

Лекция 12. **Качество тестирования**

- Код-ревью
- Нестабильные тесты
- Метрики тестирования

HOW TO MAKE A GOOD CODE REVIEW

Проблемы при проведении код-ревью:

- ✓ Сильная эмоциональна привязанность к своему коду
- ✓ Ментальная модель «Мы против Них»

Хорошая практика:

- ✓ Перед началом установить стандарты оформления кода и определиться с термином «Готово»
- ✓ Использовать чек-листы проверки кода
- ✓ Придерживаться методологии «Гуманных код-ревью».



AT LEAST WE
DON'T NEED TO
OBFUSCATE IT
BEFORE
SHIPPING

*RULE 1: TRY TO FIND
AT LEAST SOMETHING
POSITIVE*

Методология «Гуманных код-ревью» ключевые принципы:

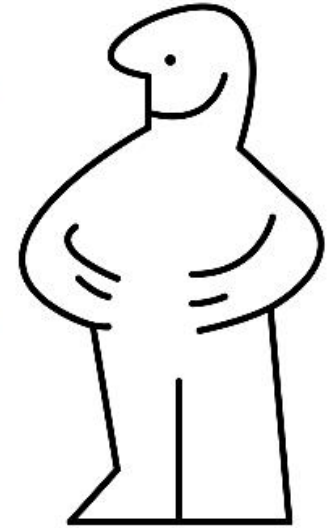
- ✓ Любой код, в том числе ваш, может быть лучше. Если кто-то хочет изменить что-то в вашем коде, то это **не нужно воспринимать как личное нападение**.
- ✓ **Речь идёт о коде, а не его авторе.**
- ✓ Никаких личных чувств. Избегайте мыслей вроде «...потому что мы всегда делали это так», «Я потратил на это уйму времени», «Это не я дурак, а ты», «Мой код лучше» и так далее. **Улыбнитесь, простите себя, взбодритесь и двигайтесь вперёд.** **Неудача – мать успеха** (кит. пословица)
- ✓ Пусть все ваши комментарии будут позитивными и направлены на **улучшение кода**. Будьте добры к его автору, но с ним не церемоньтесь.

Вместо того чтобы говорить: «**Ты пишешь код, как школьник**», попробуйте так: «**Мне сложно понять, что тут происходит**».

Чувствуете разницу? Первый вариант – личное оскорбление, а второй – конструктивная обратная связь.

Код-ревью авто-тестов

```
// Ждем обновления сообщения об ошибке  
await bro.pause(1000);
```



Find in Files 45 matches in 29 files

Q bro.pause(|

In Project Module Directory Scope

/Users/dmitrytuchs/IdeaProj

await bro.pause(1000);

await bro.pause(1000);

await bro.pause(1000);

await bro.pause(250);

await bro.pause(2000);

await bro.pause(2000);





Делаем код-ревью правильно

<https://habr.com/ru/companies/ruvds/articles/803127/>

В начале своей карьеры мы работали над платформой sentiment-анализа для социальных сетей. Наша команда состояла из семи человек. Мы были молоды и полны энтузиазма. Наш девиз можно было описать как: «Мы гибкие, быстрые и всё ломаем!». Да, мы действительно гордились своей скоростью. **Код-ревью? Я вас умоляю. Мы считали эту практику бюрократическим пережитком корпоративного мира.**

И что вы думаете? **Через несколько месяцев** наша база кода стала подобна минному полю. Проблема заключалась в том, что **никто не мог понять код, написанный другими**. У нас во многих местах дублировалась логика, и в модулях использовались разные стили кода.

Тогда до нас дошло! Нужно взять всё под контроль. **Код-ревью реально помогают сохранять код читаемым, обслуживаемым и масштабируемым.**

Итак, в двух словах: ***если вы не проводите код-ревью, или делаете их «для галочки», то обрекаете себя на боль, пусть не сразу, но в конечном итоге однозначно. Это можно сравнить с возведением дома на фундаменте из песка. Какое-то время он, может, и простоят, но явно недолго. А в мире стартапов второго шанса у вас может уже не быть.***

Проблемы авто-тестов

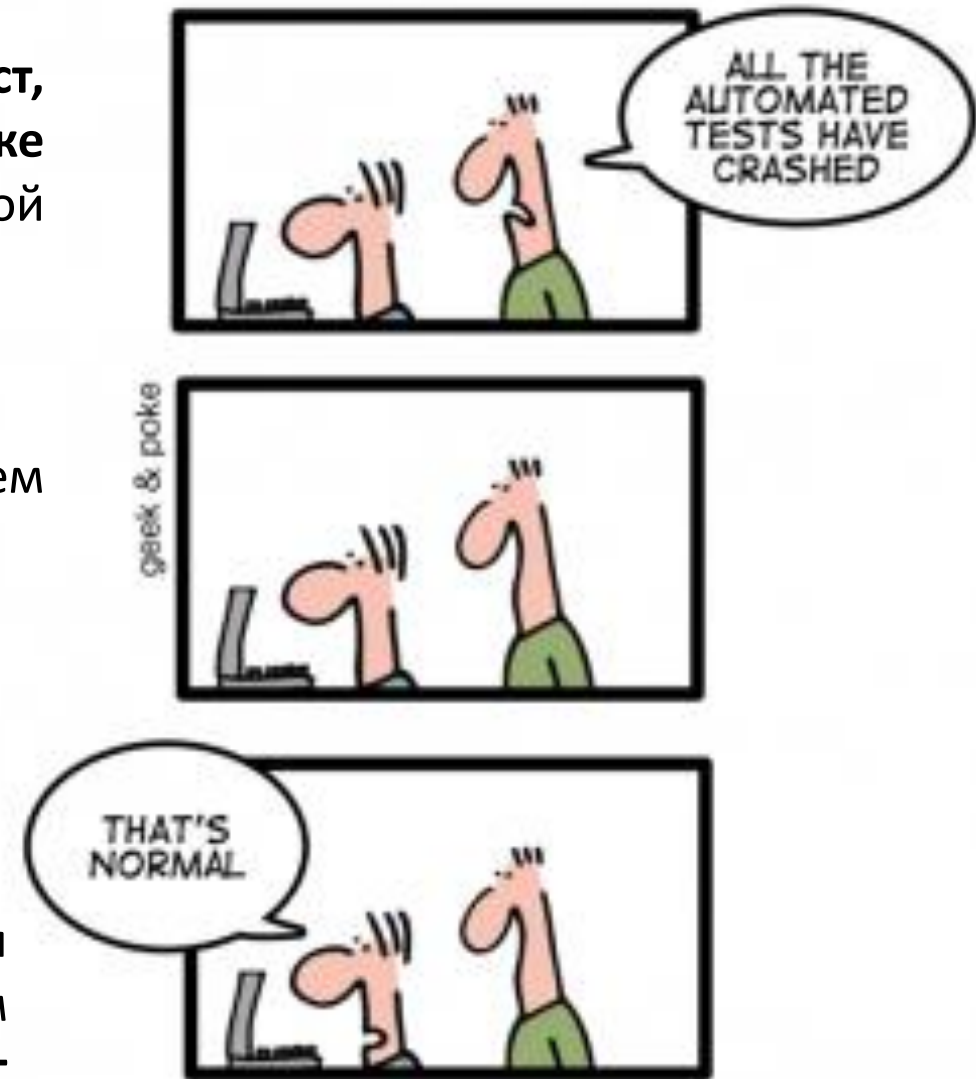
Нестабильные тесты

Flaky-тест в ИТ-тестировании означает нестабильный тест, который иногда “pass”, иногда “fail” учитывая одну и ту же конфигурацию теста, и трудно понять, по какой закономерности.

- Такие тесты сложно обнаружить.
- На такие тесты тратится много времени и ресурсов.
- Возникает задержка, пока команда не разберется, в чем дело.

НО такие тесты могут помочь найти ошибки, которые не могли бы быть обнаружены без такой нестабильности. Можно раскрыть некоторые плохие инфраструктуры или конструкции тестовой среды.

Парадокс: подавление всех нестабильных тестов не является полностью лучшим решением (дорого) и в целом недостижимо. В действительности **существует баланс, порог толерантности.**



Классификация нестабильных тестов это первостепенный шаг, позволяющий лучше использовать ресурсы (людей, время, CI и стоимость).

Цель данной классификации - иметь возможность разделить ненадежные тесты на группы в соответствии с их происхождением, чтобы расставить их по приоритетам.

Можно заметить две категории, которые кажутся более очевидными, чем другие, в программных проектах:

- ✓ **Независимые нестабильные тесты:** Тест, который дает сбой независимо, вне или внутри тестового набора. Благодаря легкости воспроизводимости их легче заметить, отладить и решить.
- ✓ **Системные нестабильные тесты:** тесты, которые не срабатывают из-за проблем с окружением, общим состоянием или даже из-за их порядка в тестовом наборе. Их гораздо сложнее обнаружить и отладить, поскольку их поведение может меняться вместе с эволюцией системы или рабочего процесса.

Наиболее часто встречающиеся причины нестабильности тестов:

- ✓ **Недостаточная изоляция:** тесты, не использующие копии ресурсов, могут привести к условиям гонки или конкуренции ресурсов при параллельном запуске. Кроме того, тесты, которые изменяют состояние системы или взаимодействуют с базами данных, всегда должны очищаться после использования.
- ✓ **Параллелизм:** несколько параллельных потоков взаимодействуют нежелательным образом
- ✓ **Зависимость от порядка тестирования:** тесты, которые могут завершиться неудачей или успехом в зависимости от порядка их выполнения в тестовом наборе.
- ✓ **Сеть:** Тесты, полагающиеся на сетевое подключение, которое не является параметром, который можно полностью контролировать.
- ✓ **Время:** Тесты, основанные на системном времени, могут быть недетерминированными и их трудно воспроизвести в случае сбоя.
- ✓ **Асинхронное ожидание, вызовы остались несинхронизированными или плохо синхронизированными:** тесты, которые выполняют асинхронные вызовы, но не ждут должным образом результата. Тесты должны избегать любого фиксированного периода сна. Время ожидания может различаться в зависимости от среды.
- ✓ **Зависимость от среды:** результаты теста могут различаться в зависимости от среды, в которой он выполняется.

Наиболее часто встречающиеся причины нестабильности тестов

(продолжение):

- ✓ **Операции ввода/вывода:** Тест может работать нестабильно, если он неправильно собирает мусор и не закрывает ресурсы, к которым он получил доступ.
- ✓ **Доступ к системам или сервисам, которые не являются абсолютно стабильными:** лучше использовать фиктивные сервисы как можно полнее, чтобы избежать зависимости от внешних, неконтролируемых факторов.
- ✓ **Использование генерации случайных чисел:** При использовании генерации случайных чисел или других объектов полезно регистрировать сгенерированное значение, чтобы избежать ненужного сложного воспроизведения сбоя теста.
- ✓ **Неупорядоченные коллекции:** не делайте предположений о порядке элементов в неупорядоченном объекте.
- ✓ **Жестко заданные значения:** тест, использующий постоянные значения, в которых элементы или механика могут изменяться со временем.
- ✓ **Слишком узкий диапазон тестирования:** при использовании выходного диапазона для утверждения может оказаться, что не все результаты были рассмотрены, и в случае их возникновения тест будет провален.

Итак, что делать, чтобы нестабильных тестов было меньше:

- тесты должны быть написаны в правильном слое "той самой пирамиды": **чем ближе слой к модульным тестам (а лучше именно в них), тем меньше шансов на моргания, потому что зависимостей меньше.**
- **основные причины нестабильности это асинхронные операции (async wait), многопоточность (concurrency), порядок тестов, утечка ресурсов, проблемы с зависимостями (сеть, время).** Поэтому, чем меньше этого в тестах, тем они стабильнее.
- основной объем бизнес-логики проверяем максимально близко к месту логики и **с максимальным количеством замокированных зависимостей** (но не переусердствуйте, а то будут другие проблемы)

Что делать, если они появились?

- если сейчас нет возможности разобраться с ошибкой, переместите этот тест в "карантин", чтобы позже с ним разобраться. Не надо держать в наборе запускаемых тестов тот, доверия к результатам которого нет.
- активно используйте трейсинг (логирование) в тестах и продакшен-коде для того, чтобы воспользоваться ими при расследовании. Совет: здорово, если у вас есть возможность "объединить" логи тестируемой системы с логами тестов. Мы активно использовали запись меток о начале/завершении теста в продакшен логах приложения. Очень помогало.
- для UI-тестов имейте возможность включить запись видео или скриншоты в момент проверки
- попробуйте переместить моргающую проверку на другой слой пирамидки
- если тест не поддается, подумайте, может стоит его удалить? Все равно смысла от него немного, особенно если думать про него не "случайно упавший", а "случайно успешный". Ну и в целом "Flaky tests are worse than _no_ tests".
- иногда советуют перезапускать упавшие тесты в надежде на удачу. В целом рабочий способ, но не надо им злоупотреблять. Он хорошо помогает с подтверждением проблемы и поиском test war. Но обнаруженные проблемы, например, с медленной инфраструктурой/сетью, особенностями фреймворков важно всегда фиксировать и планировать время на исправление.

Метрики в тестировании

Метрика – это мера, позволяющая получить численное значение некоторого свойства или процесса.

Метрики тестирования — это количественные показатели, характеризующие

- продвижение процессов тестирования ПО,
- уровень качества этих процессов,
- и производительность QA-команды.

Метрики тестирования дают возможность:

- Определить, какие улучшения QA-процессов понадобятся для создания качественного программного продукта без дефектов.
- Анализировать ход текущих этапов тестирования, а далее вносить изменения в график проекта и финансовые планы.
- Анализировать текущие технологии и процедуры, для их совершенствования.

Существует множество метрик тестирования и каждая компания использует свой набор метрик или даже разрабатывает собственные метрики в соответствии с условиями работы команды и особенностями проекта.

Тестовое покрытие

Тестовое покрытие —показывает, какой процент кода приложения был выполнен в процессе тестирования. Чем выше показатель тестового покрытия, тем больше уверенности можно иметь в том, что код работает корректно и без ошибок.

Виды тестового покрытия

- ✓ **Покрытие строк кода** (Line Coverage) — процент строк кода, которые были выполнены тестами. Здесь важно обратить внимание на то, что не все строки кода необходимо тестировать, например, комментарии и пространства имён.
- ✓ **Покрытие ветвей** (Branch Coverage) — процент ветвей кода (if, else, switch и т.д.), которые были выполнены тестами. Этот вид покрытия позволяет оценить тестирование разных сценариев выполнения кода.
- ✓ **Покрытие функций** (Function Coverage) — процент функций и методов, которые были выполнены тестами. Это позволяет оценить, насколько хорошо каждая функция или метод приложения был протестирован.

Например, инструмент **Istanbul** позволяет анализировать покрытие кода модульными тестами. Он выводит детальные отчёты, ориентируясь на которые можно получить точное представление о том, что в проекте ещё не протестировано, и оценить объём работ, необходимый для улучшения ситуации.

```
1  function add(a, b) {
2      return a + b;
3  }
4
5  function subtract(a, b) {
6      return a - b;
7  }
8
9  // Тесты
10 console.assert(add(2, 3) === 5);
11 console.assert(subtract(5, 2) === 3);
```

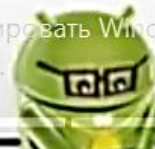
После выполнения тестов с использованием **istanbul**, мы получим следующий отчет о тестовом покрытии:

```
=====
Statements    : 100% (4/4)
Branches      : 100% (2/2)
Functions     : 100% (2/2)
Lines         : 100% (4/4)
=====
```

Анализ требований	Качество документации, Сложность требований
Тест дизайн	Скорость разработки ТС Сложность ТС
Функциональное Тестирование	Плотность дефектов Коэффициент ретеста Коэффициент регрессии
Автотестирование	Скорость разработки автотестов Длительность выполнения автотестов. Дефекты в автотестах.
Внедрение и сопровождение	К-во пропущенных дефектов в PROD Статистика использования
Управление тестированием	Время простоя тестирования Настроение команды Эффективность тестирования

Можно разделить метрики
по стадиям процесса
тестирования

Активация Windows
Чтобы активировать Windows,
зайдите в меню "Параметры".



Простые метрики процесса тестирования

Эффективность тест-кейсов (Test Case Effectiveness) =

(Количество обнаруженных дефектов / Количество выполненных тест-кейсов) x 100

Процент прошедших тест-кейсов (Passed Test Cases Percentage) =

(Общее количество прошедших тест-кейсов / Общее количество выполненных) x 100

Процент упавших тест-кейсов (Failed Test Cases Percentage) =

(Общее количество упавших тест-кейсов / Общее количество выполненных) x 100

Процент заблокированных тест-кейсов (Blocked Test Cases Percentage)=

Процент заблокированных тест-кейсов = (Количество заблокированных тест-кейсов / Общее количество выполненных) x 100

Процент исправленных дефектов (Fixed Defects Percentage)=

(Общее количество исправленных дефектов / Количество заявленных дефектов) x 100

Какие метрики чаще всего упоминаются в статьях по тестированию

- ✓ **Passed/Failed Test Cases.** Используется для оценки отношения успешно пройденных тестов к завершившимся с ошибками. Метрика помогает оценить успешность прохождения тестов.
- ✓ **Not Run Test Cases.** Метрика помогает определить причины невыполнения тестов и способы их устранения.
- ✓ **Open/Closed Bugs.** Формируется из отношения открытых багов к закрытым. Метрика оценивает скорость устранения багов, а также позволяет выявить причины, по которым ошибки остались незакрытыми.
- ✓ **Reopened/Closed Bugs.** Рассчитывает соотношение переоткрытых багов к закрытым. Метрика демонстрирует эффективность закрытия бага разработчиками и поможет выявить причины, по которым исправление ошибок находится на низком уровне.
- ✓ **Bugs by Severity/Priority.** Общее количество багов по серьёзности/приоритету. Метрика показывает качество предоставляемого кода на тестирование.

Топ 5 QA-метрик для улучшения качества тестирования

<https://habr.com/ru/articles/771070/>

- Удовлетворенность пользователей
- Дефекты, обнаруженные в продакшене после релиза
- Тестовое покрытие требований
- Дефекты во время спринта
- Соотношение обещанных и выполненных сторей

Итак

Зачем нужно собирать метрики

- ✓ Улучшение контроля над процессом тестирования
- ✓ Поиск проблемных мест
- ✓ Оценка достижения целевых показателей
- ✓ Наглядная иллюстрация количественных показателей

Когда не нужно собирать метрики

- ✓ Маленький проект
- ✓ Отсутствует команда тестирования
- ✓ Не стремимся предупредить потенциальные проблемы
- ✓ Не боимся получать баги с прода

Пример эффективности использования метрик в компании

В отделе тестирования
компании работает 80 человек



ЮMoney

Метрики.

Качество и стабильность автотестов

- **Unstable Tests (UT)** - количество нестабильных автотестов
- **Bad Tests (BT)** - "плохие" автотесты
- **Skipped Autotests (SAT)** - количество заблокированных автотестов
- **Production Found Defects (PFD)** - инциденты на проде
- **Integration Found Defects (IFD)** - дефекты, найденные на этапе интеграционного тестирования
- **Testcases** - количество тест-кейсов
- **Autotests** - количество автотестов
- **Coverage** - отношение Autotests к Testcases
- **Test Duration (TD)** - среднее время тестирования задач

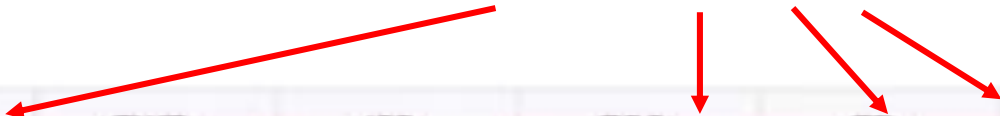
<https://www.youtube.com/watch?v=rlopjjxNxzs>

Из опыта работы с метриками тестирования компании



ЮMoney

Собрали метрики, большинство метрик **неудовлетворительные**.



Дата	TS	TWT	IFD	TQS	TD	COVERAGE	PFD	TESTCASES
09.01 - 21.01	0.68	24%	0	5.1	4.6	2%	0	121
22.01 - 04.02	0.8	60%	1	3	3.18	2%	3	121
05.02 - 18.02	0.75	30%	0	2	2.8	6%	1	121

После анализа
«красных зон»
провели работу в
следующих
направлениях

- ротация в продуктовых командах,
- оценка тестирования при планировании спринта,
- внесение изменений во флоу работы с задачами,
- оптимизация процесса тестирования,
- покрытие кода автотестами

Метрики. Первые результаты



ЮMoney

Дата	TS	TWT	IFD	TQS	TD	COVERAGE	PFD	TESTCASES
09.01 - 21.01	0.68	24%	0	5.1	4.6	2%	0	121
22.01 - 04.02	0.6	66%	1	3	3.16	2%	3	121
05.02 - 18.02	0.75	30%	0	2	2.8	2%	0	121
19.02 - 04.03	0.25	55%	0	1.3	3.3	2%	0	121
05.03 - 18.03	0.62	0%	0	1.9	2.7	2%	0	121
19.03 - 01.04	0.7	0%	0	2.3	1.8	2%	0	121
02.04 - 15.04	0.82	0%	0	3.7	1.87	7%	0	121
16.04 - 29.04	0.98	0%	0	2.7	2.08	9%	1	128
03.05 - 11.05	0.8	0%	0	1.8	1.61	9%	3	128

Не трудно заметить, что после проведенной работы по анализу метрик тестирования и принятия мер красных метрик стало значительно меньше