

# DocOps на Flow 2022

29-30 ноября прошла конференция для аналитиков [FlowConf 2022](#). Основная особенность конференции — ее ориентация на конкретные практические рецепты. Одним из направлений, которое содержит много таких рецептов, стал Docs As Code или, в более широком смысле, DocOps в работе аналитика. В этом посте представляю обзор этого направления.

Часть конференции проходила в открытом режиме (community days). Для докладов, попавших в эту часть, приведены прямые ссылки на видеозапись. Для остальных докладов указана ссылка на описание, где можно скачать презентацию.

Сергей Гришанов и Евгений Зингер [рассказали](#) о том, как в Тинькофф пришли к практике хранения документации в одном репозитории с кодом (Docs as Code). Спикеры работают по разным направлениям. У каждого из этих направлений разные задачи и организация процессов. Тем не менее результат получился одинаковый: в обеих командах значительно улучшилось взаимодействие аналитиков и разработчиков, а документация теперь полностью соответствует текущему состоянию информационного продукта.

Ссылка на видео: <https://www.youtube.com/watch?v=vW6haSf6kug>

Роман Цирульников [показал](#), как в ЮMoney используется Docs as Code для организации репозитория архитектуры, где этот подход также показал свою эффективность.

Никита Харичкин [провел](#) мастер-класс по использованию диаграмм последовательности (sequence diagram) в PlantUML. Никита давно рассказывает об этой теме, каждый раз находя всё больше и больше возможностей в данном инструменте для решения ежедневных практических задач.

Ссылка на видео: <https://www.youtube.com/watch?v=ScbZL5RX84E>

В рамках конференции также был проведен круглый стол, в котором собрались технические писатели и аналитики — Константин Валеев, Николай Волынкин, Лана Новикова, Николай Поташников, Семен Факторович, — с целью в принципе ответить на вопрос, как DocOps может помочь в работе аналитика.

Был показан пример, показывающий, что вне зависимости от того, как организована работа с документацией в методологии DocOps, на выходе мы можем получить документы в любом формате в соответствии с требованиями заказчика (html, pdf, docx, ...). Т.е. работая в методологии DocOps мы концентрируемся не на выходных форматах, а на содержании процесса документирования.

Также была рассмотрена ключевая для аналитика проблема — управление требованиями. В дискуссии приняли участие более 80 человек. Были обозначены следующие задачи управления требованиями, традиционно вызывающие проблемы:

1. управление атомарными требованиями и поддержка актуальности;

2. трассируемость вплоть до кода;
3. ведение модели;
4. срезы (представления);
5. связь между текстом и диаграммами;
6. совместная работа;
7. публичное представление;
8. разные выходные форматы документов, формируемые, в том числе, не только из требований;
9. контроль качества;
10. требования должны продолжать жить как документация.

Часть этих проблем DocOps решает понятным образом, например, возможное решение проблемы поддержки актуальности требований и их трассируемости рассмотрено в уже упомянутом докладе Сергея Гришанова и Евгения Зингера. Чаще готовых рецептов нет или они не очевидны.

Были определены технологии, которые потенциально могут обеспечить решение всех указанных проблем.

1. [Shinx-needs](#) позволяет ввести в документацию термины, свойственные для управления требованиями, и обязать их метаданными.
2. [Gherkin](#) позволяет формулировать тесты на языке, одновременно понятном и заинтересованным лицам, и интерпретируемым внутри программного продукта.
3. [Jetbrains MPS](#) и аналогичные инструменты, которые позволяют писать собственные языки. Пример языка описаний требований с помощью JetBrains MPS можно найти [здесь](#).
4. [DocHub](#) — инструмент «всё в одном» описания архитектуры через код (Architecture as a code).
5. Языки, которые содержат удобные средства для создания внутренних DSL (Kotlin DSL, Haskell, F#, Groovy, Ruby, ...).

В частности, был рассмотрен пример создания языка документации на Kotlin DSL. Данный язык является оберткой [Writerside](#), но позволяет в документацию вводить элементы обычных языков программирования — циклы, функции и т.д., которые автоматизируют рутинные операции документирования.

Этот же подход можно использовать, как и в случае со [Shinx-needs](#) для введения в язык собственных элементов, например, для управления структурой требований.

У каждой из указанных технологий есть определенные ограничения. Конечно, хотелось бы иметь универсальные решения или, хотя бы, подходы. Но даже сейчас сам подход DocOps, при котором мы объединяем процессы документирования, разработки и доставки

ИТ-продукта в одно целое, позволяет вполне эффективно подбирать технологии для решения конкретной задачи.

## Выводы

1. Технологии документирования развиваются очень быстро. Еще 10 лет назад все пользовались только MS Word. Сегодня по оценкам участников круглого стола — в лучшем случае 25%.
2. Начать использовать Docs as Code очень просто — есть очень простые подходы, которые сразу дают результат.
3. Возможности DocOps достаточно широки, чтобы эффективно решать практически любые проблемы документирования, стоящие перед аналитиками.

Хочу поблагодарить всех спикеров, экспертов и команду [JUG Ru Group](#) за подготовку и проведение перечисленных мероприятий. Отдельное спасибо Алексею Лобзову за помощь в курировании всего направления DocOps на [FlowConf 2022](#).

P.S. По горячим следам попробовал сделать [обёртку Kotlin DSL для AsciiDoc и эту заметку написать в ней](#). Конечно, писать [менее удобно](#), чем в Markdown, reStructuredText или AsciiDoc. Однако тестировать текст, автоматизировать рутинные операции, создавать собственные элементы языка можно непосредственно внутри проекта, используя привычные инструменты работы с языком Kotlin. Выгрузка в формате Nabr Markdown для публикации тоже получилась очень удобной.