Конспект по книге „Эффективное юнит-тестирование" Хориков

Глава 1

1. Цель юнит-тестирования — обеспечение стабильного роста проекта. Снижение энтропии, защита от регрессий.

2. Метрики branch coverage лучше, чем code coverage, считающую только строки кода. Но обе метрики хороший негативный показатель, но плохой позитивный. Хорошее покрытие — не говорит о хороших тестах. Плохое — говорит, что надо больше тестов написать.

3. Покрытие <60% — слишком мало кода протестировано.

4. Юнит-тесты б.б. интегрированы в цикл разработки, в идеале б. запускаться при каждом min изменении кода.

5. Хорошие тесты проверяют только max важные части, дают max защиту от багов с min затратами на сопровождение. Проще — удалим. Тесты, как и код — обязательство, а не актив.

Глава 2

1. AAA — arrange, act, assert. Структура теста.
SUT — system under test. Нейминг тестируемого кода.

2. В классической школе юнит — класс. Мокаем только совместные зависимости, которые мешают изолировать тесты друг от друга.

3. Тесты проверяют не единицы кода, а единицы поведения — имеющие ценность для бизнеса.

4. Юнит-тест: ① Проверяет 1 единицу поведения
② Делает это быстро
③ Изолированно от др. тестов
Остальное — интеграционный тест. Его подм-во — сквозной.

Глава 3

1. Не используй if в тестах — никаких. Если нужен if — тест проверяет слишком много, разбей на несколько.
2. Избегай секций Act из нескольких строк. Если это нужно — плохой API у SUT. Недостаток инкапсуляции
3. Каждый тест д. рассказывать историю, факт. Тест упал — значит, факт перестал выполняться.
4. Не переиспользуй тестовые данные между тестами. Создаёт ненужную связность между тестами. Создай методы-хелперы, которые по входным аргументам создают нужные объекты. Тогда не будет ненужной связности тестов.
5. Аналогично — не используй для подготовки тестовых данных конструктор. Это ещё и ухудшает читаемость тестов.
6. Название метода-теста — осмысленное, в форме факта, без имени тестируемого метода (тестим не его, а поведение) `delivery_with_a_past_date_is_invalid()`.
7. Посмотри аналог Fluent assertions в Python (это C# либа)

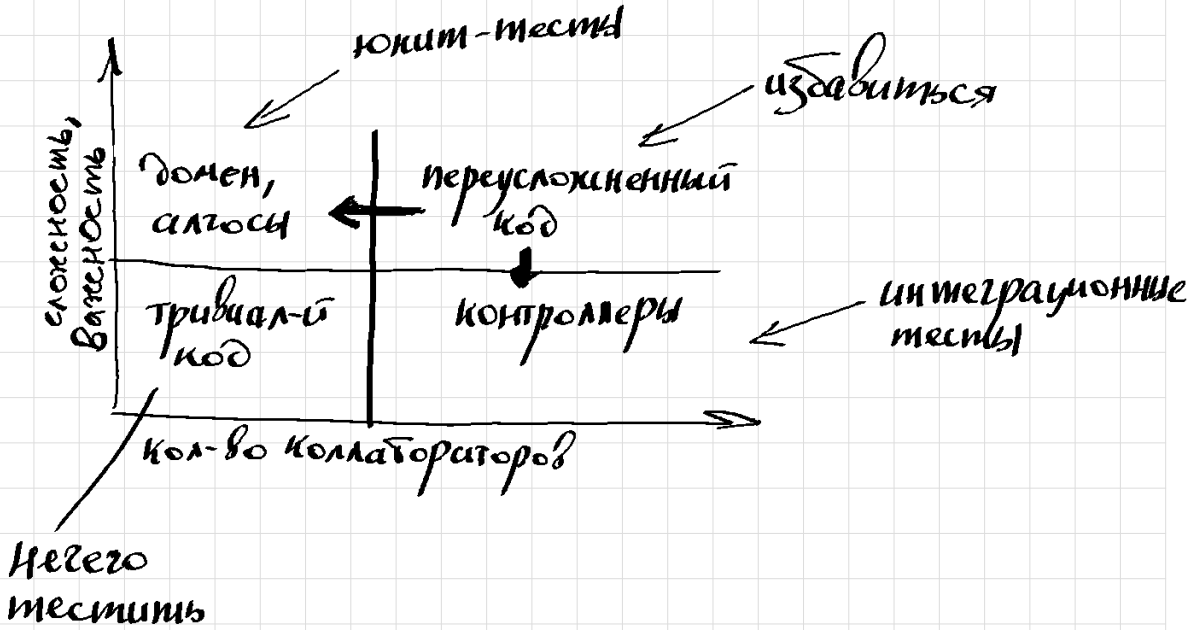
Глава 4

1. 4 аспекта хорошего теста (и юнит-теста, и интеграц-го):
   1. Защита от багов
   2. Устойчивость к рефакторингу
   3. Быстрая обратная связь
   4. Простота поддержки — понимания теста и его запуска
2. Устойчивость к рефакторингу — не предмет для компромиссов. Она должна быть. Всех нюнитов обаться так-нельзя.
3. Тестируй методом чёрного ящика, не белого. Поведение, а не детали реализации. Белый ящик при анализе теста.

Глава 5

1. Моки помогают эмулировать и проверять выходные воздействия (отправка email). Стабы — помогают эмулировать входные воздействия (получение данных).
2. Взаимодействия со стабами не проверяются. Деталь имплементации.
3. Публичный API д. раскрывать только необходимый min операций и состояния.
4. Моку для проверки внешних взаимодействий. Не для проверки внутренних зависимостей.
5. Внепроцессная зависимость, недоступная извне (СУБД, канр.), не мокируется. Это не часть наблюдаемого поведения, а деталь реализации.

Глава 7

1. 

2. Код д.б. или глубоким (сложным) или широким (с бол. кол-вом коллабораторов). Не всё вместе.

Глава 8

1. Когда исп. интеграционные тесты? Для тестирования контроллера. Проверяем 1 позитивный самый длинный сценарий, охватывающий все зависимости, а также граничные условия, не покрытые юнит-тестами.
2. Управляемые зависимости (СУБД) не мокаем, используем реальную. Неуправляемые (внешние по отношению к нашей системе — почта, внешние API) — мокаем, чтобы сохранить контракт.
3. Как тестить БД — вставляем данные в БД, выполняем тестируемый код, смотрим состояние БД.
4. Интерфейсы с одной реализацией — нарушение YAGNI, они не нужны — если только для мока. Т.е. для внепроцессных неуправляемых зависимостей (Email-отправка, вызов внешних API, передача в шину сообщений и т.п.)

Глава 9

1. Проверяй взаимодействие на границе системы. Свой тип адаптер над API шины — его мокай, а не более высокоуровневый код поверх адаптера. Ближе к контракту системы.
2. Моки только для интеграционных тестов. БЛ — в слое домена, она сложная и без внепроцессных зависимостей, там нечего мокать. Координация в слое контроллера, здесь внепроц-е зависимости и нет сложности (в идеале нет if), это интегр-е тесты с моками неуправляемых зав-тей.
3. Для мока проверяем наличие ожидаемых обращений и отсутствие неожиданных — проверяем количество обращений.
4. Мокай только свои типы данных. Внешняя либа? Создай адаптер и мокай его.
5. Верный sut result в блоке assert хардкодится, а не вычисляется. assert (sum (2, 3), 5), а не (sum, 2+3).

Глава 10

1. Интегр-е тесты не запускают параллельно. Юниты можно для ускорения, инт. — нет. Не усложняем.
2. В тесте три разных транзакции / use — в блоках AAA
3. Очистка БД — часть этапа подготовки. Базовый класс для всех интегр. тестов чистит БД. Просто голый SQL-скрипт с TRUNCATE. Справочные таблицы не чистим. Прям в конструкторе базового класса вызываем очистку.
4. Операции чтения из БД можно не тестировать. Только какие-то особо сложные сценарии. Слой домена в таких операциях не нужен, домен для согласованного изменения данных.
5. Репозитории и БД тестируем в составе общих инт-х тестов. Отдельные тесты для них незачем.

Глава 11

1. Приватные методы не тестируем, это детали реализации.