Тесты - инструмент разработчика

Антон Галицын, Continuous Delivery

Всем привет, меня зовут Антон, я работаю в команде CD. Наша тема сегодня - тесты, как инструмент разработчика

Agenda

- Проблемы с качеством
- Потребности бизнеса
- Виды тестов, разделение ответственности QA и Dev
- Unit, Функциональные тесты (с демо)
- Кросс-продуктовые сценарии
- Заключение

Для тех, кто не знает, что будет, хочется огласить план.

Цели

- Внести ясность в терминологию
- Объяснить, что некоторые виды тестов - ответственность разработчика
- Показать, что некоторые виды тестов - инструмент для разработки
- Зародить новые мысли у разработчиков о улучшениях в своей команде
- Призвать более осознано относиться к состоянию продукта в целом

Цели данного мероприятия следующие:

Ответы на вопросы

- Для чего developer должен писать тесты?
- А какие тесты developer должен писать?
- А что тогда будет делать QA?
- Какие есть best practices для написания unit и функциональных тестов?

Чего точно не будет?

1. Что такое Continuous Delivery?
2. Проблема выбора данных для тестов или Fixtures
3. UI тесты
4. Что такое качество?
5. Взаимодействие QA и Dev
6. TDD
7. Deploy
8. Load тесты
9. A/B тесты
10. Monitoring и Alerting

Сегодня я открываю серию докладов о качестве.

Первые 5 скорее всего будут в рамках tech talks. Каждую тему важно проговорить вслух.
Каждый, кто заинтересован и хочет помочь - welcome.

Проблемы с качеством

- “У нас нет QA, безресурсье!…”
- “Сейчас у нас мало тестов, допишем после релиза…”
- “Да у нас много тестов, но они все skipped и писали их 3 года назад…”
- “Опять этот релиз, он такой жесткий, пойду пить…”
- “Какие-то баги на бою, на моем вагранте все работает…”
- DPWEB-9 - Лекция "Введение в тестирование ПО"

По объективным причинам есть проблемы с качеством. Об этом говорит и джира, и количество писем и фразы коллег. Например, <прочитать фразы>. Осознание проблемы девелопментом настолько велико, что появился тикет о просьбе сделать такую лекцию.

Потребности бизнеса

- Быстрая доставка кода клиентам
- Высокое качество продуктов

Мы все работаем в веб отделе.

Мы делаем продукты на рынок, то есть в конкурентную среду.

От того, на сколько быстро мы выпускаем продукты зависит положение компании на рынке. Потребности людей увеличиваются, и им нужно, чтобы они удовлетворялись как можно быстрее.
Под быстрой доставкой кода понимается следующие три составляющие: новые продукты, новые фичи в продуктах, фикс багов найденных на боевом окружении.

При этом необходимо всегда помнить о качестве выпускаемых продуктов. То есть, грубо говоря, мы должны выпускать новый функционал, не выпуская при этом новых багов.

Потому что высокое качество программных продуктов это стандарт. Пользователи привыкли к качественному софту и качество является одним из критериев выбора продукта.
То есть даже если мы выпускаем проект или задачу в срок, но при этом в продакшне находится много багов, сложно считать такой проект успешным.

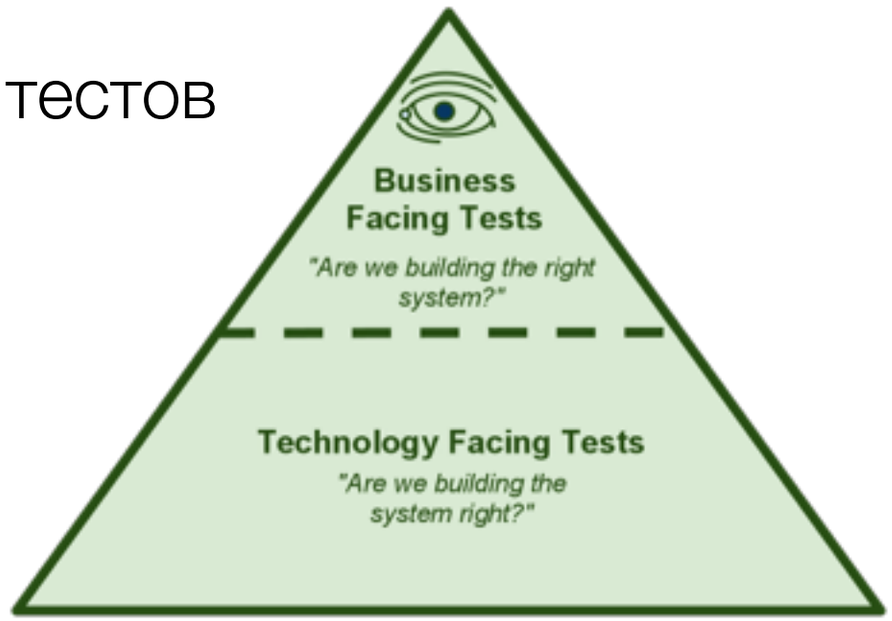
Под обеспечением качества понимается не только тестирование продуктов силами команды QA, но и разработчиками.

Резюмируя, можно сказать следующее, что конкуренция на сегодняшний день идет по следующим параметрам:
1. Скорость выпуска нового функционала
2. Качество
3. Стабильность
4. Производительность
5. И некоторых других

Наличие авто тестов - неотъемлемая часть решения по удовлетворению потребностей рынка

Наличие работающих и актуальных авто тестов - неотъемлемая часть решения по удовлетворению потребностей рынка. Мы в нашей команде называем это Continuous Delivery pipeline, но это штука - материал другого tech talks, я уже сказал это. Я не могу о ней не упомянуть. Все вопросы по этой теме после доклада.

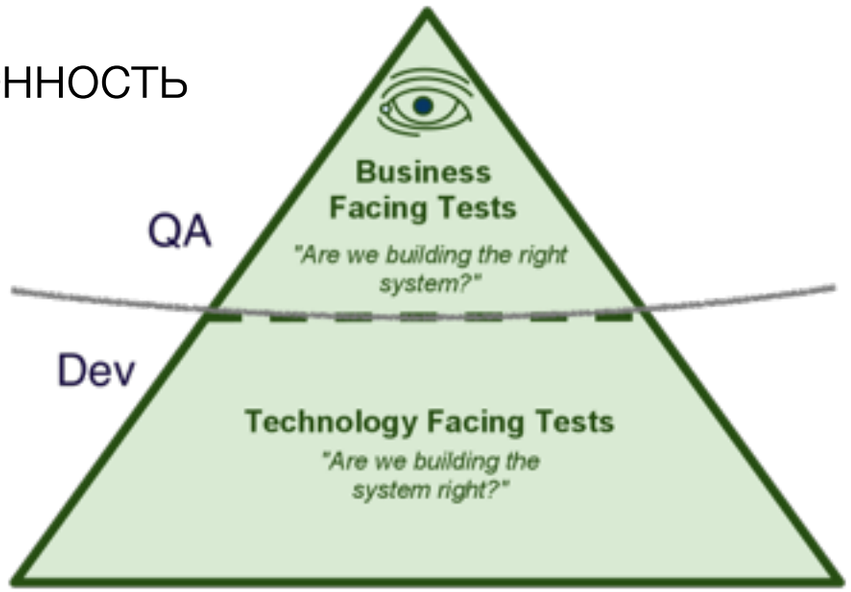
Группы тестов



Я бы очень грубо выделил 2 большие группы, на которые можно поделить тесты:

* Тесты, проверяющие приложение, отвечающие на вопрос, строим ли мы систему правильно?
* Тесты, проверяющие продуктовые бизнес-сценарии, отвечающие не вопрос, а правильную ли систему мы строит?

Ответственность

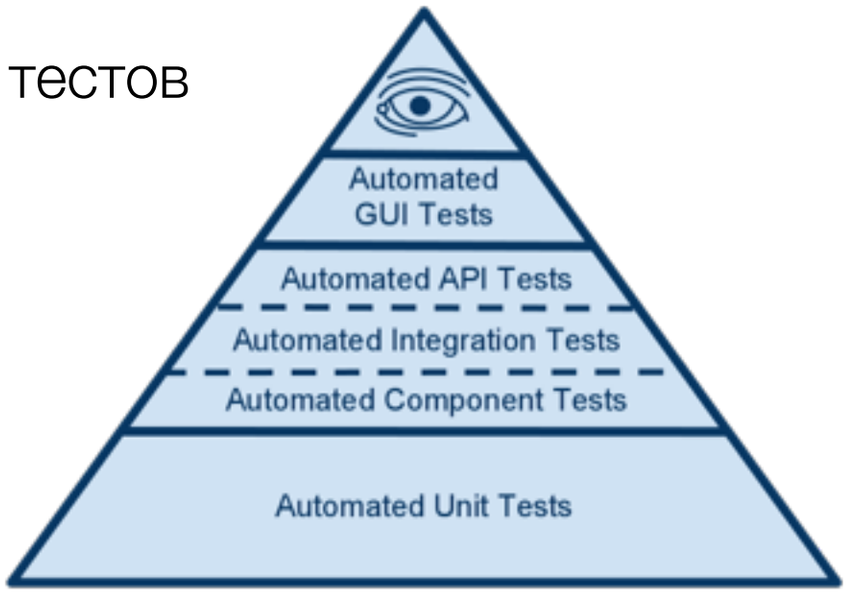


Чаще всего, нижняя часть - Тесты, проверяющие приложение, зона ответственности Dev

Верхняя часть - Тесты проверяющие продуктовые бизнес-сценарии, зона ответственности QA

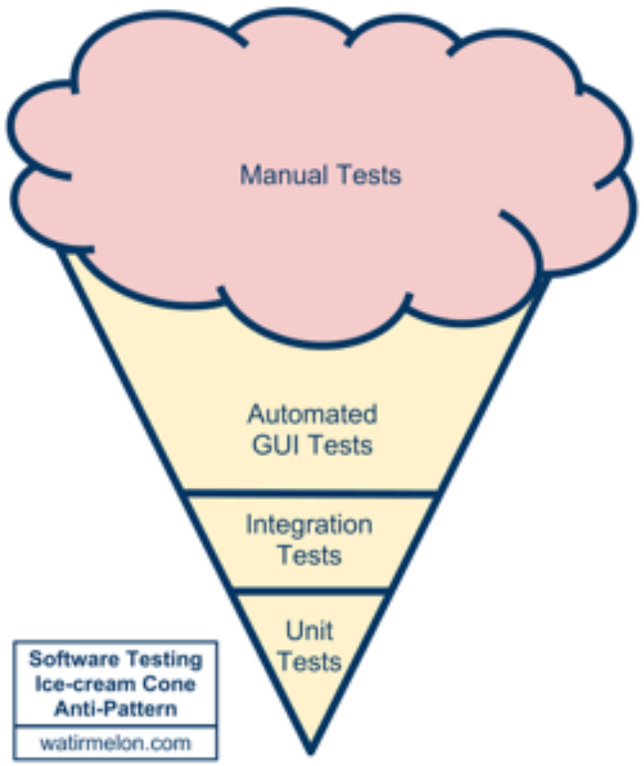
Я бы добавил про пропорции покрытия функционала потребителя, например, юнит тестов может быть 17000 и они будут покрывать
3% всего клиентского функционала. А UI тестов может быть 1500, и они могут покрывать свыше 50% функционала (Взято с реального проекта)

Пирамида тестов



Первые 2 нижних кирпича - инструмент разработчика, ответственность разработчика и фокус нашей сегодняшней презентации

Типичная ситуация



Но чаще всего мы видим такую картину, автотесты есть только на бизнес-сценарии, есть ручное тестирование, фактически отсуствуют юнит тестов. Так быть не должно.

Отсюда выразите ряд проблем:
* Релиз “перекрестившись”
* Боязнь рефакторинга
* Новые баги
* Циклические поломки одного и того же

```
commit 9955b7319157edb09db9703e726cbf7e0810f581
Author: Anton Galitsyn <a.galitsyn@2gis.ru>
Date:   Thu Nov 13 14:52:35 2014

    Temporary mark failed UI tests as skipped
```

Неприемлемо

Так же я слышал, что часто в некоторых командах, некоторые люди помечают тесты, которые не проходят, как skipped.
Такие вещи не приемлемы, чаще всего такие тесты остаются нерабочими навсегда.

“А как мне замокать sapphire2 для unit теста?”

–Разработчик, имя которого нельзя называть

Теперь детальнее обсудим тесты приложения, которые помогают нам строить систему правильно. Перед вами реальный вопрос от разработчика, имя которого нельзя называть.

Естественно реакция примерно такая

Изоляция

- Тесты приложения не должны зависеть от сторонних сервисов
- Максимальная связность - слои вашего приложения
- Приложение - белая коробка, вы знаете, как оно работает

Справа - схема связей продуктов в веб отделе, можно даже не пытаться ее разглядеть, главное понять, что компонентов очень много.

Unit тесты

Проверка логики небольших фрагментов кода без зависимостей

Демо

Unit тесты

- Изолируем класс, зависимости в mock
- Тестируем логику отдельного куска кода
- Не надо инфраструктуры - только код

И так, резюме

Чем поможет?

- Рефакторинг
- Упрощение интеграции слоев приложения
- Тест, как документация
- Заставляет писать менее связанный код

Модульное тестирование позже позволяет программистам проводить рефакторинг, будучи уверенными, что модуль по-прежнему работает корректно (регрессионное тестирование).

Модульное тестирование помогает устранить сомнения по поводу отдельных модулей и может быть использовано для подхода к тестированию «снизу вверх»: сначала тестируя отдельные части программы, а затем программу в целом

Модульные тесты можно рассматривать как «живой документ» для тестируемого класса. Клиенты, которые не знают, как использовать данный класс, могут использовать юнит-тест в качестве примера.

Например, класс пользуется базой данных; в ходе написания теста программист обнаруживает, что тесту приходится взаимодействовать с базой. Это ошибка, поскольку тест не должен выходить за границу класса. В результате разработчик абстрагируется от соединения с базой данных и реализует этот интерфейс, используя свой собственный mock-объект. Это приводит к менее связанному коду, минимизируя зависимости в системе.

Когда использовать?

“На pre-commit unit тесты запускать ты должен…”

Так как такие тесты отрабатывают быстро, надо запускать их ВСЕГДА

Слои Web-приложения

- HTTP abstraction
- Routing
- ORM
- Templating
- API
- Configs
- Cache
- …

Давайте поговорим про структуру приложения.

Это прежде всего совокупность разных подсистем, каждая из которых выполняет конкретную функцию. Для примера я привел компоненты, которые есть в любом фреймворке любого языка для веб приложений.

Функциональные тесты

- Проверка способности решать задачи, которые исходят из функциональных требований к приложению
- Проверяют интеграцию разных слоев приложения
- Изолированны в рамках приложения

Функциональное тестирование — это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определённых условиях решать задачи, нужные пользователям. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает.

Больше компонентов

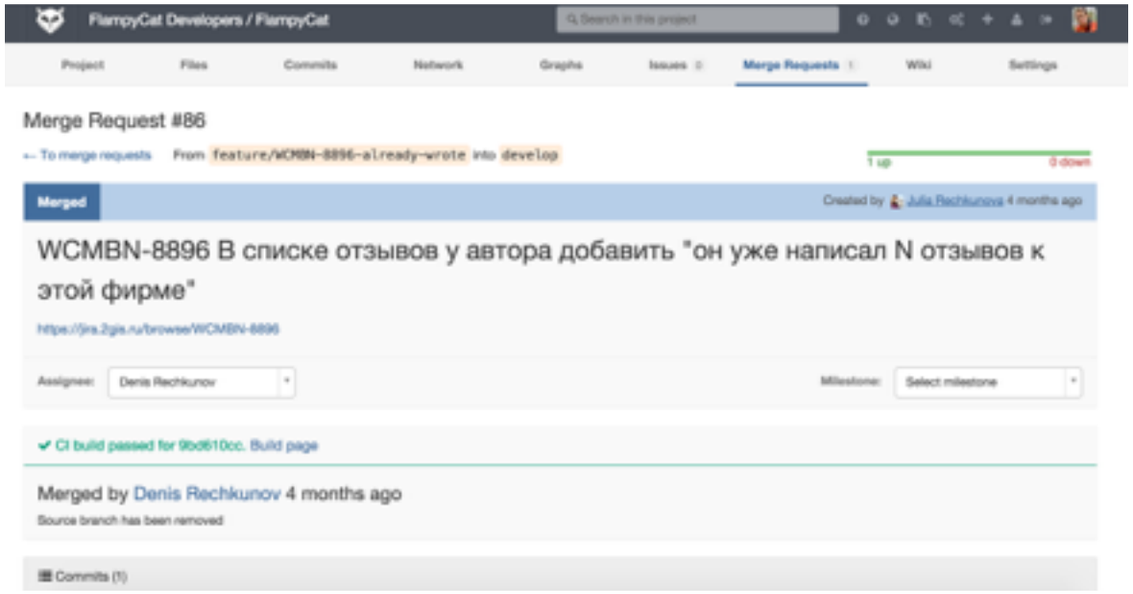
- Отдельный config для test env
- Тестовая БД
- Fixtures
- Fake backend’ы внешних сервисов
- …

Больше компонентов, но только в рамках приложения. Помним об изоляции

Демо

Чем поможет?

- Тестируется работа всей системы в целом
- Тесты пишутся с учетом спецификации, выступают примером использования
- Видно интерфейс использования
- “Глубоко” проходят по коду, а пишутся быстро



Когда использовать?

На merge request

Скорее всего такие тесты идут несколько минут, нецелесообразно запускать тесты на комит, лучше на мертв реквест. Картинки из флампа

“Мой проект настолько стар, насколько я не стар. Как я напишу тесты?”

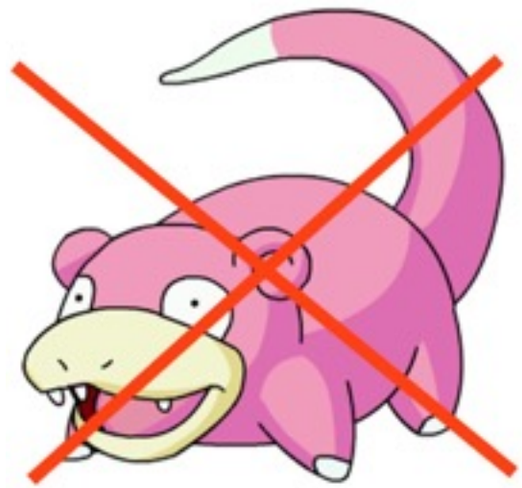
–Разработчик, имя которого нельзя называть

Всегда есть выход

- Bug fix - отличная возможность написать функциональный тест
- Refactoring - отличная возможность написать unit тест

Немного инициативы

- Закладывайте время на тесты в оценки задач
- Начинайте писать тесты
- Собирайте рабочие группы совместно с QA
- Приходите за консультациями в CD и Online



У меня есть пример, про параллелс плеск, где не было юнит тестов

Спасибо за внимание!

a.galitstyn@2gis.ru

Кросс-продуктовые сценарии

- Есть сценарии, которыми пользуется конечный клиент
- Сценарии определяет либо ПМ, либо QA
- Нужны спеки по фичам
- Нужны эксперты в предметной области
- Сервисы настоящие, много технических проблем

Сценарии чаще всего кросс-сервисные, для апи это использование онлайном.

Процесс тестирования

- Деплой сервисов
- Конфигурация, соединение сервисов
- Выбор тестовых сценариев
- Прогон тестовых сценариев
- Репортинг результатов
- Анализ

По опыту parallels, ежедневный процесс был такой

- https://gitlab.2gis.local/agalitsyn/introduction-to-testing/tree/master
- http://www.slideshare.net/AndrewDzynia/quality-built-in
- http://watirmelon.com/2012/01/31/introducing-the-software-testing-ice-cream-cone/
- http://watirmelon.com/2011/06/10/yet-another-software-testing-pyramid/
- http://symfony.com/doc/current/book/testing.html
- http://symfony.com/doc/current/components/console/introduction.html#testing-commands
- http://habrahabr.ru/post/148951/