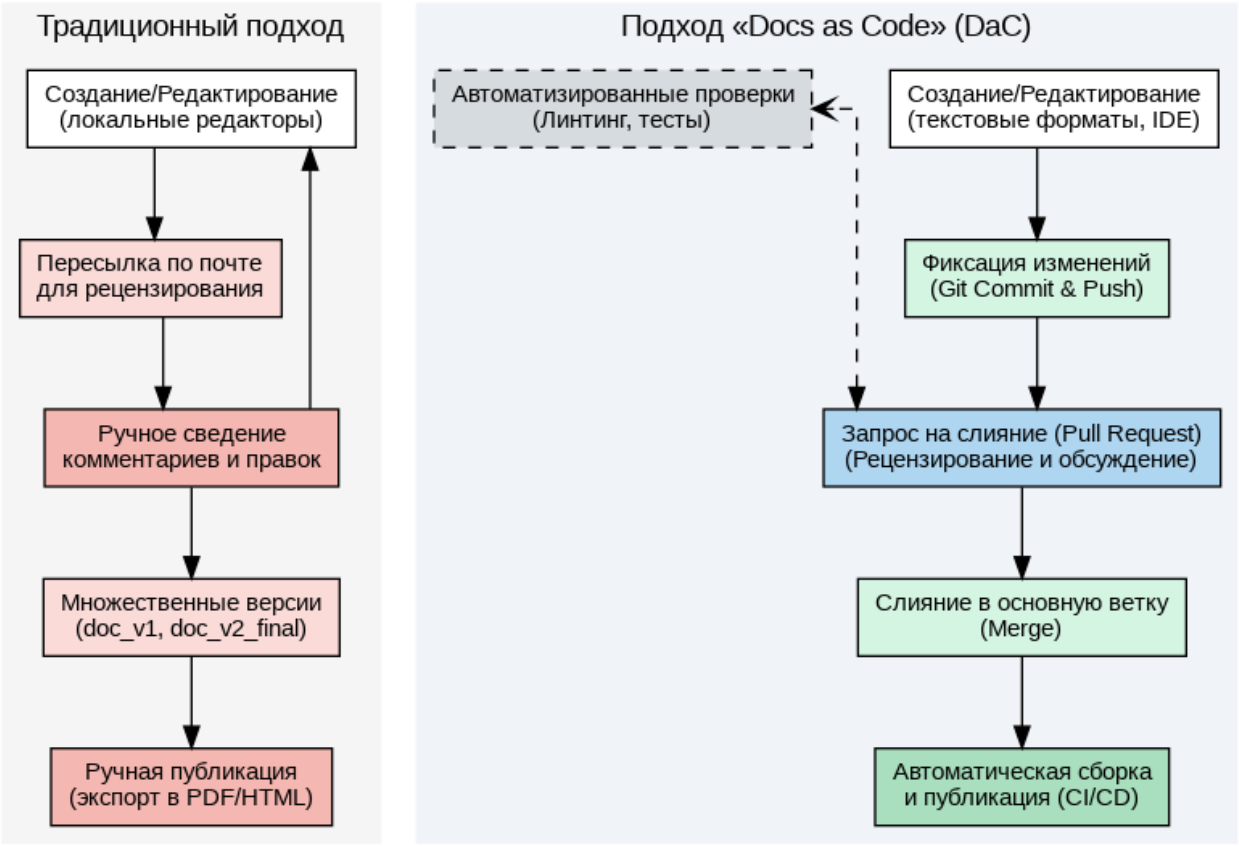

Рисунок 1. Сравнение подходов к ведению документации

Системы контроля версий как методологическое решение

Для решения перечисленных проблем в разработке программного обеспечения уже давно нашли решение — системы контроля версий [1]. Их применение к документации позволяет выстроить всю работу на принципах прозрачности, контролируемости и автоматизации.

Основополагающей функцией VCS является ведение подробной и проверяемой истории всех изменений. Они оперируют на уровне «коммитов» — неделимых наборов правок. Каждый коммит неразрывно связан с информацией о его авторе, точной временной меткой и осмысленным сообщением, описывающим суть правок [1]. Так создаётся полная и честная история документа. Любой участник проекта может в любой момент не только увидеть текущее состояние документа, но и проанализировать всю цепочку изменений,

которые к нему привели. Это обеспечивает высокий уровень контроля и создаёт надёжный аудиторский след. Возможность сравнения любых двух версий или мгновенного отката к любому предыдущему стабильному состоянию сильно повышает надёжность процесса.

Для организации параллельной работы в VCS существует механизм ветвления. Эта концепция позволяет создавать изолированные рабочие среды («ветки») для решения конкретных задач, не затрагивая основную, стабильную версию документации. Например, один специалист может работать над описанием новой функциональности, в то время как другой оперативно исправляет ошибку в уже опубликованном разделе. Эти задачи не пересекаются и не мешают друг другу. По завершении работа из ветки проходит через чёткую процедуру слияния в основную версию [1].

Эта процедура, как правило, реализуется через запросы на слияние (Pull Requests). Это не просто техническая операция, а мощный инструмент для организации процесса рецензирования. Прежде чем изменения будут приняты, они представляются на рассмотрение коллегам, которые могут изучить предложенные правки, оставить комментарии и инициировать обсуждение. Изменения интегрируются в основную версию только после получения необходимого количества одобрений, что создаёт коллективную ответственность за качество.

Применение VCS помогает создать «единый источник истины» для всей проектной документации. Наличие центрального репозитория убирает путаницу с версиями и даёт всем участникам доступ к актуальным файлам [5]. Больше не существует проблемы «финальной» версии на чьём-то локальном диске; единственной актуальной версией всегда является та, что находится в основной ветке репозитория. Распределённая природа большинства современных VCS, таких как Git, делает систему ещё надёжнее: у каждого участника есть полная локальная копия всего репозитория, что позволяет работать автономно и обеспечивает высокую отказоустойчивость.

Размещение документации в том же репозитории, что и исходный код,

помогает их синхронизировать. Разработчики, работая над новой функциональностью, могут в той же ветке создавать или модифицировать соответствующую документацию. Это значительно упрощает её актуализацию и помогает создать культуру, в которой документация рассматривается как неотъемлемая часть продукта [7].

Потенциал автоматизации и эффекты синергии

Ключевое преимущество парадигмы DaC — это возможность автоматизации [6]. Когда документация хранится не в бинарных проприетарных форматах, а в легковесных текстовых языках разметки, таких как Markdown или AsciiDoc, с ней могут работать программы. Это позволяет использовать для неё те же инструменты, что и в DevOps-практиках [2].

Центральным элементом здесь выступают конвейеры непрерывной интеграции и непрерывной доставки (CI/CD). Это автоматические сценарии, которые запускаются при каждом новом изменении в репозитории. В контексте документации такой конвейер может проверять качество по многим параметрам. Например, можно настроить автоматический запуск линтеров — специализированных инструментов, которые проверяют текст на соответствие руководствам по стилю, корректность терминологии и грамматику. Также могут быть встроены проверки на наличие неработающих (например, устаревших) ссылок. В итоге значительная часть рутинной работы по вычитке перекладывается на плечи роботов, освобождая людей для работы над смыслом.

Помимо проверок, CI/CD конвейер полностью автоматизирует сборку и публикацию. После того, как изменения успешно прошли все проверки и были интегрированы в основную ветку, конвейер может запустить превращение исходных текстов в финальный вид — будь то статический веб-сайт или PDF-документ. Для этого используются генераторы статических сайтов (SSG), которые преобразуют текстовую разметку в готовый сайт с навигацией, поиском и современным дизайном. Готовый результат автоматически публикуется на сервере, и пользователи всегда видят самую свежую, проверенную версию.

Совместное использование VCS и автоматизации даёт мощный

синергетический эффект. Процесс создания документации становится предсказуемым, воспроизводимым и прозрачным. Уменьшается влияние человеческого фактора и минимизируется количество ошибок. Обратная связь становится быстрее: авторы и рецензенты могут оперативно взаимодействовать, а автоматические проверки мгновенно сообщают о проблемах. Всё это серьёзно повышает и качество самой документации, и общую эффективность производственного процесса [3].

Заключение

Проведённый анализ показывает, что традиционные подходы к управлению документацией исчерпали себя в условиях современной динамичной разработки. Проблемы актуальности, версионирования, совместной работы и обеспечения согласованности являются системными и требуют кардинального пересмотра рабочих процессов.

Применение систем контроля версий в рамках парадигмы «Документация как код» предлагает хорошо обоснованное решение этих проблем. VCS делают процесс полностью прозрачным и контролируемым, дают надёжную основу для параллельной работы и коллективного рецензирования, а также открывают широкие возможности для автоматизации. Внедрение этих практик позволяет не только значительно повысить качество документации, но и улучшить взаимодействие между техническими писателями, разработчиками и другими специалистами [5].

Следует, однако, признать, что внедрение VCS — это важный, но не единственный шаг. Успех зависит от готовности компании к переменам, а также от того, насколько грамотно будет выстроена вся экосистема, включающая не только саму систему контроля версий, но и сопутствующие инструменты: генераторы статических сайтов, линтеры, платформы для CI/CD и специализированные редакторы [2]. При этом не следует относиться к данному решению как к панацее — любая технология и концепция имеет свои ограничения, и DaC — не исключение. Сложность инструментов для нетехнических специалистов, необходимость поддержки целого стека

технологий и неудобство рецензирования текстовой разметки вместо визуального документа создают определённые барьеры. Таким образом, выбор в пользу DaC должен быть взвешенным решением, учитывающим не только потенциальные выгоды, но и готовность инвестировать в обучение персонала и построение соответствующей технической инфраструктуры.

Список источников:

1. Чакон С., Штрауб Б. Git для профессионального программиста / С. Чакон, Б. Штрауб; пер. с англ. И. Размайкина. – СПб.: Питер, 2024. – 494 с.
2. Ким Д., Хамбл Д., Дебуа П., Уиллис Д. Руководство по DevOps. Как добиться гибкости, надёжности и безопасности мирового уровня в технологических компаниях / пер. с англ. – М.: Манн, Иванов и Фербер, 2018. – 480 с.
3. Форсгрен Н., Хамбл Д., Ким Д. Ускоряйся! Наука DevOps. Как создавать и масштабировать высокопроизводительные технологические организации / пер. с англ. – М.: Олимп-Бизнес, 2020. – 368 с.
4. Макаровских Т. А. Документирование программного обеспечения. В помощь техническому писателю. – Екатеринбург: Издательские решения, 2022. – 212 с.
5. Бабич А. Делаем документацию здорового человека в Git на примере Docs Ozon [Электронный ресурс] // Хабр. – 2022. – Режим доступа: https://habr.com/ru/companies/ozontech/articles/695868/ – Дата обращения: 14.06.2025.
6. Сергеев А. Почему As Code — это не просто тренд, а новая реальность разработки [Электронный ресурс] // Tproger. – 2023. – Режим доступа: https://tproger.ru/articles/pochemu-as-code---eto-ne-prosto-trend--a-novaya-realnost-razrabotki/ – Дата обращения: 14.06.2025.
7. Кузнецов Д., Певнева А. Создание системы документирования, или как мы от «ворда» к docs as code за месяц переходили [Электронный ресурс] // Хабр. – 2022. – Режим доступа: https://habr.com/ru/companies/cloud_ru/articles/686050/ – Дата обращения:

14.06.2025.