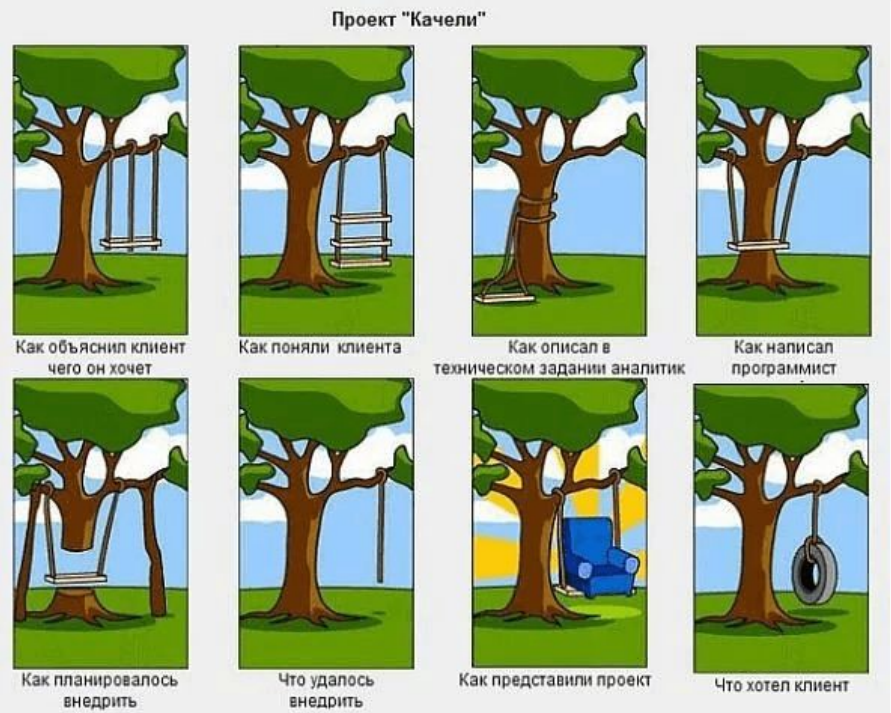
Организация тестирования в команде разработчиков, коммуникация и взаимодействие.

Что тестировать? Как тестировать? Кому тестировать?

Что такое тестирование?

Тести́рование програ́ммного обеспе́че́ния — процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определённым образом

Что такое тестирование?



Где применяется тестирование?

- Разработка ПО
- Производство
- Образование
- Научные исследования
- Спорт
- и т. д.

Важность тестирования

Иногда тестированию не уделяется должное внимание. В таких случаях конечный пользователь становится одним из звеньев тестирования, а вероятность передачи конечному пользователю некачественного продукта существенно возрастает. Цена тестирования уменьшается, но репутационные издержки существенно возрастают. При этом важно помнить, что практически невозможно создать продукт, в котором дефекты отсутствуют на 100%

Требования

Создать модуль, который выполняет работу с прямоугольником. Методы:

create, получающий на вход координаты верхнего левого и нижнего правого углов, а также цвет и вид заливки (может рисоваться контур, или залитый прямоугольник). Возвращается объект прямоугольник

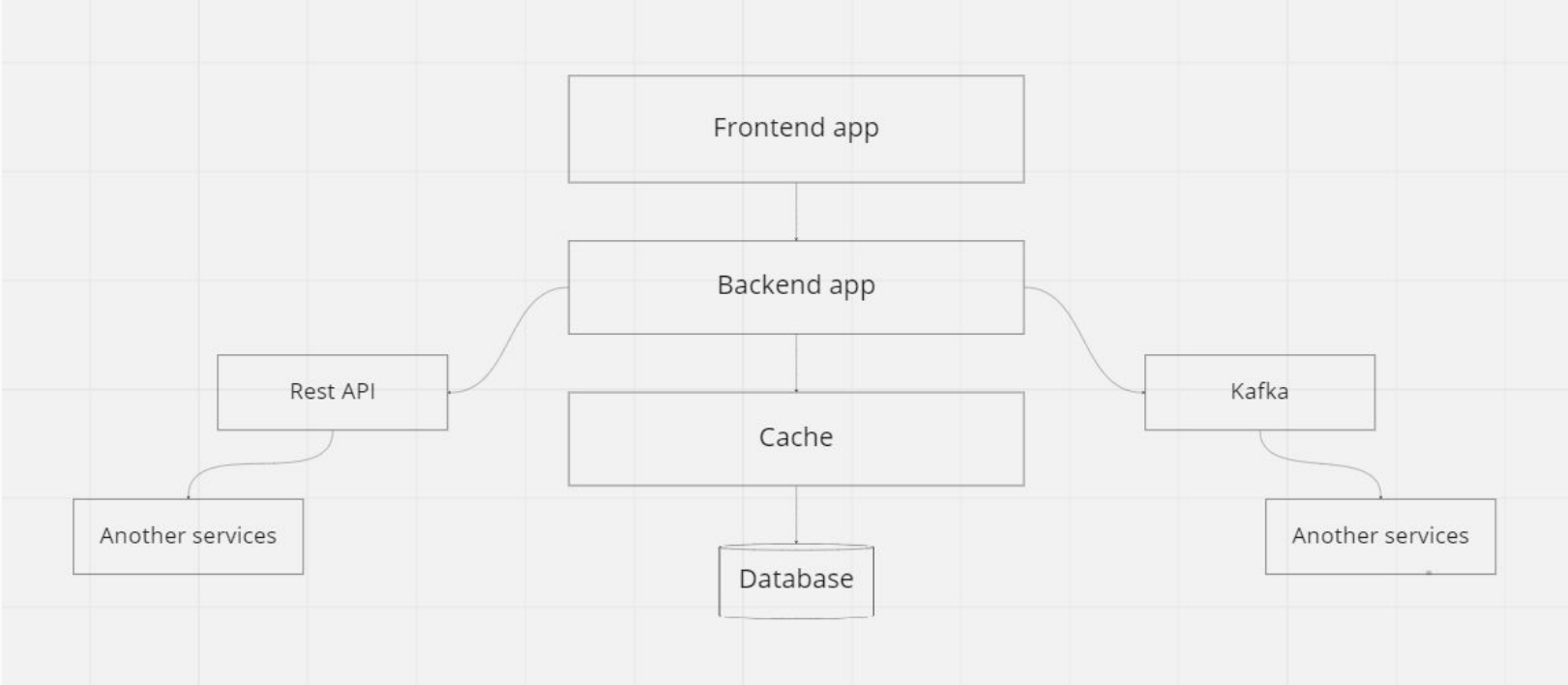
join, получает на вход два прямоугольника, цвет и вид заливки, возвращает прямоугольник наименьшего размера, в который можно вписать оба исходных прямоугольника

move, получает на вход прямоугольник и две целочисленных переменных. Возвращает прямоугольник, смещенный по x и y на значения переменных

Основы теории тестирования

- Ожидаемое поведение
- Фактическое поведение
- Требования
- Баг (дефект)
- Тестовое покрытие

Сложное приложение



Виды тестов

- Unit-тесты
- Интеграционные тесты
- Функциональные тесты
- Приемочные тесты
- Тесты производительности
- Smoke-тесты
- Ручные и автоматические тесты

Тестовые окружения (environments)

- local
- develop
- integration
- UAT - user acceptance testing
- production
- additional (automate, load, etc)

Классическая команда

- Тимлид
- Аналитик(и)
- Фронтенд-разработчик(и)
- Бэкенд-разработчик(и)
- Тестировщик(и)
- Автоматизатор(ы) тестирования

Подходы к организации тестирования

- Тестирование, основанное на требованиях
- Тестирование, основанное на рисках
- Экспертное тестирование

Артефакты тестирования

- Стратегия тестирования
- Тест-кейсы
- Чек-листы
- Протоколы тестирования
- Отчеты о тестировании

Варианты стратегии тестирования

- Тестируется каждая задача (задача закрывается тестировщиком)
- Тестируется каждая задача в рамках отдельной задачи
- Тестируется результат спринта (релиз целиком)
- Тестирование выполняется “по необходимости” (экспертное тестирование)
- Тестирование не выполняется (или выполняется только разработчиком)

Кейс 1: дефект требований

- Задача взята в работу. Разработка занимает неделю
- Разработчик считает, что выполнил ее и передает в тестирование
- Тестировщик приступает к тестированию и находит ошибки
- Разработчик берет заведенный по итогу дефект в работу и понимает, что в текущем виде требования реализовать невозможно

Итог: существенные временные затраты на реализацию задачи, которую невозможно реализовать, требуется доработка, которая займет много времени

Как избежать? Необходимо выполнять тестирование требований, или хотя бы валидацию требований тестировщиком

Кейс 2: дефект интеграции

- Разработчик берет задачу в работу и закрывает ее
- Тестирование изолированного приложения не находит ошибок
- Новый релиз приложения устанавливается на следующий тестовый контур, где обнаруживается блокирующий дефект: данные из смежной системы не поступают, возникает ошибка, вызывающая остановку приложения

Итог: при анализе выясняется, что во время разработки использовались устаревшие требования интеграции, необходима существенная и трудозатратная доработка

Как избежать? Согласовывать интеграцию с командой, которая отвечает за сервис, с которым выполняется интеграция. Не доверять описаниям

Кейс 3: “плавающий” дефект

- Задача взята в работу и закрыта
- Тестирование не выявило дефектов
- Задача успешно прошла приемочные испытания и вышла в релиз
- Во время эксплуатации периодически (редко) появляются баг-репорты от клиентов. Дефект не удается воспроизвести не на production окружении

Итог: существенные издержки для поиска и устранения дефекта. Риски, связанные с предоставлением доступа в production команде разработки

Как избежать? Стремиться к тому, чтобы окружения были максимально приближены к Production. Использовать в тестировании данные, построенные на реальных (напр. с использованием machine learning)

Выводы

Тестирование - важный и трудоемкий процесс, которым нельзя пренебрегать

Нет универсального паттерна организации тестирования, важно понимать особенности конкретного проекта и команды

Начинать тестировать надо как можно раньше, начиная с этапа формирования требований

Разработчики должны писать unit-тесты в объеме, который позволит исключить простые логические ошибки

Тестировщик должен глубоко понимать все детали проекта и активно общаться со всеми участниками процесса