Код-ревью как «бутылочное горлышко»

Почему код готов, а в проде его нет?

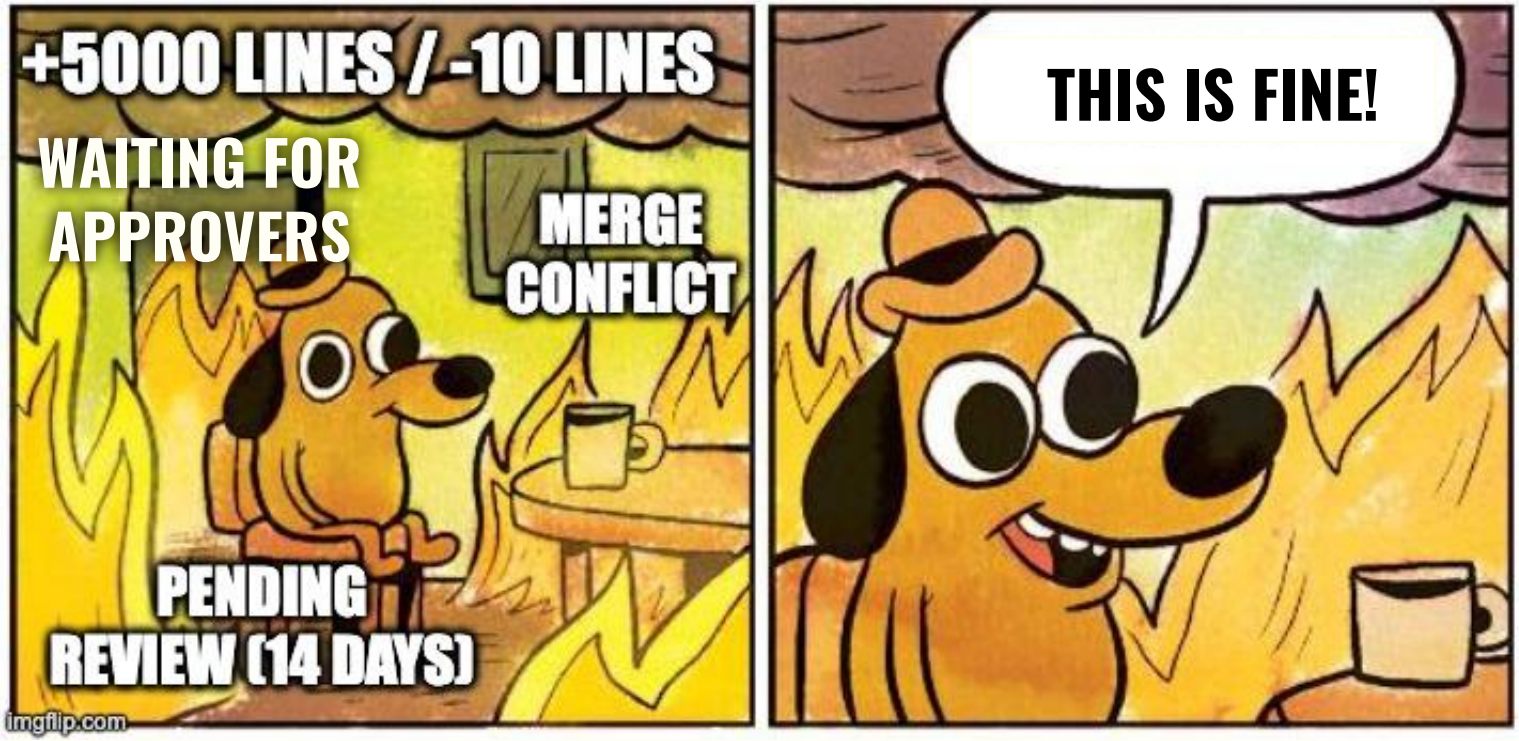
Дмитрий Михальченков

Senior Android Engineer, inDrive

- 15+ лет в IT, из них 8+ в Android-разработке
- Нативная и кроссплатформенная разработка
- Масштаб: Работал в командах от 3 до 50+ человек.
- Проекты: от healthcare и delivery до ride-hailing с многомиллионной аудиторией
- Автор технического блога на Medium

@mikhaltchenkov
mikhaltchenkov@gmail.com

Вы часто ждете ревью больше 2 дней?

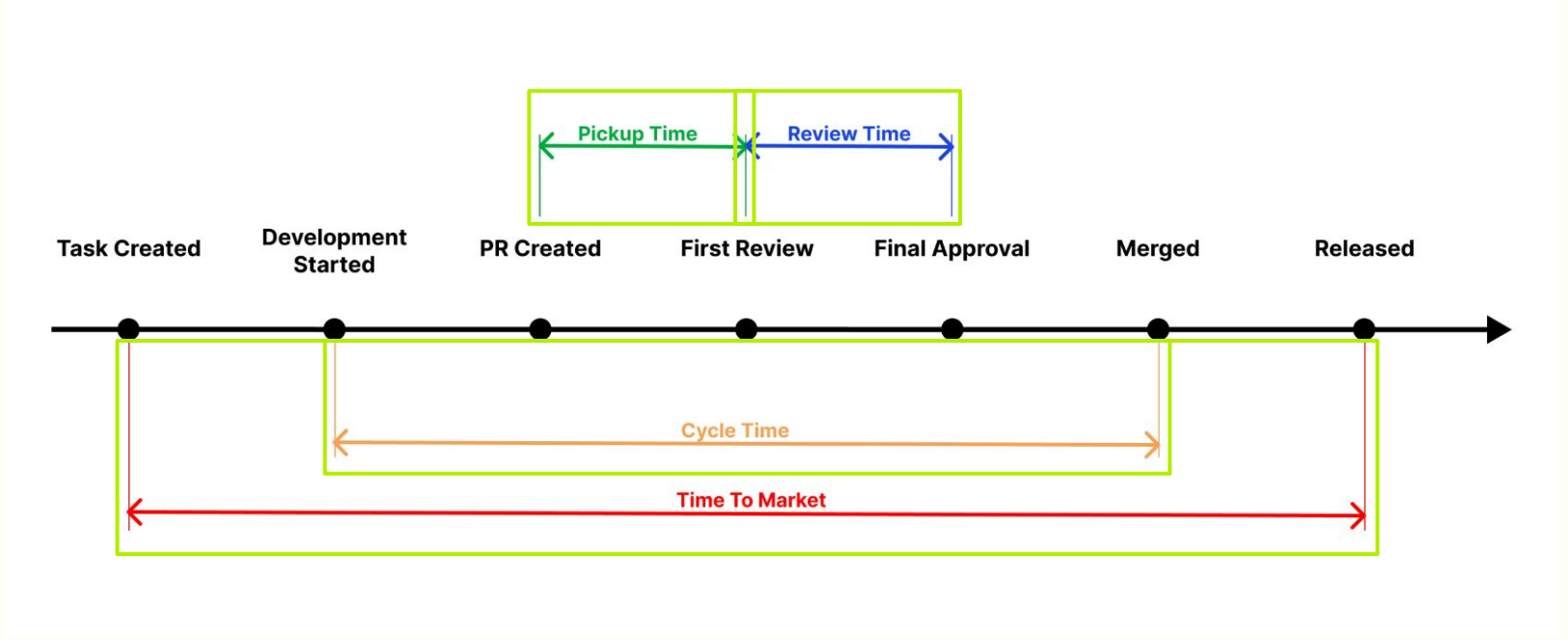


Как ревью влияет на Time To Market

Планирование → Разработка → Ревью → Тестирование → Раскатка

Вопрос не в том, как быстро вы пишете код. Вопрос — как быстро код попадёт в прод

Измеряем здоровье процесса ревью



Time To Market = Cycle Time + планирование + тестирование + раскатка

Что сильнее влияет на скорость ревью?

Желание ревьюить код

Количество строк в PR

My code review on a
PR with 10 lines of code

"275 suggestions"

My code review on a
PR with 1000 lines of code

"LGTM"

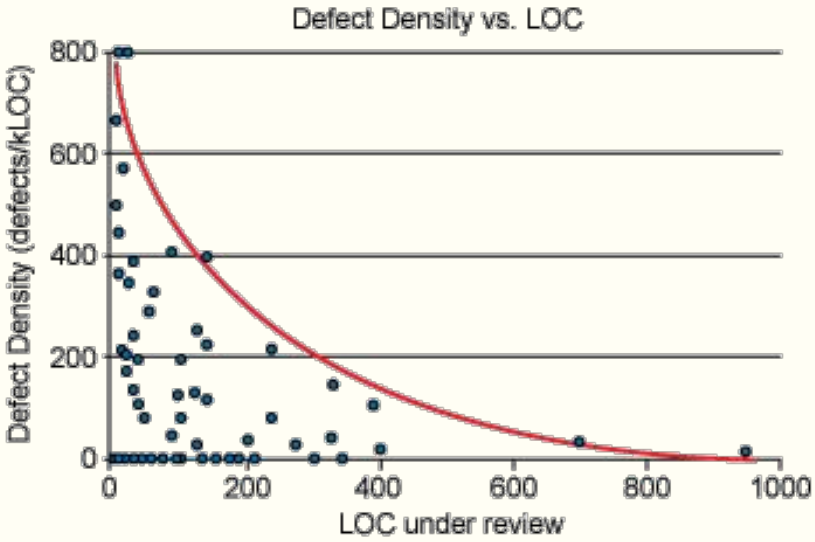
Исследования SmartBear / Cisco

Магия чисел

Исследование SmartBear / Cisco

- **до 200–400 строк (LOC) за сессию** ⇒ пик обнаружения дефектов: 70–90%.
- **После 400 строк** ⇒ внимание падает, дефекты проходят незамеченными.
- **Оптимальная сессия:** 60–90 минут

Это не лень. Это биология.



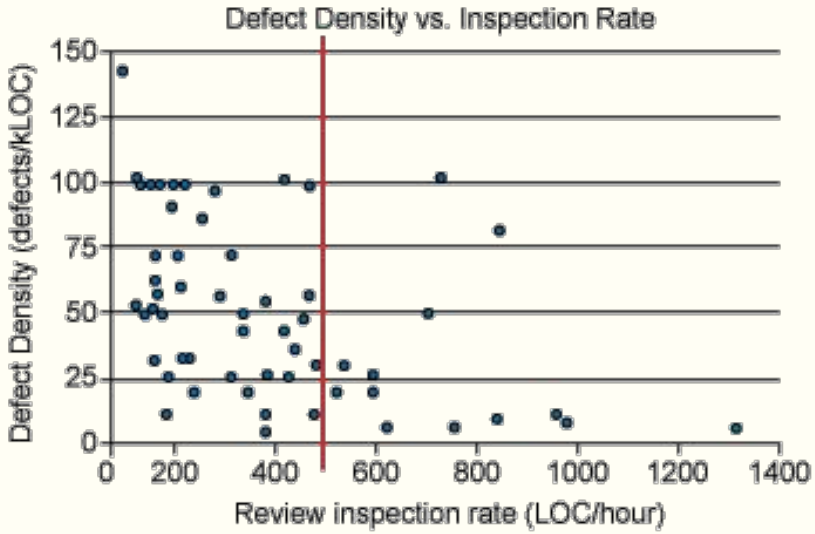
Ловушка «Гонщика»

Лимит скорости

- **Критический порог:** 500 строк кода (LOC) в час
- **Если быстрее** ⇒ количество найденных дефектов стремительно падает

Формула идеального ревью

- Разумный объем изменений
- Медленный, вдумчивый темп
- Ограниченное время сессии



LinearB Software Engineering Benchmarks Report

Реальный мир: данные LinearB

LinearB 2026 Software Engineering Benchmarks Report - 8.1 млн PR, 4 800 команд, 42 страны

Team Level	PR size, lines	Cycle Time, hours
Elite	<100	< 25
Good	100–155	25–72
Fair	156–228	72–161
Needs focus	>228	>161

- Метрика P75: 75% PR команды укладываются в этот размер
- Elite = верхние 25% команд из 4 800 в выборке
- Needs focus = нижние 25%

Три эффекта Code Review

Долгое ожидание (Pickup time)

Большой diff чаще “лежит” без ревью - сложнее взять в работу

	P75 Size	Pickup Time
Обычный PR	157	~3.3 ч
AI-assisted PR	408	~8.3 ч
Agentic AI PR	293	~17.6 ч

Три эффекта Code Review

Поверхностное ревью

Крупные PR чаще смотрят поверхностно

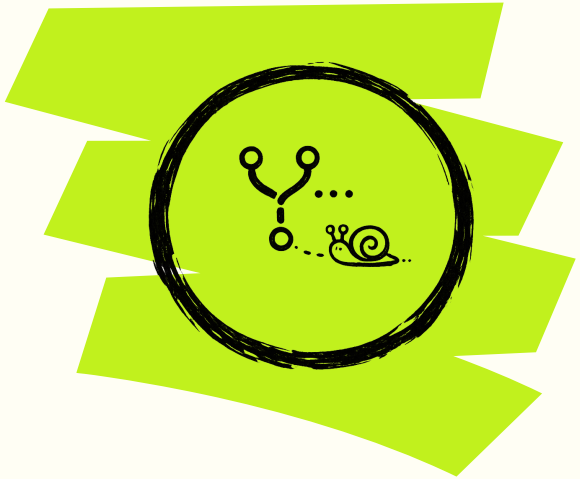
> reviewers may be scanning for high-level issues rather than deeply evaluating the full changeset

Три эффекта Code Review

Медленный мёрж (Cycle Time)

Больше кода в PR ≠ больше ценности в проде

- Elite (PR < 100 строк) → < 25 часов
- Needs focus (PR > 228) → > 161 часа
- Больше размер PR → Больше request changes



Материалы

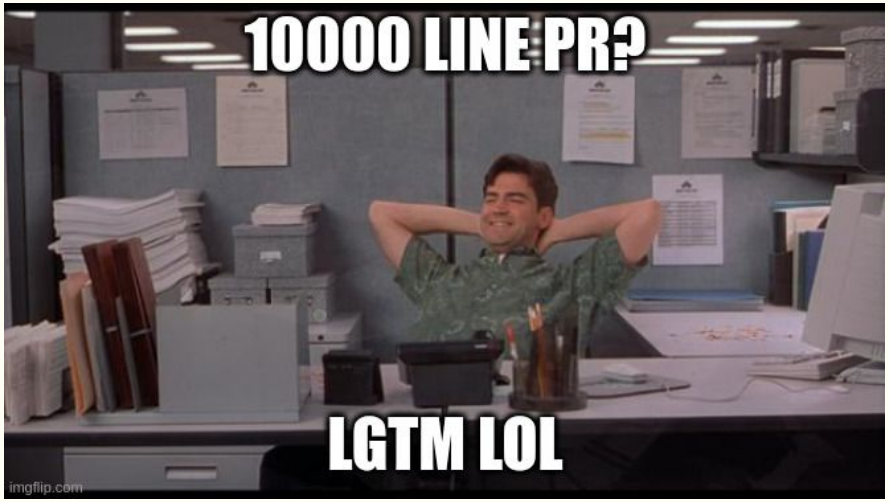
SmartBear: Best Practices for Code Review

LinearB Software Engineering Benchmarks Report

Борьба с PR-монстрами

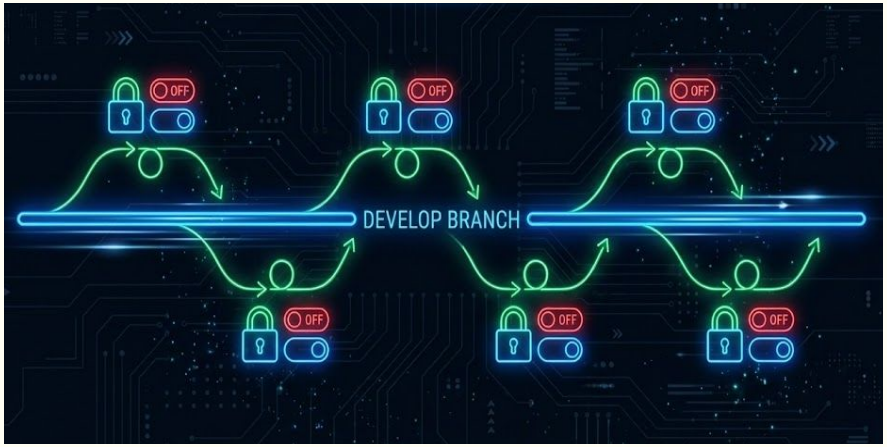
Скрытые издержки больших PR

- Экспоненциальный рост времени ревью
- Устаревание кода и Merge-конфликты
- Снижение мотивации
- Эффект LGTM и упущенные баги



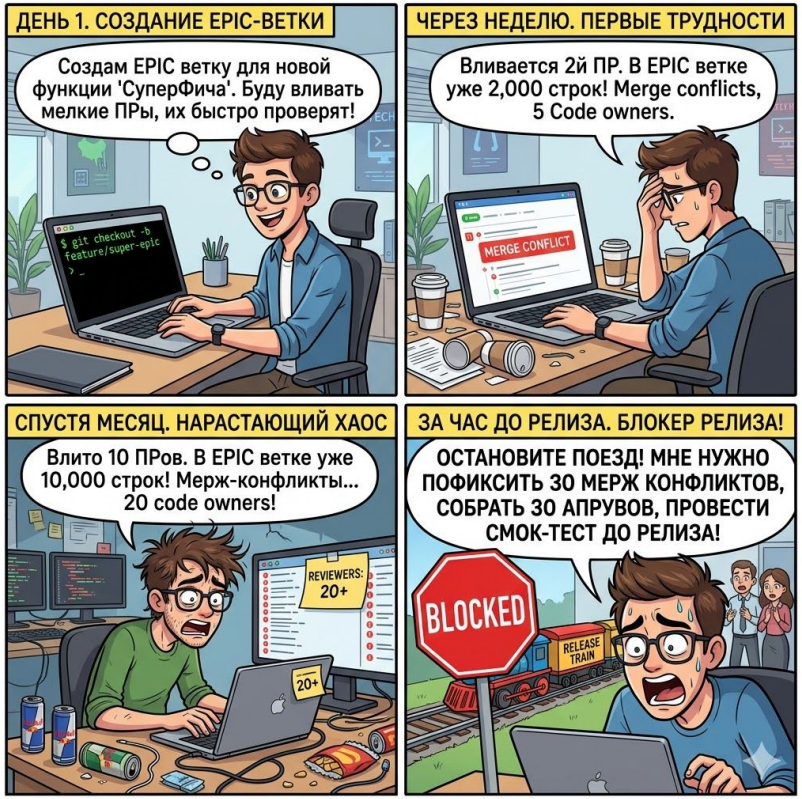
Дробим слона: Trunk Based Development

- Срок жизни ветки < 3 дней
- Атомарные PR < = 500 строк
- Безопасность: Feature Toggles + изоляция слоёв.
- Деплой кода != Релиз фичи



Ловушка Epic-веток

- **Иллюзия скорости:** быстрые ревью внутри фичи
- **Проблема:** Epic-ветка = изоляция. За неделю dev уходит далеко вперед.
- **Результат:** Адские конфликты при слиянии (Merge Hell) и устаревание кода.



Почему PR стали расти?

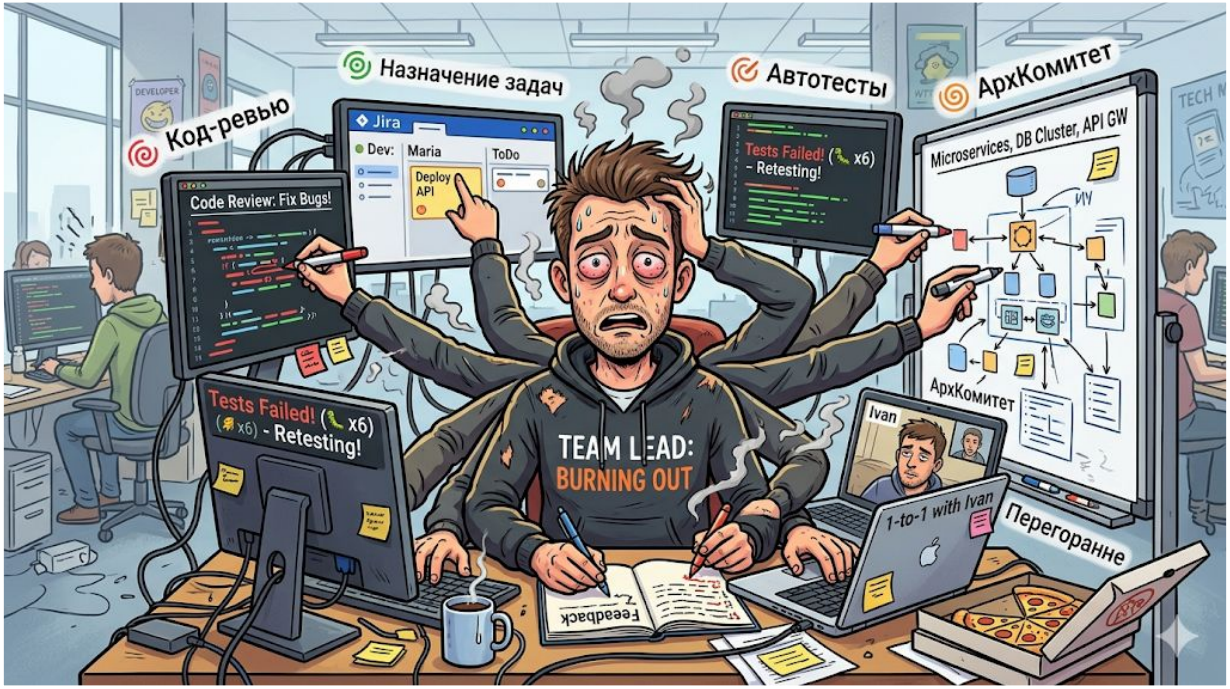
- **«Щедрость» без фильтра:** Избыточные KDoc, логи и инлайн-комментарии
- **Изоляция от контекста (One-shot сессии):** Скопированные из чата функции часто несут чужеродную архитектуру и дублируют утилиты, которые уже есть в проекте
- **Отсутствие рамок:** В автодополнении ИИ не заботится о лаконичности, раздувая дифф лишними строками
- **Последствия:** Такие PR ждут первой реакции дольше

Укрощение «агентов»

- **Системный правила (AGENTS.md):** Базовые инструкции на уровне репозитория
- **Жесткий фильтр «шума»:** Прямой запрет на генерацию избыточных комментариев и чужеродных паттернов.
- **Синхронизация с проектом:** ИИ должен знать ваши ADR, Style Guide, эталонные примеры тестов.

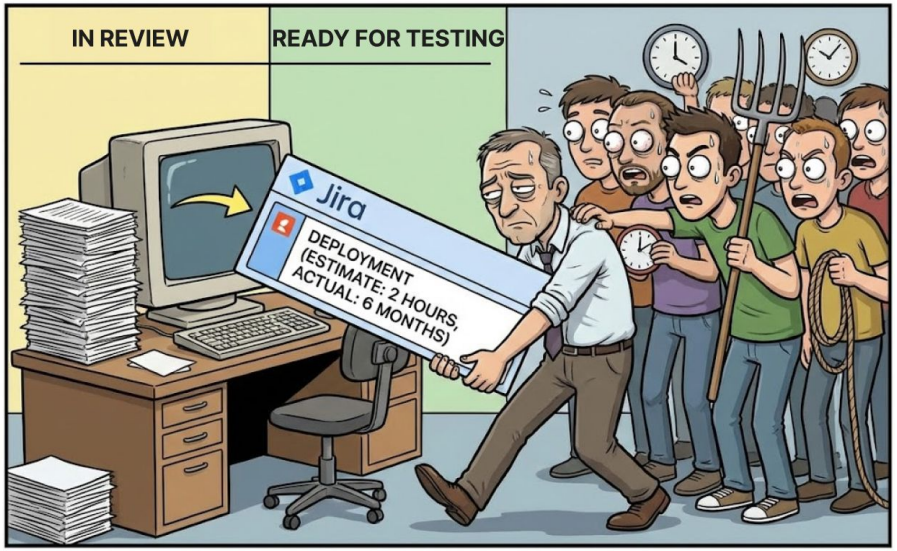
Bottleneck в процессах

Bus Factor: Ловушка централизации



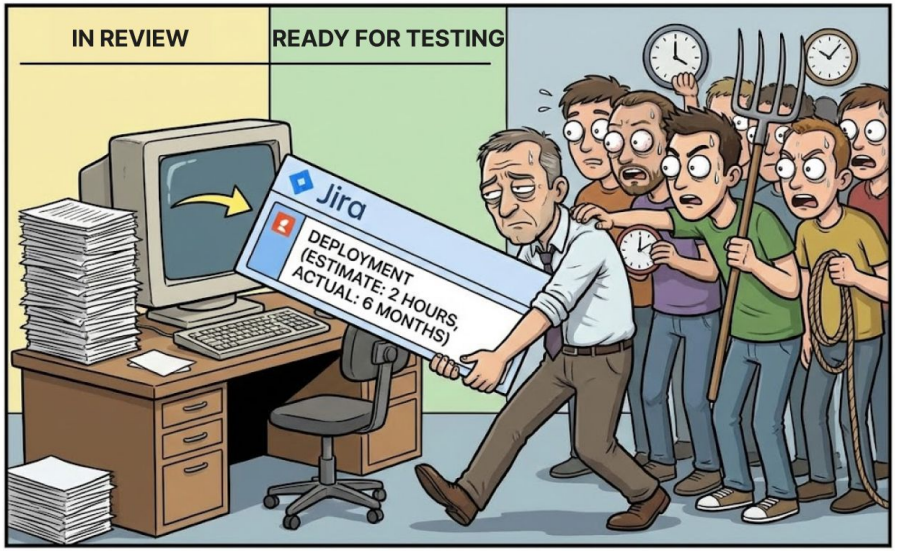
Bus Factor: Ловушка централизации

Действие
Апрув PR
Перевод таски "In Review" → "Ready for Test"
Перевод "Tested" → "Done"
Мерж в develop



Bus Factor: Ловушка централизации

Действие	Время ожидания
Апрув PR	1 д.
Перевод таски "In Review" → "Ready for QA"	0.5 - 1 д.
Перевод "Tested" → "Done"	1 д.
Мерж в develop	1-2 д.

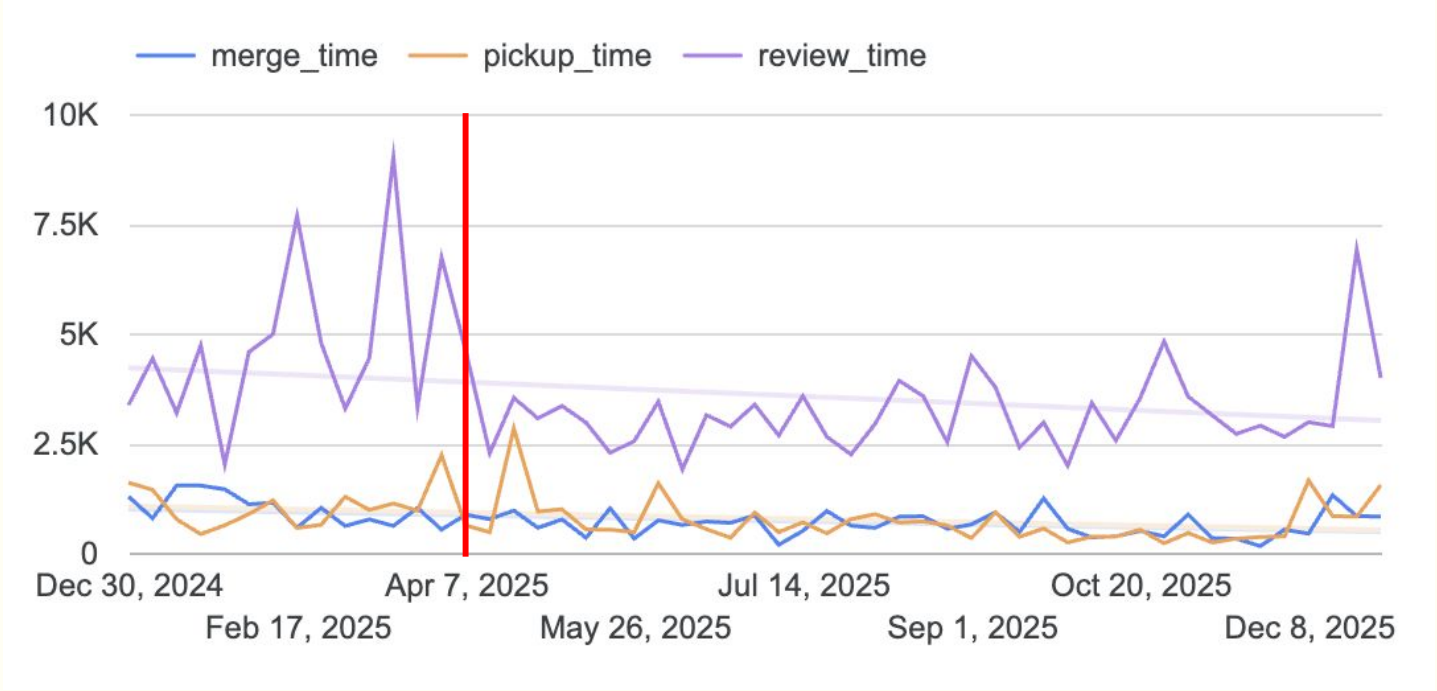


От личного контроля к автоматизации

- **Jira Automation + CI/CD Pipeline:** Смена статусов задачи при изменении статуса ПР
- **Definition of Done (DoD):** Чек-лист, который регламентирует порядок в процессах
- **Feature Toggles:** Разделяем деплой кода и доступность фичи

Наблюдения внутри inDrive

Android команда



Геймификация

- **Проблема:**

  Ревью - это "невидимая работа", которую часто не ценят.

- **Решение:**

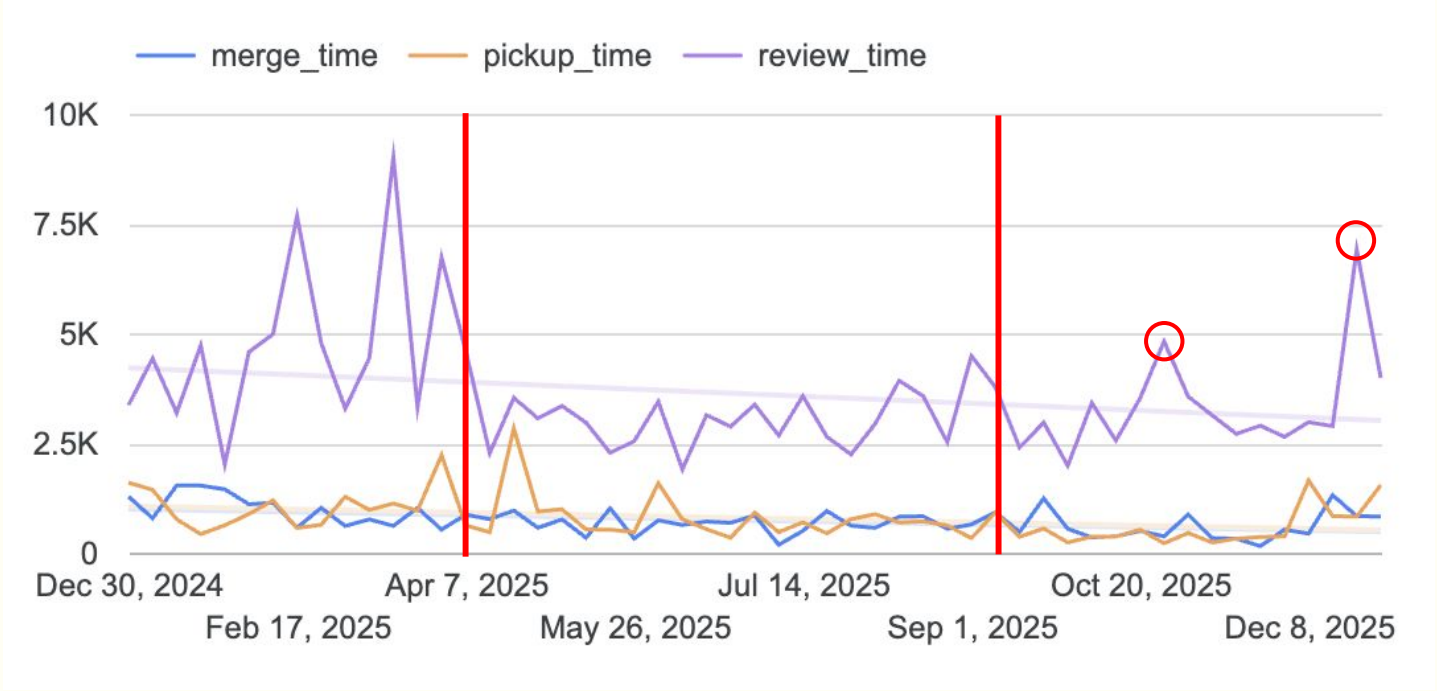
  Лидерборд (обновляется раз в неделю).

  - *Approve → 1 балл*
  - *Comment → 2 балла*
  - *Request Changes → 3 балла*

- **Награда:**

  Мерч, ачивки в профиле, респект на ретро, визибилити в компании.

Android команда



Автоматизация: ИИ-агент как первый ревьюер

- **AI Summary:**

  Генерирует краткое описание изменений - экономит время на погружение.

- **Первичный фильтр:**

  Потенциальные баги, NPE, неиспользуемые переменные, утечки памяти

  Нейминг, нарушения SOLID, неконсистентный стиль

  Соответствие ADR и стайлгайдам проекта

- **Проверка контекста:**

  Сверяет изменения с описанием задачи в Jira

  Отмечает «лишние» изменения, не связанные с задачей

Автоматизация: ИИ-агент как первый ревьюер

- **AI Summary:**

  Генерирует краткое описание изменений - экономит время на погружение.

- **Первичный фильтр:**

  Потенциальные баги, NPE, неиспользуемые переменные, утечки памяти

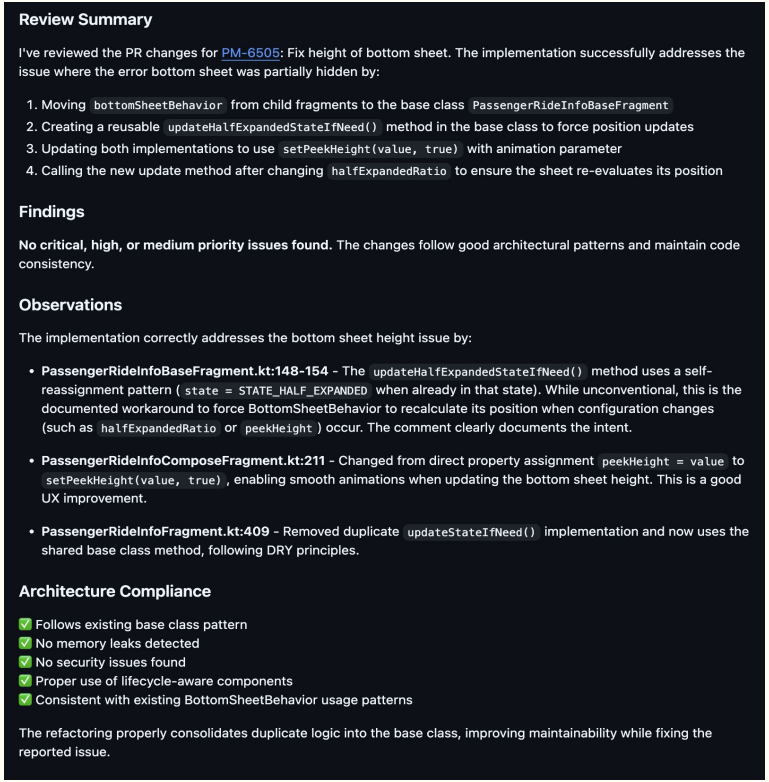
  Нейминг, нарушения SOLID, неконсистентный стиль

  Соответствие ADR и стайлгайдам проекта

- **Проверка контекста:**

  Сверяет изменения с описанием задачи в Jira

  Отмечает «лишние» изменения, не связанные с задачей



Review Summary

I've reviewed the PR changes for PM-6505: Fix height of bottom sheet. The implementation successfully addresses the issue where the error bottom sheet was partially hidden by:

1. Moving `bottomSheetBehavior` from child fragments to the base class `PassengerRideInfoBaseFragment`
2. Creating a reusable `updateHalfExpandedStateIfNeed()` method in the base class to force position updates
3. Updating both implementations to use `setPeekHeight(value, true)` with animation parameter
4. Calling the new update method after changing `halfExpandedRatio` to ensure the sheet re-evaluates its position

Findings

No critical, high, or medium priority issues found. The changes follow good architectural patterns and maintain code consistency.

Observations

The implementation correctly addresses the bottom sheet height issue by:

- **PassengerRideInfoBaseFragment.kt:148-154** - The `updateHalfExpandedStateIfNeed()` method uses a self-reassignment pattern (`state = STATE_HALF_EXPANDED` when already in that state). While unconventional, this is the documented workaround to force BottomSheetBehavior to recalculate its position when configuration changes (such as `halfExpandedRatio` or `peekHeight`) occur. The comment clearly documents the intent.
- **PassengerRideInfoComposeFragment.kt:211** - Changed from direct property assignment `peekHeight = value` to `setPeekHeight(value, true)` , enabling smooth animations when updating the bottom sheet height. This is a good UX improvement.
- **PassengerRideInfoFragment.kt:409** - Removed duplicate `updateStateIfNeed()` implementation and now uses the shared base class method, following DRY principles.

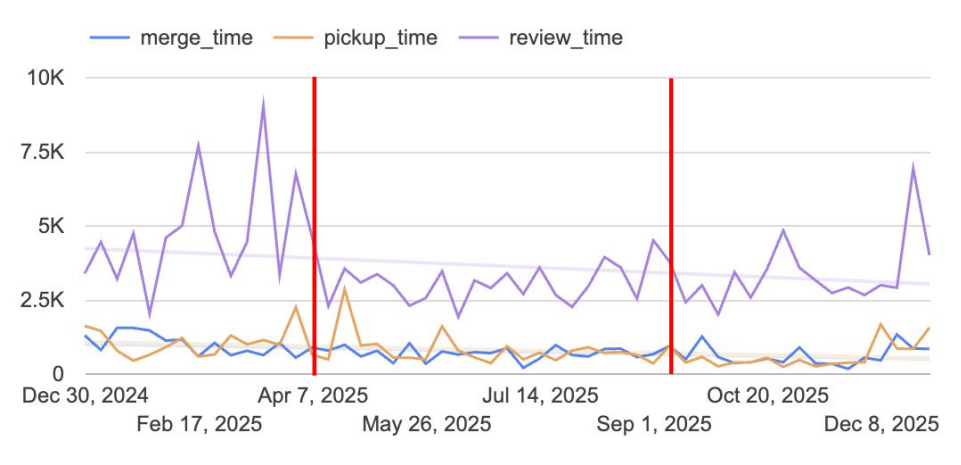
Architecture Compliance

✅ Follows existing base class pattern
✅ No memory leaks detected
✅ No security issues found
✅ Proper use of lifecycle-aware components
✅ Consistent with existing BottomSheetBehavior usage patterns

The refactoring properly consolidates duplicate logic into the base class, improving maintainability while fixing the reported issue.

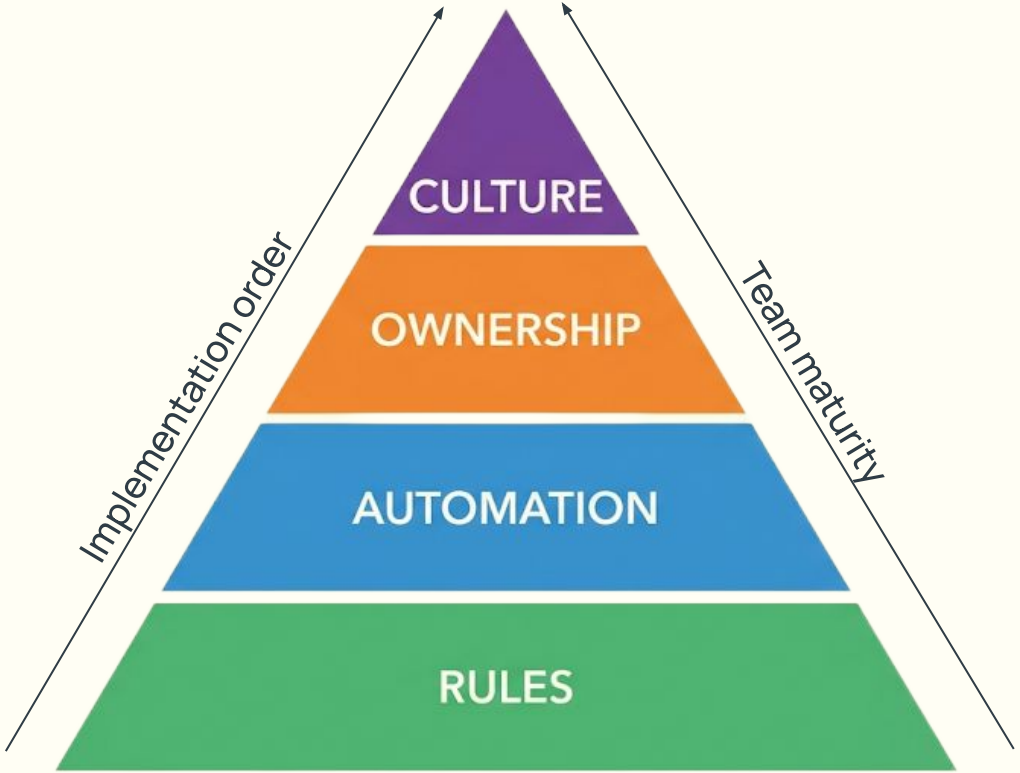
Два эксперимента - выводы

- Геймификация: Review Time / 3(Android), но эффект угасает за 4 месяца
- ИИ-ревьюер: устойчивое снижение у обеих платформ

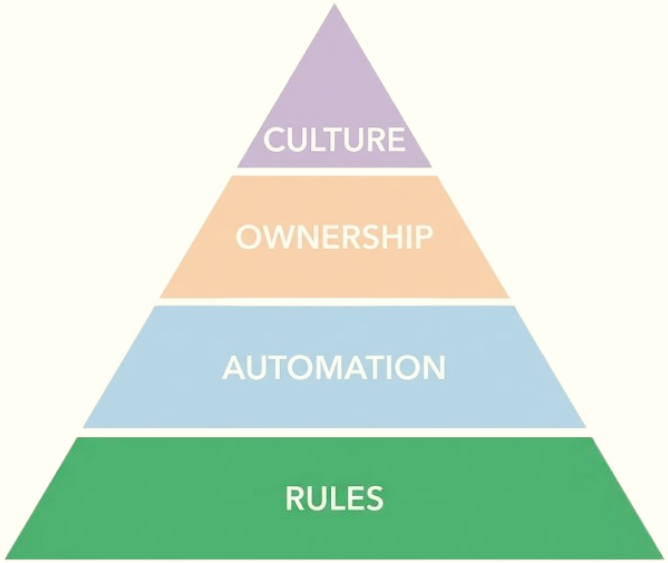


Способы улучшения процесса ревью

Способы улучшения процесса ревью

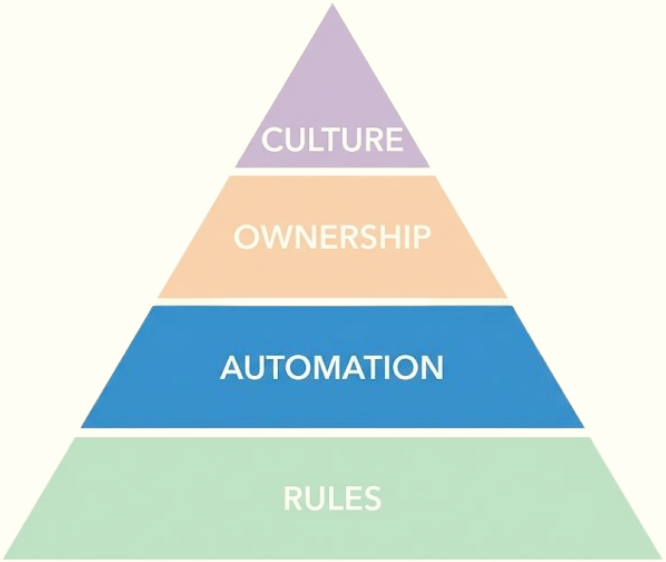


Уровень 1: Правила



- Code Styles & Hygiene
- Architecture Decision Records
- PR size limitation
- Review SLA
- Code Coverage
- Definition of Done

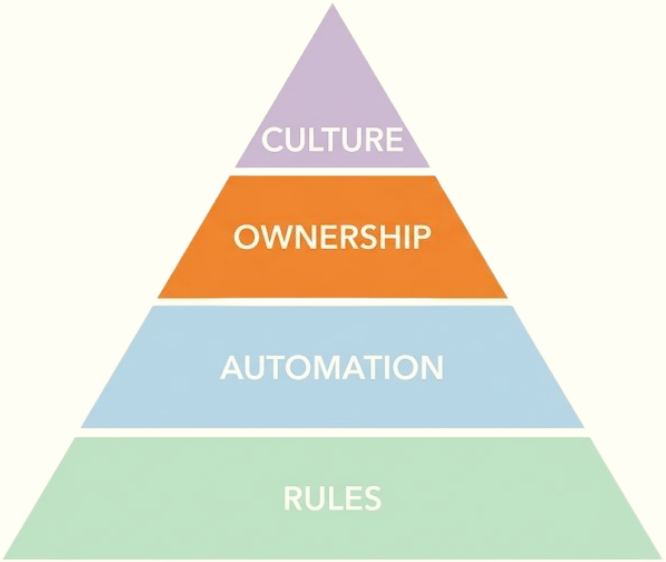
Уровень 2: Автоматизация



- Pre-commit/pre-push Hooks
- Static Analysis & Formatting
- Quality Gates: Unit/UI/Screenshot Tests
- PR-Meta Checks (Danger Bots)
- Pipeline Speed & Stability
- Ai Reviewers

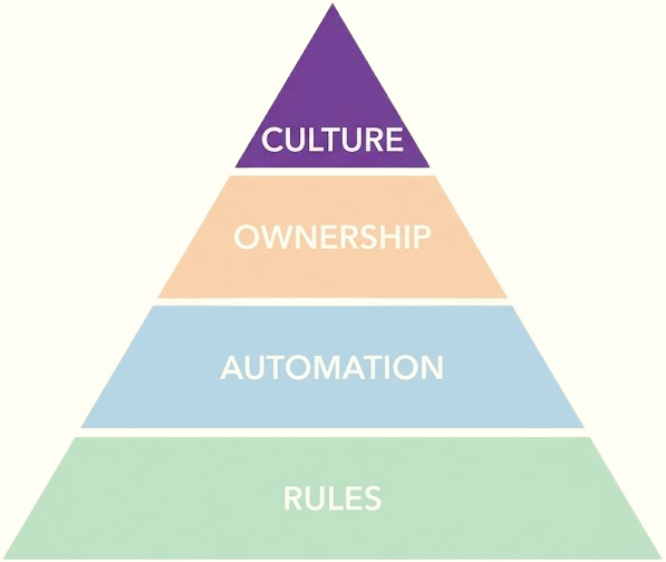
Если ревьюер пишет "поправь отступ" — ваша автоматизация провалилась

Уровень 3: Ответственность



- Code Owners
- Super-approvers
- Review Roulette / Load Balancing
- Author Ownership

Уровень 4: Культура



- Visibility & Recognition
- Gamification
- Blameless Culture
- Mentorship & Knowledge Sharing
- Constructive Feedback

Выводы

- **Метрики > Ощущения:** Следите за Pickup и Cycle Time
- **Автоматизация — это не только Lint:** Делегируйте ИИ рутину и сбор контекста для человека.
- **Культура и Процессы:** Стройте процесс ревью снизу вверх (правила → боты → люди), защищая команду от выгорания
- **Мёрж - это единственная цель:** Идеальный код в ветке не имеет ценности.

Спасибо за внимание. Вопросы?

Telegram

LinkedIn

Medium