



## Немного о Code Review

Борисова Юлия  
м.н.с. ЛабКИИ

# Что такое code review?

***Проверка кода** - один или несколько разработчиков смотрят код другого, обсуждают, предлагают изменения, прежде чем влить новый код в основную кодовую базу.*

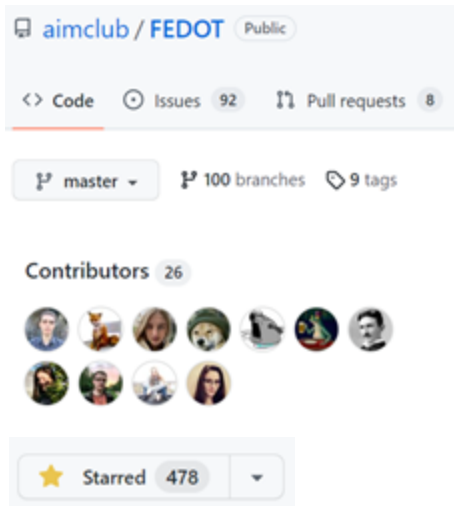
## База:

Проект/ разработка/ систематически  
дополняющийся код

## Участники:

- Автор изменений
- Проверяющие (ревьюеры)

Хороший пример для подражания



# Зачем это нужно?



## Ведь можно просто...

- Вливать изменения прямо в основную ветку;
- Договориться не трогать чужие файлы;
- Хранить изменения в отдельных локальных файлах;
- Каждому писать в свою ветку, а там как-нибудь всем вместе разгребать.

# Зачем это нужно?

## Ведь можно просто...

- Вливать изменения прямо в основную ветку;
- Договориться не трогать чужие файлы;
- Хранить изменения в отдельных локальных файлах;
- Каждому писать в свою ветку, а там как-нибудь всем вместе разгребать.

- Повышать риск простых и архитектурных ошибок;
- Затруднять контроль версий для других коллег.

# Зачем это нужно?

## Ведь можно просто...

- Вливать изменения прямо в основную ветку;
- Договориться не трогать чужие файлы;
- Хранить изменения в отдельных локальных файлах;
- Каждому писать в свою ветку, а там как-нибудь всем вместе разгребать.

- Повышать риск простых и архитектурных ошибок;
- Затруднять контроль версий для других коллег.

**ІТМО**

- Отказаться от унифицированной структуры и ООП;
- Усложнить поддержку кода

# Зачем это нужно?

## Ведь можно просто...

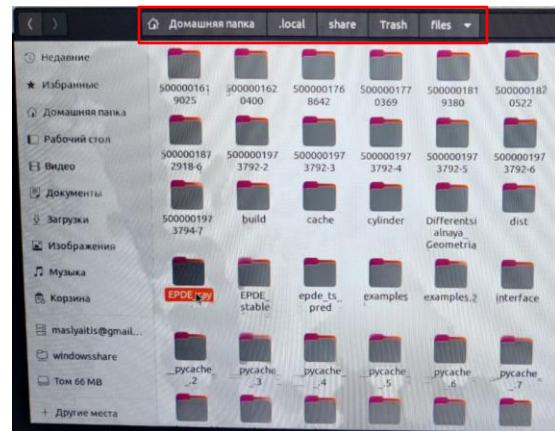
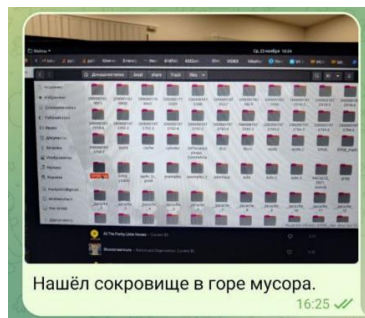
- Вливать изменения прямо в основную ветку;
- Договориться не трогать чужие файлы;
- Хранить изменения в отдельных локальных файлах;
- Каждому писать в свою ветку, а там как-нибудь всем вместе разгрести.

- Повышать риск простых и архитектурных ошибок;
- Затруднять контроль версий для других коллег

# ИТМО

- Отказаться от унифицированной структуры и ООП;
- Усложнить поддержку кода

- Заставить коллег рыться в вашей локальной машине



# Зачем это нужно?

## Ведь можно просто...

- Вливать изменения прямо в основную ветку;
- Договориться не трогать чужие файлы;
- Хранить изменения в отдельных локальных файлах;
- Каждому писать в свою ветку, а там как-нибудь всем вместе разгрести.

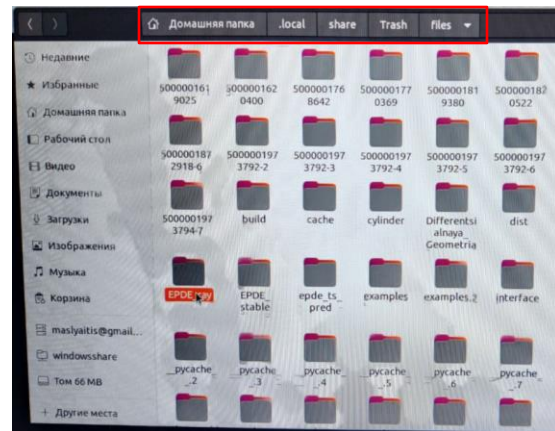
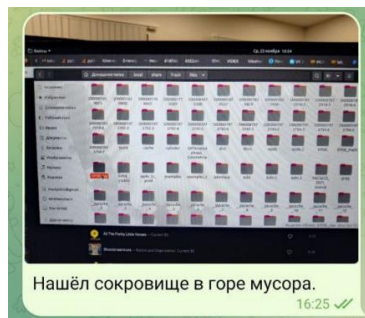
- Потратить кучу времени нескольких человек на совещание об объединении частей кода

- Повышать риск простых и архитектурных ошибок;
- Затруднять контроль версий для других коллег

# ИТМО

- Отказаться от унифицированной структуры и ООП;
- Усложнить поддержку кода

- Заставить коллег рыться в вашей локальной машине



# Правильное хранение кода, Pull Request

## 1. Системы контроля версий



GitHub



GitLab

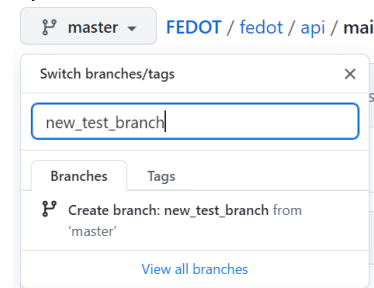
...

## 2. Среда разработки

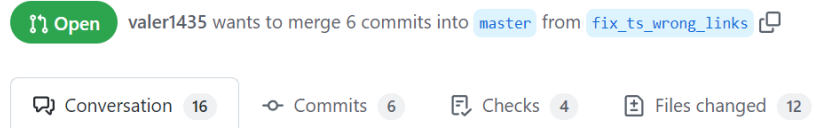
- обладает встроенными инструментами контроля версий (переключение между ветками, коммиты и т.д.);
- PEP8, опечатки

## 3. Работа в отдельной ветке

- Не мешаете другим разработчикам, даже если правите тот же файл
- Подтягиваете изменения из мастера по мере необходимости



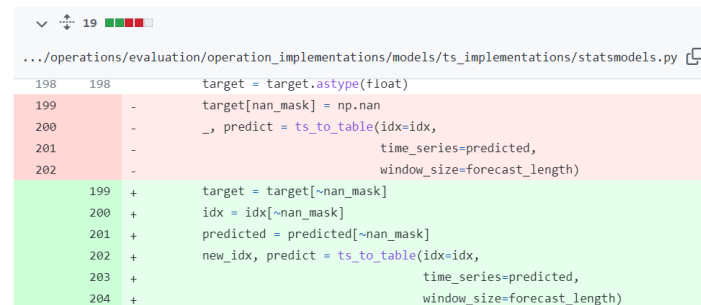
## Fix ts wrong links #994



Оставить вопросы, предложения, рекомендации

Посмотреть историю правок

Успешность прохождения тестов



Внесенные изменения, основная вкладка для ревьюера



# Как реализовано?

Формальная принятая схема\*

## Сложный PR

Крупные изменения,  
вносящие новую  
функциональность или  
правлящие старую

## Простой PR

Небольшие  
изменения, фиксы  
багов и т.п.

2 старших  
разработчика

1 младший  
разработчик

2 младших  
разработчика

1 старший  
разработчик

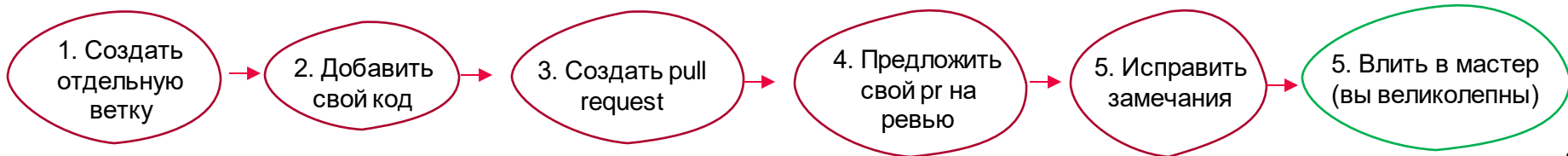
\* на деле на мелких PR'ах бывает по одному ревьюеру, а на глобальных архитектурных решениях больше трех

А судьи кто?

(кому делать ревью в этот раз)

- 1) Авторам сопряженной функциональности
- 2) Заинтересованным в новом блоке
- 3) Тем, кого назначил тимлид
- 4) Тем, у кого есть свободная минутка
- 5) Тем, кого выбрал рандомайзер

## Линейный алгоритм, как влить изменения



# Как реализовано?

Формальная принятая схема\*

## Сложный PR

Крупные изменения,  
вносящие новую  
функциональность или  
правлящие старую

2 старших  
разработчика

1 младший  
разработчик

## Простой PR

Небольшие  
изменения, фиксы  
багов и т.п.

2 младших  
разработчика

1 старший  
разработчик

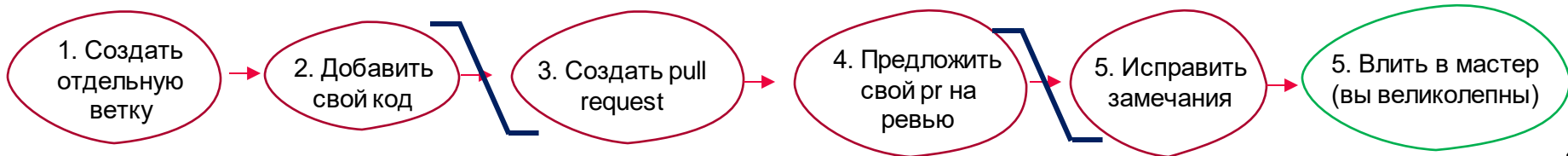
\* на деле на мелких PR'ах бывает по одному ревьюеру, а на глобальных архитектурных решениях больше трех

А судьи кто?

(кому делать ревью в этот раз)

- 1) Авторам сопряженной функциональности
- 2) Заинтересованным в новом блоке
- 3) Тем, кого назначил тимлид
- 4) Тем, у кого есть свободная минутка
- 5) Тем, кого выбрал рандомайзер

**НЕ** Линейный алгоритм, как влить изменения



# Что проверить перед созданием Pull Request'a

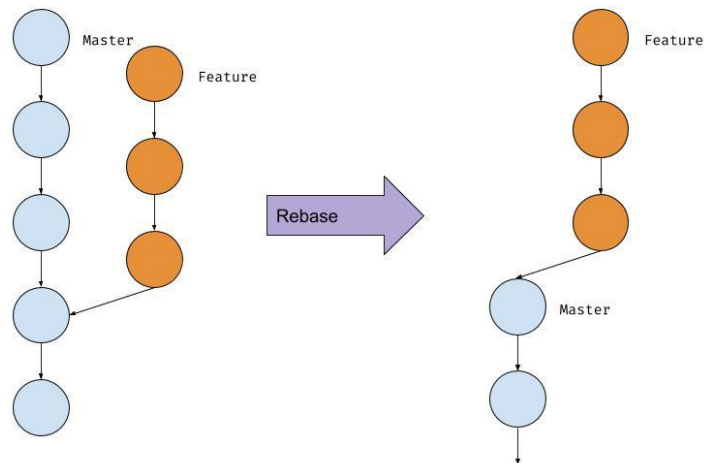
**Простой вариант:** конфликтов с текущей версией основной ветки нет (скорее всего до вас никто не вливал свои изменения в мастер или вы работали в разных файлах)

**Стандартный вариант:** конфликты с текущей версией основной ветки присутствуют – система контроля версий не может автоматически определить, чьи изменения применять.

Необходимо подтянуть новые изменения из основной ветки в свою ветку.

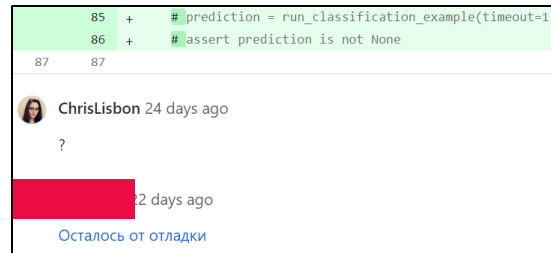
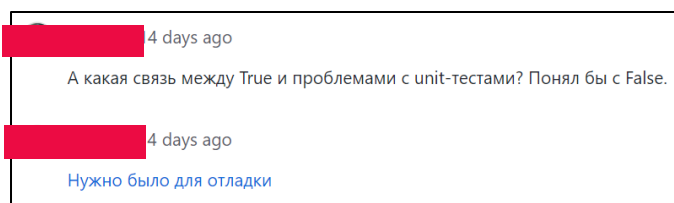
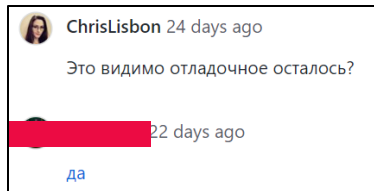
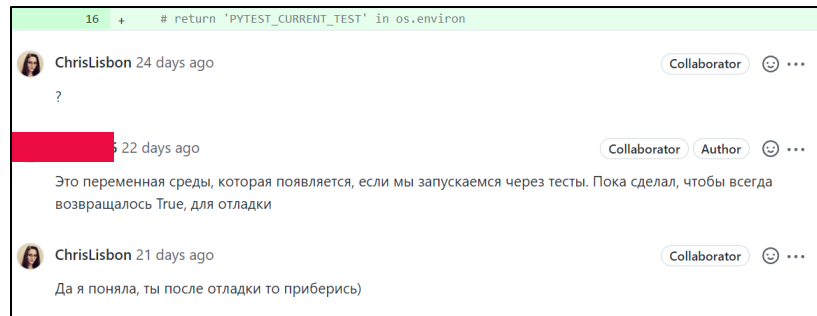
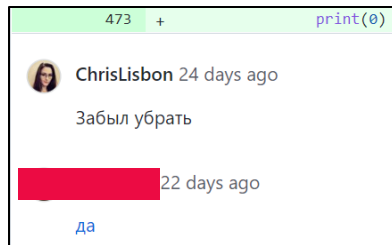
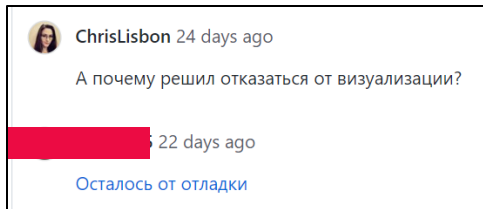
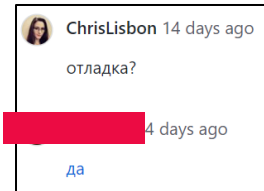
**Важно!** Сохранить логику своих изменений. Нужно вчитываться в код.

*Никто лучше вас не проверит как и зачем вы модифицировали чужой код*



# Общие правила хорошего тона (для автора) (1)

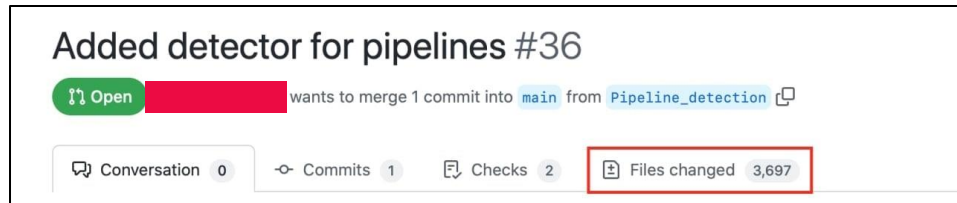
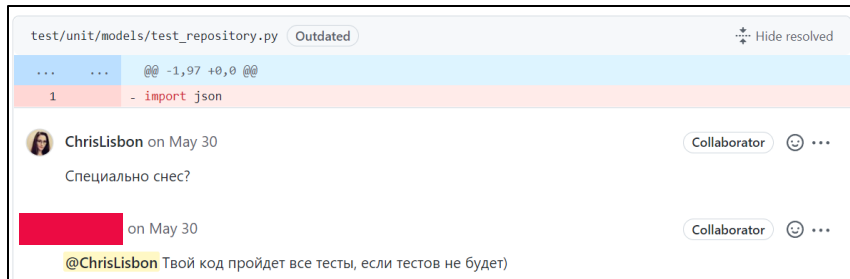
- 1) Делать rebase до назначения ревьюера – избавление людей от лишней работы;
- 2) Делать подробное и понятное описание PR'a;
- 3) Добавлять комментарии, указатели типов и описания функций, даже если кажется что «тут все очевидно»;
- 4) Убирать отладочные строки ДО назначения ревьюера.



# Общие правила хорошего тона (для автора) (2)



- 5) Добавить ссылку на issue, если PR его решает;
- 6) Убедиться, что новый код выполняет ту задачу, какую задумывалось;
- 6) Посмотреть на использованные паттерны, убедиться в том, что логика выдержана;
- 7) Убедиться, что пр содержит разумное количество изменений (200-400 строк\*). Если правок много, проще разбить их на несколько отдельных PR'ов;
- 8) Убедиться, что новые функции покрыты тестами ;
- 9) Убедиться, что все уже имеющиеся тесты успешно проходятся.



# Общие правила хорошего тона (для ревьюера)

## *Hard-skills*

- 1) Вникнуть в решаемую задачу, прогнать код при необходимости;
- 2) Проверить архитектурные решения, успешность решения задачи;
- 3) Проверить мелкие детали – опечатки, имена переменных и функций;
- 4) Проверить наличие и работоспособность тестов.

## *Soft-skills*

- 1) Не тянуть с началом ревью и каждым из циклов (оповещения на Git и т.п.);
- 2) Оформлять комментарии в обезличенном виде как предложения, а не требования;
- 3) Подкреплять предложения примерами уже реализованного кода;
- 4) Не стесняться спрашивать, если какие-то функции или участки кода непонятны;
- 5) Не стесняться просить о комментариях, если код не самоочевидный.

## Недостатки:

- 1) Внедрение концепта и обучение сотрудников;
- 2) Незначительное замедление процесса решения задачи.

## Достоинства:

- 1) Улучшение качества финального продукта, сокращение числа багов за счет их отлова на стадии проверки кода;
- 2) Повышение читаемости и качества кода – код должен быть понятен не только одному человеку;
- 3) Улучшение навыков младших сотрудников, применение «хорошего примера»;
- 4) Вовлеченность всей команды в различные задачи, взаимозаменяемость специалистов.

Статьи на habr:

Code Review – зачем и как использовать в команде?

Code review по-человечески (часть 1)

Code review по-человечески (часть 2)

Англоязычные:

GitLab - What are code reviews, and how do they work?

Google - How to do a code review



Спасибо за внимание!

**it**MO *re than a*  
**UNIVERSITY**