ozon банк

Как не превратить автотесты в объект ненависти

Красные флаги в инфраструктуре

HeisenBug 2026

Привет!

Я Иван Левиков

Специалист по мобильному антифроду в NDA Company

ex TeamLead тестирования контентных сервисов VK и соавтор DeviceHub

11:25

Привет!

Я Даниил Смирнов

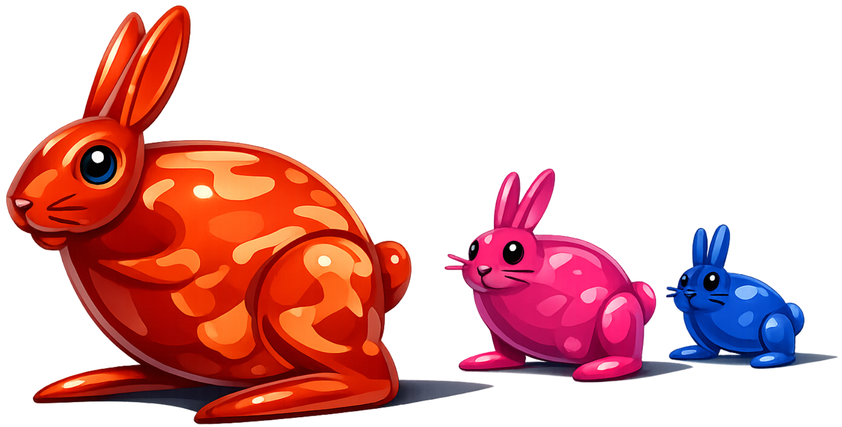
Эксперт по автоматизации тестирования в Ozon Банке

ex TeamLead VK DeviceHub

20:31

О чем мы сегодня поговорим

Мы сделали много проектов для инфраструктуры мобильных автотестов и поднимали её с нуля до тысяч тестов



О чем мы сегодня поговорим

Обсудим, почему инфраструктуру ненавидят?

О чем мы сегодня поговорим

Объясним, что нет единственной «правильной» архитектуры

Как мы считаем уровень команды*

Характеристика	Маленькая	Средняя	Большая
Время прогона, мин	До 5	До 15	15 и выше
Число автотестов	До 100	От 100 до 1000	Больше 1000

*На основе нашего опыта

nda company

Маленькая команда, и совсем немножко тестов

До 100 тестов

Как мы считаем малые команды

Характеристика	Маленькая команда
Устройства	Внешняя запускалка / Эмуляторы
CI/CD	SaaS (GitHub Actions) или выделенный хост
Данные	Фиксированные наборы
Аналитика	Логи в консоли
Импакт-анализ	Ручной, по папкам
Время прогона, мин	До 5
Число автотестов	До 100

Как обычно выглядит этот этап

- Маленькая команда (меньше 5 человек)
- Мало тестов
- Тесты гоняются локально только на релизных регрессах

Что посоветует нам ИИ

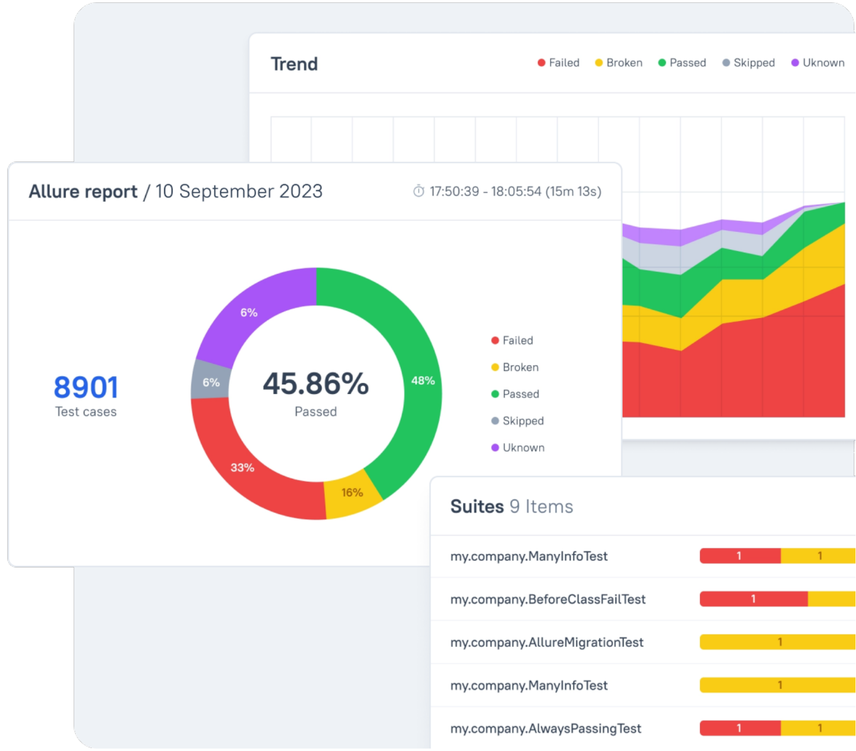
1. Единый CI с первого дня (PR → тесты → стабильный pipeline)
2. Тестируйте критичные сценарии, а не «покрытие ради покрытия»
3. Окружение = Docker / воспроизводимая сборка
4. Наблюдаемость: логи, артефакты, быстрый дебаг
5. Сразу закладывайте масштабирование: независимые тесты, параллелизация

Что стоит сделать сразу

- Написать документацию, о том как все работает

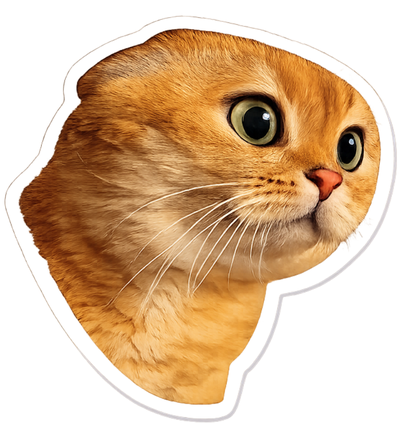
Что стоит сделать сразу

- Написать документацию, о том как всё работает
- Настроить генерацию отчетов



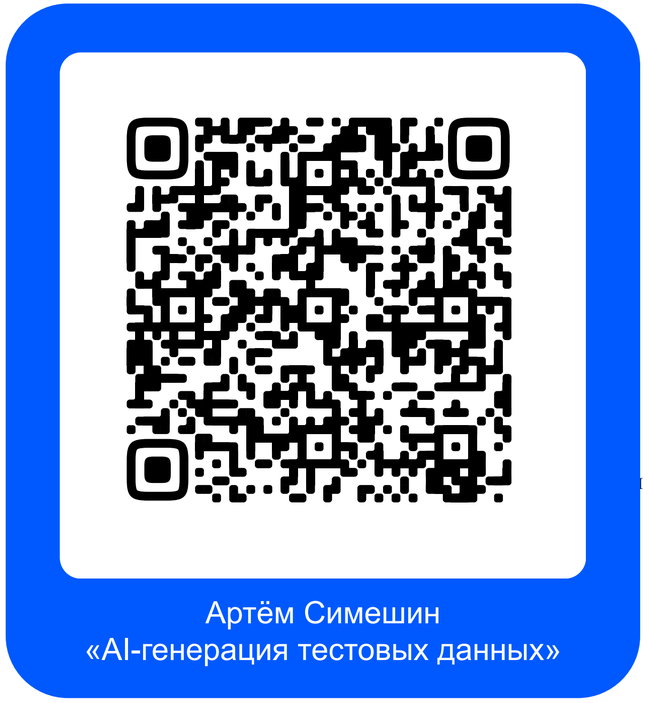
Что стоит сделать сразу

- Написать документацию, о том как всё работает
- Настроить генерацию отчетов
- Обозначить, куда можно прийти с проблемами и кого звать, если возникли сложности



Что стоит сделать сразу

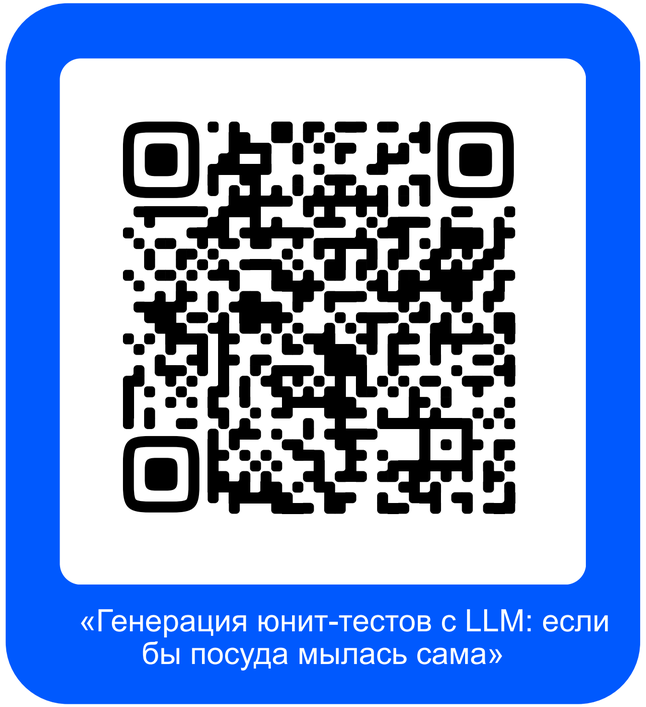
- Написать документацию, о том как всё работает
- Настроить генерацию отчетов
- Обозначить, куда можно прийти с проблемами и кого звать, если возникли сложности
- Продумать генерацию тестовых данных и стабильного окружения



Артём Симешин
«AI-генерация тестовых данных»

Что стоит сделать сразу

- Написать документацию, о том как всё работает
- Настроить генерацию отчетов
- Обозначить, куда можно прийти с проблемами и кого звать, если возникли сложности
- Продумать генерацию тестовых данных и стабильного окружения
- Обучите, при возможности, ИИ писать базовые юнит-тесты на функционал



«Генерация юнит-тестов с LLM: если бы посуда мылась сама»

Как сделать хорошо

- Устройства: одна-две железки в офисе
- Фреймворк: Appium или XCUITest/Espresso
- CI/CD: GitHub Actions или GitLab CI (SaaS-версия)

Как сделать хорошо

- Устройства: одна-две железки в офисе
- Фреймворк: Appium или XCUITest/Espresso
- CI/CD: GitHub Actions или GitLab CI (SaaS-версия)

Выбор фреймворка

- На ранних этапах еще возможно сменить фреймворк, но на поздних стадиях сделать это уже сложно и затратно

«7 best Appium alternatives to reduce flaky»

Выбор фреймворка

- Выбирая **Appium**, вы жертвуете скоростью прогона тестов и сложностью инфраструктуры

- Выбирая **нативный фреймворк**, вы жертвуете скоростью написания автотестов на ранних этапах

Выбор раннера

- Выбор раннеров довольно сильно ограничен
- Не оставайтесь на базовом раннере — лучше сразу выбрать что-то полноценное

«Cookbook Android UI testing»

Как сделать хорошо

- Устройства: одна-две железки в офисе
- Фреймворк: Appium или XCUITest/Espresso
- CI/CD: GitHub Actions или GitLab CI (SaaS-версия)

Пару слов об устройствах

Давайте спросим про устройства у ИИ

У меня есть несколько классов с XCUI-тестами. Как я могу добиться максимальной параллелизации на симуляторах?

Как понять, сколько симуляторов мне нужно?

Давайте спросим про устройства у ИИ

Максимальная параллелизация = классы + CPU + RAM
Симуляторы ≤ число тестовых классов

Формула: симы = min (классы, CPU, RAM)
CPU: ~1 сим = 1 ядро (лучше −20–30%)
RAM: ~4–6 ГБ на сим + 8–10 ГБ системе

Итог: берём минимум и масштабируемся
CPU забит → упёрлись в процессор
swap/флаки → в память

Что не так

ИИ забывает о том, что Xcode при прогоне не дробит тестовые классы

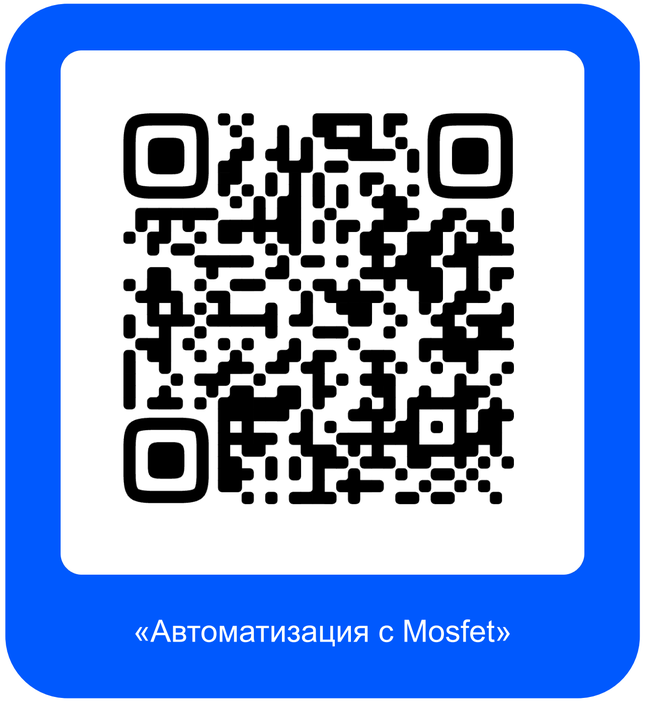
А если мы выбрали физические устройства

Как уменьшить количество болей с физическими устройствами

- Проблема USB-шины. Для реальных устройств (iOS/Android) узкое место

Как уменьшить количество болей с физическими устройствами

- Автоматизируйте переподключение устройства через включение и выключение питания
- Задумайтесь над организацией удаленного доступа к устройствам



«Автоматизация с Mosfet»

Чего точно не стоит делать

- Продолжать использовать ноутбук, тестировщика или разработчика, как раннер тестов
- Изобретать собственные раннеры и прочие велосипеды
- Использовать статичные данные, тестировать на проде

Средняя команда, и тестов уже больше сотни

От 100 до 1000 тестов

Как мы считаем средние команды

Характеристика	Средняя команда
Устройства	Локальная ферма (Grid + STF/DeviceHub)
CI/CD	Self-hosted раннеры
Данные	API сброса / уникальные юзеры
Аналитика	Allure + видео
Импакт-анализ	Автоматический по карте файлов
Время прогона, минут	До 15
Число автотестов	От 100 до 1000

Как обычно выглядит этот этап

- Несколько команд разработки и тестирования
- Все совместно пишут тесты и код в разных ветках
- Есть выделенный DevOps

Как обычно выглядит этот этап

- Тесты прогоняются на нескольких билд-агентах или ферме
- Присутствует оркестрация и параллелизация прогонов
- При росте числа автотестов наблюдается увеличение числа падений автотестов

Что обычно идет не так

- Появляются нестабильные тесты, которые зачастую красные
- Появляется задержка по доставке кода

Что обычно идет не так

- Есть проблемы с тем, чтобы выяснить однозначно причину падения пайплайна
- Не контролируется включение фичей, A/B-экспериментов и прочих изменений на бэкенде

Что посоветует нам ИИ

Импакт-анализ
Делим тесты по функционалу, в PR запускаем только нужные (git diff / теги)
Результат: 3–5 мин вместо 20

Анализ прогона
Смотрим тайминги в CI, если этап > 50% — оптимизируем (например, эмуляторы)

Кэш и прекондишены
Кэш зависимостей (Gradle, npm, CocoaPods), готовые образы вместо установки с нуля

Что стоит сделать в первую очередь

- Поговорите с QA
- Поговорите с разработкой
- Реализуйте точку входа для документации

Что стоит сделать в первую очередь

- Поговорите с QA
- Поговорите с разработкой

- Реализуйте точку входа для документации

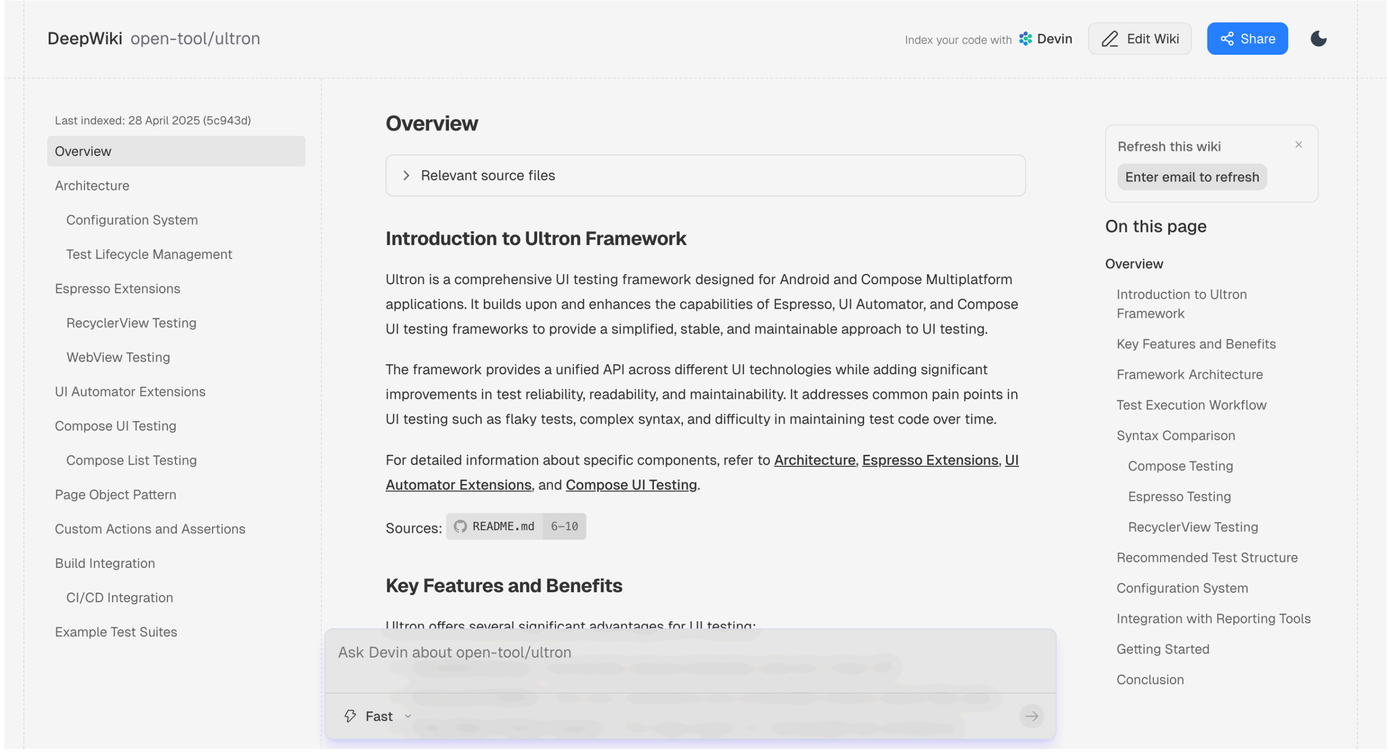
Что стоит сделать в первую очередь

- Поговорите с QA
- Поговорите с разработкой

- Реализуйте точку входа для документации

Автоматизация документации

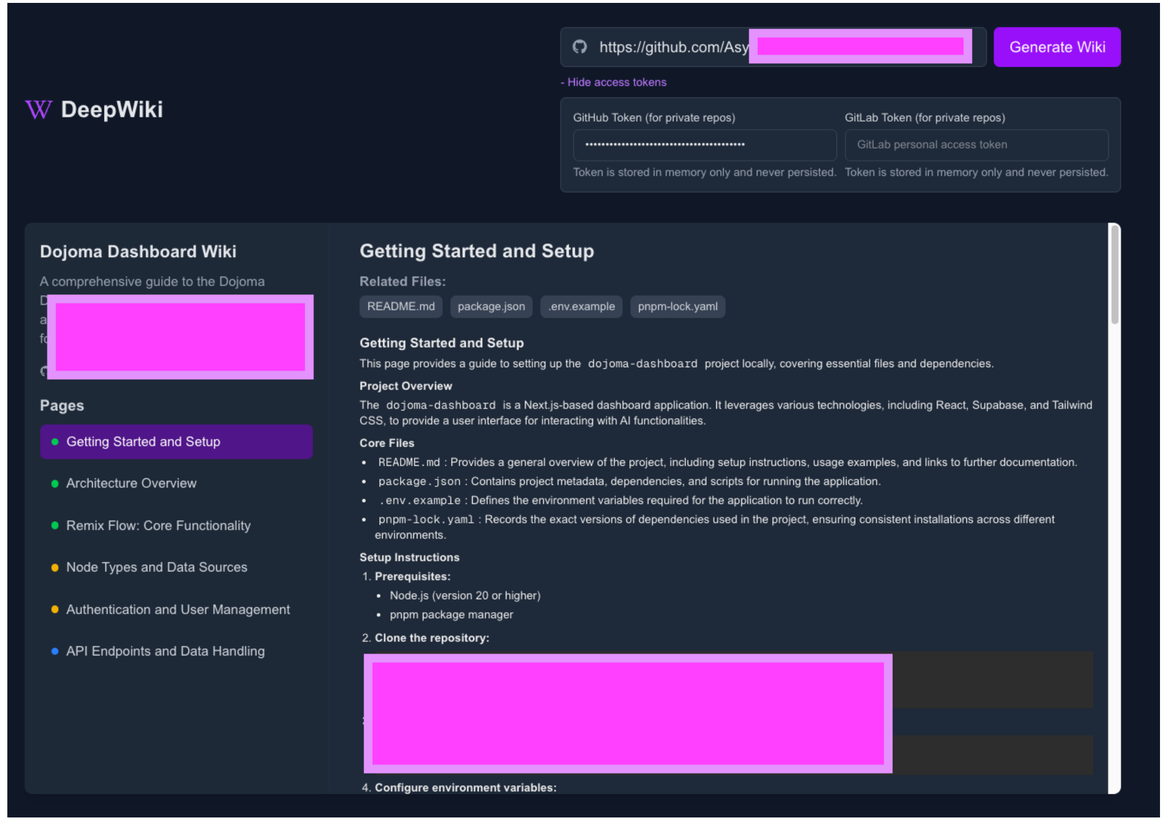
- Для OpenSource проекта



DeepWiki.com - Ultron

Автоматизация документации

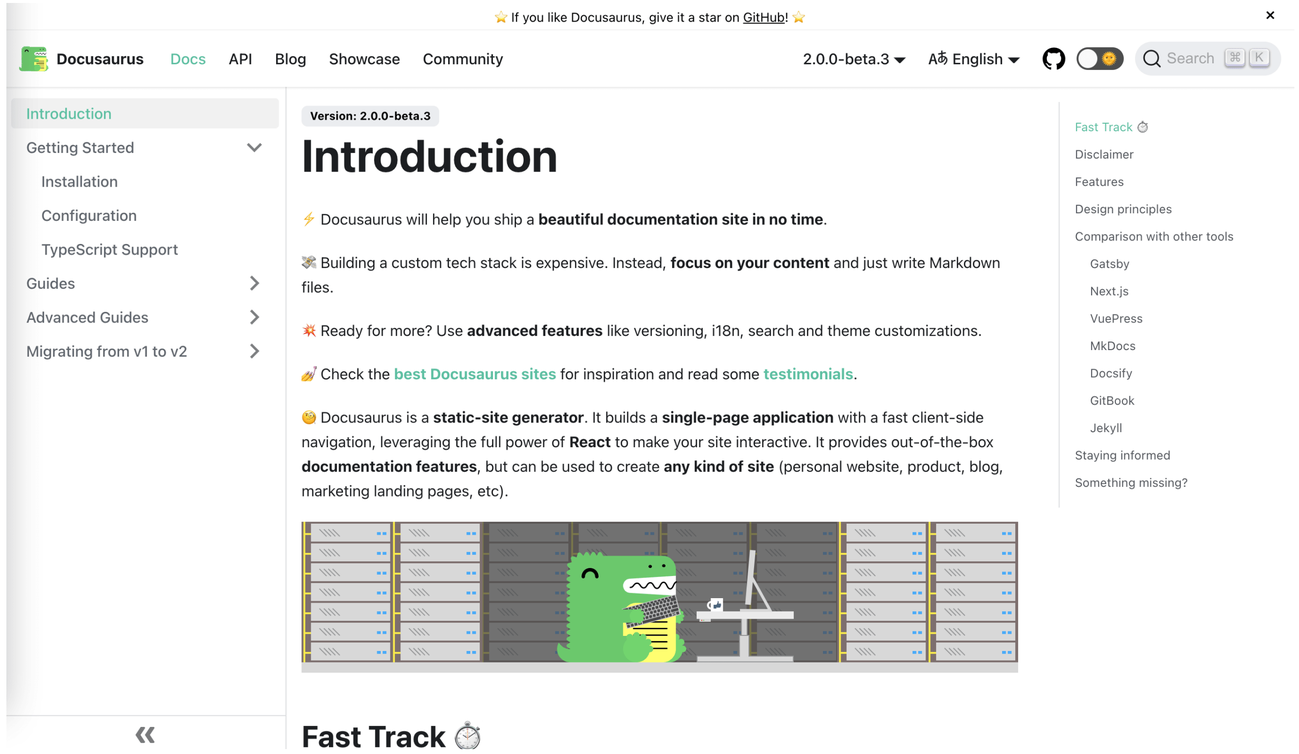
- Для закрытого проекта



DeepWiki.com - Local

Автоматизация документации

- Если хочется без ИИ



Организация Meta, а также её продукты на которые мы ссылаемся в этой статье, признаны экстремистскими на территории РФ

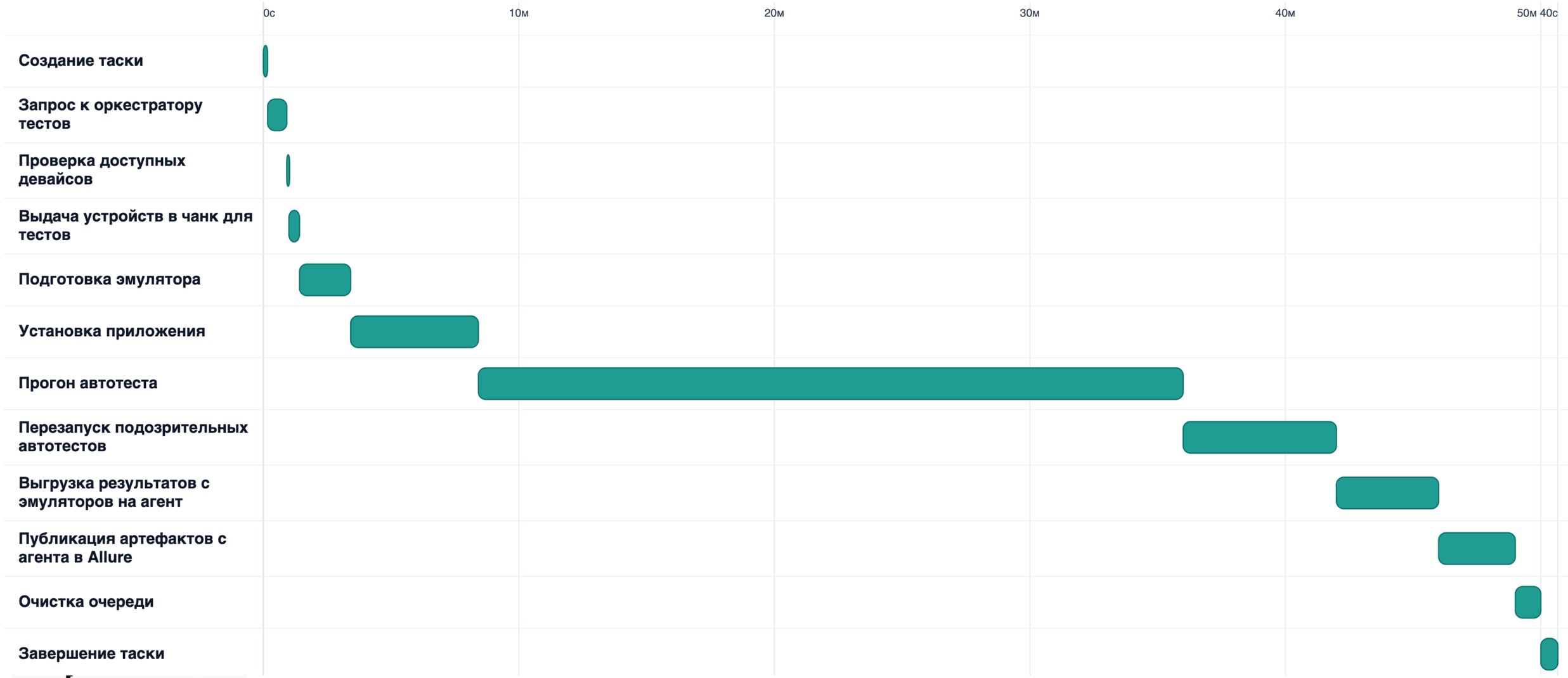
Оптимизация и ускорение прогонов

Анализируем этапы прогона

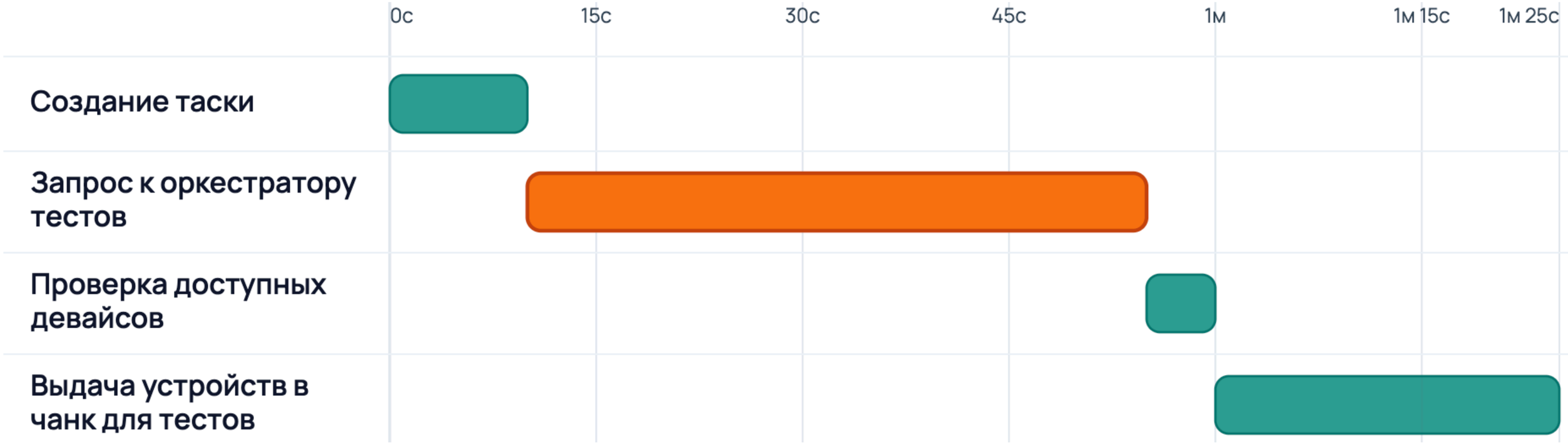
Анализируем прогон

- Разделить прогон на этапы
- Посмотреть, какой этап прогона выбивается из нормальных значений
- Оценить временные затраты на каждый этап прогона

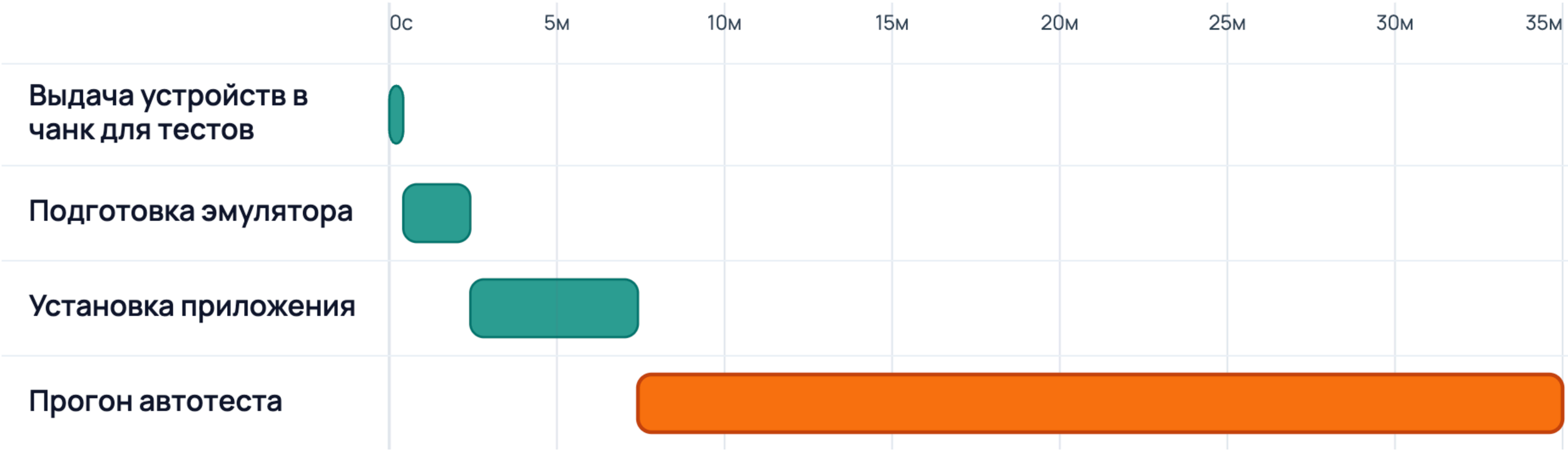
Где чаще всего теряется время



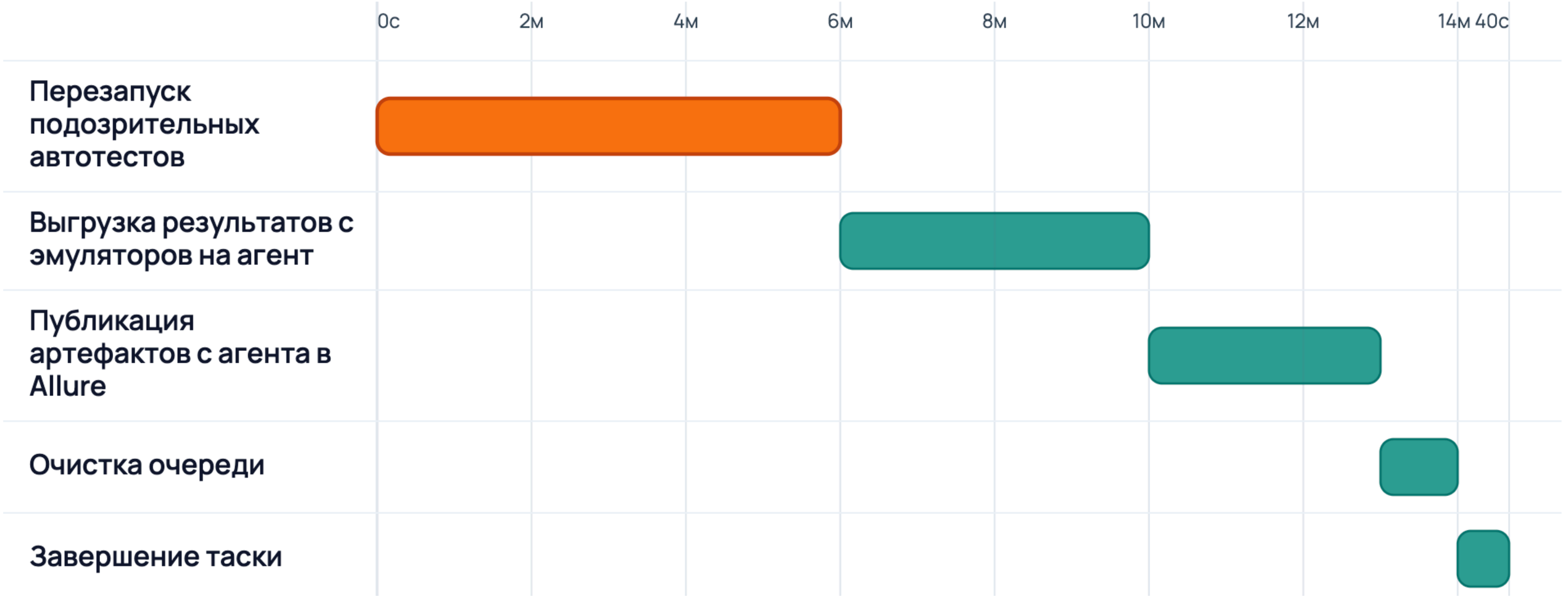
Разбираем этапы прогона



Разбираем этапы прогона



Разбираем этапы прогона



Проблемы, которые можно увидеть

- Если какой-то этап явно больше своих соседей, то стоит подумать над его ускорением
- Самым длинным этапом должен быть прогон тестов, остальные должны быть значительно меньше

Что даёт реальный прирост скорости

- Импакт анализ на основе использования кода
- Экономия на подготовке тестовых данных и окружения
- А также оптимизации сети, хранения, сборки приложения

Ограничения и точки роста

- Узкий сетевой канал
- Слабая серверная хост-машина
- HDD вместо SSD-дисков

Уменьшаем количество прогоняемых тестов

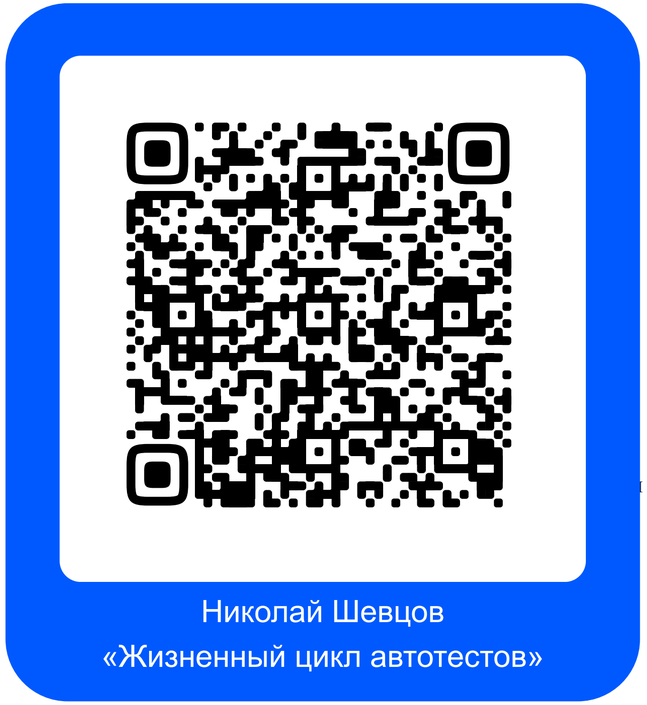
Строим impact анализ

На основе ручной разметки и тест-кейсов

- Анализируем все разделы приложения
- Выделяем список тегов по модулям и фичам в приложении
- Задаем каждому тест-кейсу релевантный тег
- Если TMS нет, то объединяем через сьюты

Стоит научиться работать с флаки

- Автоматический вывод в карантин с предисловиями
- Мониторинг success-рейт теста



Николай Шевцов
«Жизненный цикл автотестов»

Большая команда, и тестов уже несколько тысяч

От 1000 до бесконечности тестов

Как мы считаем крупные команды

Характеристика	Большая команда
Устройства	K8s + динамический шардинг
CI/CD	Service Catalog / оркестратор
Данные	Контейнеризация БД / моки
Аналитика	Метрики (SLA, Cost, Flakiness) + ML
Импакт-анализ	ML / предиктивный
Время прогона, мин	15 и выше
Число автотестов	Больше 1000

Как обычно выглядит этот этап

- Множество команд, микросервисная архитектура, фича-флаги
- Проблемы: время прогона - часы, стоимость инфраструктуры, зависимость команд от одного ресурса
- Цель: открытость инфраструктуры, наблюдаемость, экономическая эффективность

Типичные ошибки

- Единая монолитная очередь прогонов для всех команд

Типичные ошибки

- Единая монолитная очередь прогонов для всех команд
- Игнорирование стоимости тестов

Типичные ошибки

- Единая монолитная очередь прогонов для всех команд
- Игнорирование стоимости тестов
- Отсутствие возможности самостоятельно написать пайплайн

Разметка на основе покрытия ветвлений

- Прогоняем все тесты с включенным CodeCoverage или аналогом JaCoCo

```
[CI] Running full test suite with coverage...

> Task :test

LoginTest PASSED
PaymentTest PASSED
OrderTest PASSED
FeatureFlagTest PASSED

Tests: 124 total, 0 failed

> Task :jacocoTestReport

[JaCoCo] Coverage summary
Lines:    78%
Branches: 69%
Classes:  85%

[JaCoCo] Execution data collected for 124 tests

BUILD SUCCESSFUL in 6m 12s
```

А что если у меня нет таких средств под рукой

- Можно покрыть каждое ветвление в коде специальным логом и собрать маппинг на его основе

```
[CI] Running tests...

LoginTest PASSED
PaymentTest FAILED
FeatureFlagTest PASSED
OrderTest PASSED

Tests: 4 total, 1 failed

[JaCoCo] Coverage
Lines: 73%   Branches: 66%

[Branch mapping]
PaymentService: branch FALSE NOT covered
FeatureFlag: enabled/disabled covered

[Impact Analysis]
Changed: PaymentService, FeatureFlagManager
Affected tests: PaymentTest, FeatureFlagTest

BUILD FAILED in 3m 42s
```

Выделение критичных смоук-сценариев

Нужно выделить основные пользовательские сценарии:

- Исходя из бизнес-метрик
- Пообщавшись с менеджерами насчет клиентской воронки

Настраиваем мониторинг, чтобы было хорошо

Что посоветует ИИ

Ключевые метрики

- Pass rate — процент успешных прогонов (тренд)
- Flakiness — доля нестабильных тестов (падают/проходят случайно)
- Время выполнения — общее и по отдельным тестам (медленные тесты)
- Количество тестов — всего, упавших, пропущенных
- MTTR — время восстановления стабильности после падения
- Потребление ресурсов (CPU, память) — если критично

Что посоветует ИИ

Ключевые метрики

- Pass rate — процент успешных прогонов (тренд)
- Flakiness — доля нестабильных тестов (падают/проходят случайно)
- Время выполнения — общее и по отдельным тестам (медленные тесты)
- Количество тестов — всего, упавших, пропущенных
- MTTR — время восстановления стабильности после падения
- Потребление ресурсов (CPU, память) — если критично

Также не забываем мониторить состояние дежурного и боли коллег через точки входа, обозначенные ранее

Откуда собрать метрики

- С хост-машины, где они запускаются
  - CPU и RAM
  - Утилизация сети
  - Утилизация диска

```
[CI] Running full test suite on host: test_cloud_AR_a9f3

[HOST METRICS]

CPU (Intel Xeon Platinum 8360Y):
usage: 76% (user 61%, system 15%)
load avg: 27.3 / 23.9 / 18.2

RAM (4×32GB Samsung M393A4K40DB3-CWE DDR4 ECC):
used: 97 GB / 128 GB (76%)
swap: 0.6 GB used

DISK (Intel SSD D7-P5510 NVMe 3.84TB):
read: 470 MB/s
write: 210 MB/s
iowait: 4%

NETWORK (Intel X550-T2 10GbE, limited to 1Gbps, full-duplex):
in: 118 MB/s (~0.94 Gbps)
out: 92 MB/s (~0.74 Gbps)
latency: 3 ms

[TESTS]
running: 820 tests on 10 simulators...
duration: 15m 22s
```

Откуда собрать метрики

- Логи из раннера и автотестов
  - Временные рамки прогона и его этапов
  - Ошибки, происходящие в тестах

```
test.exceptions.WrongTestConditionException: TestDataServer returned exception:
ociated with hostname, cause: null
  at test.testdata.testdataclient.TestDataClient.acquireUser(TestDataClient.kt:
  at test.testdata.testdataclient.TestDataClient.acquireUser$default(TestDataCl
```

О чем могут сказать ошибки при прогоне тестов

- О проблемах с инфраструктурой
- О проблемах с приложением
- О проблемах с тестами

О чем могут сказать ошибки при прогоне тестов

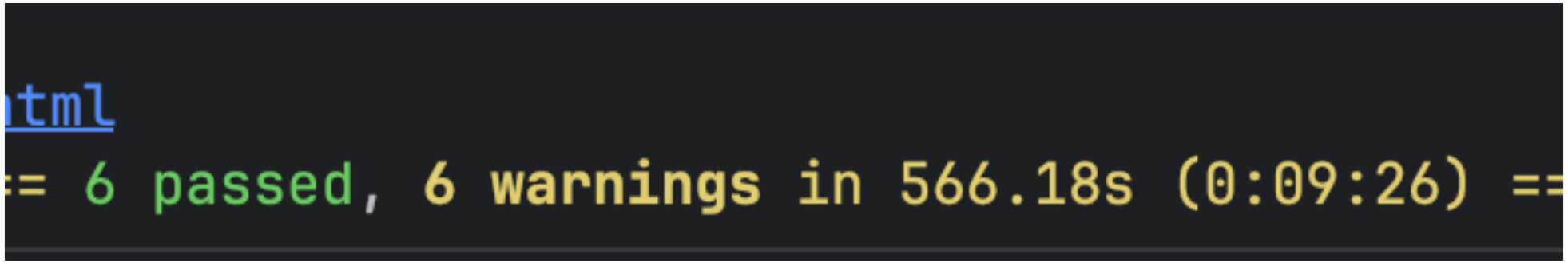
- О проблемах с инфраструктурой
- О проблемах с приложением
- О проблемах с тестами

О чем могут сказать ошибки при прогоне тестов

- О проблемах с инфраструктурой
- О проблемах с приложением
- О проблемах с тестами

Откуда собрать метрики

- Пропарсить отчет о прогоне
  - Временные рамки прогона и его этапов
  - Количество тестов по статусам



Что измерять не стоит?

- Прогоны, не связанные с релизом
- Лучше сравнивать один и тот же пайплайн во времени

Что измерять не стоит?

- Среднее время пайплайна
- Лучше измерять P95 или P99

Метрика	Неделя 1	Неделя 2
Среднее	9,7	10,2
Медиана	9,3	9,5
P95	**11,3**	**16,8**
Max	12,2	18,5

О чем говорит резкое изменение этих метрик?

- В инфраструктуре произошло изменение
- Чем точнее метрика, тем легче найти причину проблемы
- Максимальная нагрузка на сервер не всегда хорошо

«Проценты использования процессора — это ложная метрика»

Что еще можно сделать с данными из мониторинга

- Понять, был ли всплеск ошибок локальным и связан с MR
- Найти части инфраструктуры и приложения, к которым нужно внимание

А еще можно попросить ИИ

- Для выявления нестандартных проблем целесобразно использовать ИИ и анализировать уже его репорт

* Особенно с большим объемом данных

Тесты на перформанс

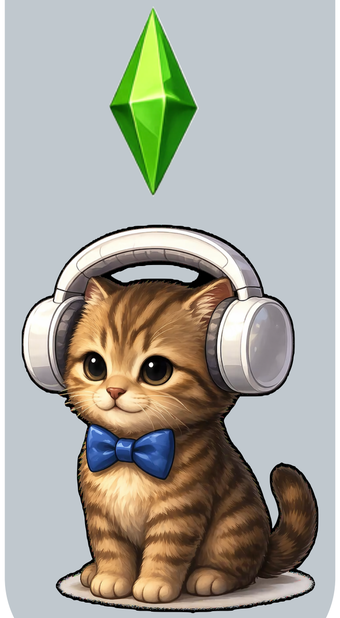
- Запускать на каждый MR и измерять дифф между ним и мастером
- Регулярно прогонять мастер, чтобы отслеживать изменения на бэкенде
- Запускать собственное приложение и приложения конкурентов, чтобы понимать, в чём разница

Михаил Шваркунов
«Как мы ВКонтакте измеряем перфоманс наших приложений»

Проводите регулярные учения

- Разработайте сценарии и периодически их проигрывайте, не предупреждая, какой сценарий будет
- Также стоит регулярно проводить нагрузочное тестирование инфраструктуры автотестов

«Проектирование отказоустойчивости IT-систем»



Подвёдем итоги

Ошибки, которые важно не допускать на всех этапах

- Отсутствие изоляции данных

Ошибки, которые важно не допускать на всех этапах

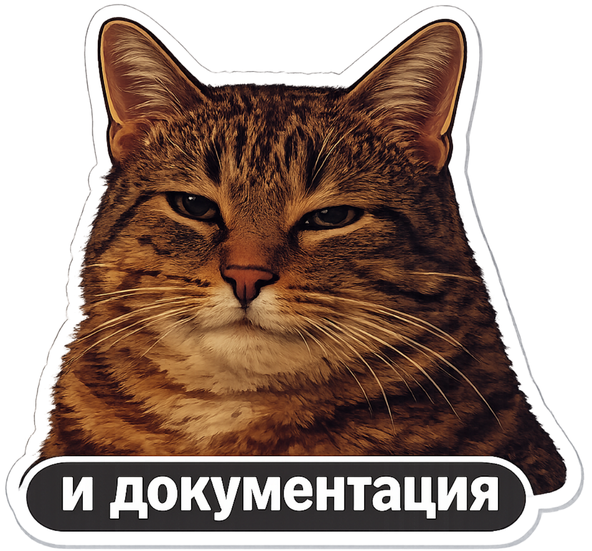
- Отсутствие мониторинга и низкое observability

Ошибки, которые важно не допускать на всех этапах

- Несоответствие сложности масштабу

Как понять, что все хорошо

- Разработчику не нужно думать где и что запустить, все запустится «само»
- DevOps может отключить сломанную часть без влияния на весь кластер
- Накопительная аналитика видит, какие тесты флакают чаще всего, и предлагает по ним решения



Как понять, что все хорошо

- Разработчику не нужно думать где и что запустить, все запустится «само»
- DevOps может отключить сломанную часть без влияния на весь кластер
- Накопительная аналитика видит, какие тесты флакают чаще всего, и предлагает по ним решения
- Результат: пайплайн зелёный в любое время дня и ночи*

* Кроме инцидентов


ВСЁ!