Практичный TDD

Антон Галицын, Continuous Delivery

План

- TDD - это просто?
- Фазы TDD
- Мифы о TDD
- Проблемы реального мира
- TDD и MVP продукт
- Выводы

TDD - это просто

- Красный - пишем тест
- Зеленый - быстро пишем код, чтобы тест проходил
- Рефакторинг - делаем код приемлемого качества, улучшаем дизайн системы

С первого взгляда выглядит, что эти принципы просты, всего-то 3 шага

Правила TDD - …

- Мало учят тому, как применять методику

Первым делом, я хочу немного усмирить ваш аппетит, и немного развеять простоту.

Техника сама по себе мало учит тому, как ее применять. Обычный подход при изучении чего-то нового состоит в простом следовании правилам, пока не набирается достаточно опыта, для понимания того, когда следует применять эти правила, а когда – нарушать.

Но у TDD настолько простые правила, что не понятно просто ничего.

TDD - это просто?

- Как протестить логику с БД?
- Как протестить UI сценарий?
- Как протестить код без загрузки множества зависимостей?
- Позитивный тест писать или негативный?

А вот как насчет вопросов, которые сразу возникают в голове разработчика, например

TDD - …

- Озвучивает идею, а не правила
- Заставляет изучать новое
- Хорош ровно на столько, насколько хорош разработчик
- Не может гарантировать отсутсвие багов

К сожалению TDD не дает ответа на предыдущие вопросы. У TDD есть идея, которая стоит за этими правилами, и того, что их использование главным образом зависит от контекста. Отсюда же происходит типовой ответ «а это зависит от ситуации»

TDD не является заменой думанию и не отменяет необходимость в навыках проектирования.
Естественно, чтобы покрыть тестом какой-то код, он должен быть изолируемым, компоненты должны быть слабосвязными. Вам необходимо уже уметь строить правильные абстракции, что требует глубоких знаний в области объектно-ориентированного дизайна, паттернов, принципов SOLID и DRY, и тому подобного. Пока не будут заполнены пробелы в этих знаниях, и разработчики не начнут писать слабосвязный код, применять TDD будет невозможно. Соответственно эта практика делает разработчика более квалифицированным.

В действительности, TDD ничего не делает самостоятельно – все делает разработчик, и эта техника хороша ровно настолько, насколько хорош сам разработчик.

И это не волшебная пилюля, если вы используете TDD, у вас все равно есть баги.

“Красный, Зеленый, Рефакторинг”

–TDD слоган

Основной слоган TDD гласит “Красный, Зеленый, Рефакторинг”. Что под этим подразумевается? Рассмотрим каждую фазу отдельно

Красная фаза

Шаг 1 - Получить задачу, ознакомиться с бизнес требованиями

Представим, что мы сейчас работаем с использованием TDD.
Первое, что нам нужно, это бизнес требования, описание задачи в джире

Красная фаза

Шаг 2 - Написать тест на требуемый функционал

Мы начинаем с написания несрабатывающего теста. Зачем?

Существующие тесты проходят нормально, значит, в коде нет известных проблем. Создав тест для выявления недостающего функционала, мы четко выявляем задачу, которую собираемся решить.

Фиксируя задачу и, следовательно, ее возможные решения мы делаем процесс работы над ней более простым.

Вопросы к функционалу

- В чем заключаются обязанности тестируемой системы?
- Какой API удобен для того, чтобы тестируемый код выполнял задуманное?
- Что нужно тестируемой системе для выполнения своих обязательств (данные, другие классы)?
- Что мы имеем на выходе и какие есть побочные эффекты?
- Как узнать, что система работает правильно? Достаточно ли хорошо определена ли эта «правильность»?

Посмотрим, какие вопросы возникают при создании самого первого, несрабатывающего теста

При создании кода все перечисленные вопросы когда-нибудь должны быть заданы, и сила TDD в предоставлении удобного инструмента для этого.

Вместо процесса поочередного решения вопросов с предотвращением потенциальных конфликтов по мере их возникновения, TDD позволяет задать абстрактный вопрос «Как мне написать следующий тест?» и в процессе его решения ответить на ряд конкретных вопросов. Вот почему TDD не лучше программиста – все равно придется искать ответы на те же самые ключевые вопросы, что требует соответствующих знаний и опыта. TDD просто облегчает поиск.

Красная фаза

- Четкая формулировка поставленной задачи
- Гарантия, что результирующий код покрыт тестом
- Наброска будущей архитектуры
- Быстрая, точная и честная обратная связь

Мы так же гарантируем, что результирующий код будет покрыт тестами на правильность реализации и защищен от регрессии (хотя этого можно достичь и без TDD с помощью любого достаточно глубокого автоматического теста).

Но самым важным здесь является создание наброска будущей архитектуры. Просто делается это не на доске для рисования (что имеет свои преимущества), а прямо в коде, что позволяет незамедлительно понять, насколько просто использовать и тестировать этот код.

Как и любая хорошая абстракция, он дает возможность рассматривать множество мелких составляющих как одно целое, при создании которого можно разом применить все имеющиеся технические и другие навыки. Кроме того, поиск ответа на этот единственный вопрос облегчает еще одно свойство TDD: предоставление быстрой и точно обратной связи с кодом.

Процесс создания теста дает всестороннюю оценку разрабатываемому модулю. Подготовка к использованию чересчур громоздка? Получается многовато сценариев использования или фикстур? Вероятно, тестируемый код имеет слишком много обязанностей. Трудно изолировать тестируемое поведение или непонятно, как его проверить? Наверное, у нас неправильное API или сама абстракция, и ее надо выделить как-то иначе.

У меня было много случаев, когда покрываешь новый функционал тестом, видишь какой функционал не удобный. Или апи тупое, или cli интерфейс ужасный.

С TDD эти проблемы становятся очевидными мгновенно. Самое дешевое время для исправления кода – до его написания.

Зеленая фаза

Шаг 3 - Пишем самый простой код, чтобы тест проходил

Затем мы пишем самый простой код, удовлетворяющий тесту. Максимально быстро пишем.

Зеленая фаза

- Фокус на маленьком участке кода
- Быстрая разработка, так как в голове все складно

Как ни странно, это наименее интересная часть TDD. Мы уже закончили всю тяжелую работу по выявлению требуемого поведения, и сейчас осталось только реализовать его.

Реализация слишком сложна? Тогда придется вернуться назад и изменить тест – мы только что узнали с помощью TDD, что попытались сделать слишком большой шаг в разработке модуля. Реализация тривиальна и имеет очевидные недоработки? Отлично, теперь мы знаем, каким будет следующий тест.

Рефакторинг

Шаг 4 - Пишем код production качества

Фазу рефакторинг ложно воспринимают, как просто какое-то улучшение. Это не так. Это обязательный шаг, а не опциональный. Здесь мы должны написать код, такой, чтобы коллеги не ржали на коде ревью.

Рефакторинг

- Пора взглянуть на картину целиком
- Пора убрать дублирование, улучшить дизайн классов, методов

Самое время охватить взглядом всю картину. Если было реализовано решение «в лоб», то можно избавиться от дублирования или выделить отдельный метод, чтобы код лучше описывал то, что он делает.

Еще более важным является выявление высокоуровневого дублирования – повторений не просто участков кода, а схожего поведения, которое может быть выделено в абстракцию или выведено на структурный уровень.

Заканчиваем работу

Шаг 5 - Получаем качественный код, покрытый тестом

Шаг 6 - Code review

Как ни странно, это наименее интересная часть TDD. Мы уже закончили всю тяжелую работу по выявлению требуемого поведения, и сейчас осталось только реализовать его.

Реализация слишком сложна? Тогда придется вернуться назад и изменить тест – мы только что узнали с помощью TDD, что попытались сделать слишком большой шаг в разработке модуля. Реализация тривиальна и имеет очевидные недоработки? Отлично, теперь мы знаем, каким будет следующий тест.

На этапе Code review другим разработчиком гораздо легче понять, что вы сделали, прочитав ваш тест. В результате качество ревью тоже растет.

Мифы

- TDD - это только юнит-тесты
- Нельзя писать код, а потом тест

Нигде не написано, что TDD основывается исключительно на юнит-тестах, но большинство новичков думают, что процесс основан именно на них, и из этого вытекает дальнейшее непонимание.

TDD обеспечивает обратную связь с кодом. Когда вы уже знаете, что именно нужно (например, если работа заключается в дополнении мелкими частями готовой архитектуры), та часть TDD, что отвечает за проектирование, не дает много пользы.
Единственное преимущество предварительного написания тестов здесь в том, что можно убедиться, что изначально они не срабатывают; это способ протестировать сами тесты. В этом случае подход «сначала тесты» является техникой создания кода, покрытого тестами, а не проектирования.

Реальные проблемы

- У нас связный код, я не могу изолировать свою задачу
- Я застрял в тестах, потратил кучу времени
- Я не понимаю как начать, что именно мне нужно
- Я не могу тратить время на “подумать”, продакт стоит над душой, ждет фичу

Ни на одну из этих проблем нет однозначного ответа. Но команда вместе в силах решить такие проблемы. Рассмотрим все по-очереди.

Чуваки, так а кто такой код написал? Любой код можно отрефакторить, соберитесь командой, запланируйте такие работы. Всегда можно найти решение. Главное не откладывать на потом, и не говорить, что мы вот-вот разберемся с багами и начнем писать. тесты. Нет, не начнете, их надо начать писать вчера. Современная IT индустрия требует качественного софта на рынке. Пока вы не начнете знакомиться с проблемами вашего проекта, он не станет лучше.

Если вы застряли, опять же обсудите с коллегами. Попробуйте поработать, используя покрытие тестами более высокого уровня абстракции. Просто попробуйте другие подходы. TDD является хорошим, но не единственным средством ограничить проблему и выйти на решение.

Сделайте то, что необходимо прямо сейчас, чтобы решить задачу, но не забудьте вернуться и попытаться понять причину остановки позже. Уловить разницу не получится, если сдаться слишком быстро. Лично меня следование этому принципу привело в свое время к открытию таких штук как мокирование, DI-контейнеры, BDD и т.д. (и я все равно еще далек от полного отсутствия пробелов в знаниях).

Здесь нет единственного верного ответа. Главная цель — получение обратной связи с кодом, над которым идет работа в данный момент. Разнообразные задачи создают разные проблемы проектирования и требуют разнообразных видов обратной связи. Использование TDD специфично для каждой задачи (и для каждого разработчика). Нужно найти метод, который дает необходимую обратную связь и обеспечивает развитие архитектуры. Некоторые проблемы хорошо решаются с помощью тестов «от края до края», с использованием реальных внешних баз данных или сервисов, некоторые нет.

На хабре есть статья, TDD, как катание на сноуборде. Начав использовать TDD, вы можете обнаружить, что работаете медленнее, чем обычно, — это в основном из-за того, что вы будете работать вне «зоны комфорта», чувствуя себя неестественно. После того как вы ощутите, что написание тестов стало

MVP продукт

На входе:

- Высокоуровневые требования от продактов
- Сжатые сроки
- Большие надежды
- Неопределенное будущее

Все слышали на вебтолкс, что в этом году будет запущен ряд продуктов. Представим себя на месте команды этих проектов. На входе мы имеем …

MVP продукт

- Понятно, какую систему надо стоить
- Не понятно, как строить систему

TDD и MVP продукт

- Последовательная разработка дизайна кода каждого компонента вашей системы

TDD может помочь последовательно разработка дизайна каждого компонента вашей системы

Когда речь заходит о сроках и скорости, можно и нужно объяснять менеджменту, что мы не будем говнокодить с креативной скоростью. Подробнее - в начале книги Совершенный код.

Выводы о TDD

- Инструкции просты, но использование сложно
- Идея важнее алгоритма
- Техника для обучения проектированию софта

Хотя инструкции процесса TDD просты, сама техника таковой не является. Инструкции обеспечивают лишь самую минимальную помощь на старте, в то время как шумиха вокруг TDD формирует совершенно нереалистичные ожидания.

Мы так же не будем впадать в отчаяние в случае невозможности применения правил TTD, так как знаем, что эта техника всего лишь является инструментом проектирования, и всегда найдутся пробелы в знаниях или недостаток опыта, которые придется заполнить перед тем, как станет возможным решать новые задачи.

По существу, TDD дает отличный способ продумывать и итеративно развивать дизайн приложения.

Проектировать всегда трудно. TDD позволяет сконцентрироваться на дизайне и понять, как сделать его лучше.

Рекомендуемая литература

- Кент Бек - Экстремальное программирование, разработка через тестирование
- Стив МакКоннелл - Совершенный код

Спасибо за внимание!

a.galitstyn@2gis.ru

Вопросы