micb
13 дек 2017 в 13:26

Разработка через приемочные тесты (ATDD). Что это такое, и с чем его едят

9 мин 34K

Тестирование IT-систем*, Анализ и проектирование систем*, Управление разработкой*, Развитие стартапа,

Разработка через тестирование (TDD) – отличный способ повысить качество и надежность кода. Этот же подход может быть распространен и на разработку требований. Он называется "Разработка через приемочные тесты" – acceptance test driven development (ATDD). Сначала я присматривался к этому подходу, потом пробовал применить, потом долго тюнинговал, чтобы приспособить его под мои нужды, и теперь хочу поделиться мыслями. И для себя еще раз разложить все по полочкам.

В этой статье я расскажу небольшое введение в тему. Пример будет совсем простой и скорее для иллюстрации. А в следующей статье постараюсь поделиться историей, как я применял ATDD на практике при разработке настоящей фичи в реальном продукте.

Вместо введения

Когда я работал программистом в аутсорс компании на один банк, то мне приходилось изучать спецификации требований и оценивать трудоемкость задач. Оценивать нужно было как можно точнее, мы работали по модели оплаты за проект (Fixed Price), и все промахи в сроках были на нашей стороне и не оплачивались. Каждый раз, когда я читал спецификации, мне было все понятно, я не замечал в них нелогичные моменты, упущения, странности. Но как только начиналась разработка, то все косяки требований вылезали наружу, и было удивительно, как я их пропустил в начале. Несмотря на все усилия, я так и не мог придумать способ, как читать спецификации и находить в них проблемы до реализации.

Потом я перешел на работу в крупную компанию, которая занималась разработкой большого и сложного коробочного продукта, над которым работала огромная команда. Аналитики общались с партнерами и клиентами и записывали их пожелания. Потом эти спецификации, прежде чем быть взятыми в работу, проходили процедуру ревью, в которой участвовали и разработчики. Чтобы не тратить время на самой встрече, надо было сначала прочитать требования и подготовить вопросы. Как и в предыдущем проекте, большинство вопросов к содержимому документов возникали позднее – во

время разработки, а не тогда, когда должны были возникнуть, то есть на этапе ревью.

Затем я ушел в свой стартап. Естественно, там не было никаких аналитиков, спецификаций и ревью. Обратную связь мы получали от пользователей в виде имейлов или звонков, тут же превращали это в фичи и включали в план разработки. Несмотря на отсутствие задокументированных требований, все равно приходилось оценивать трудоемкость задач. То, что на первый взгляд казалось очень простым, на деле становилось головной болью и наоборот. При быстром переключении контекста с одной проблемы на другую, уже реализованные решения вылетали из головы и становилось все труднее и труднее совмещать их в одном продукте. Нам было нужно какое-то подобие технической документации, тестпланов и требований. И чтобы стоило недорого.

Что такое ATDD

Acceptance test driven development (ATDD) является развитием идеи test driven development (TDD). Общий смысл в том, что прежде чем что-то делать, надо придумать критерий выполненной работы и критерий того, что работа сделана правильно. Почему это важно? Потому что эти критерии позволяют на самом раннем этапе понять, что именно требуется сделать, как это сделать, что именно считать хорошим результатом. Т.е. не выяснять детали по ходу дела, строя прототипы, а сразу приближаться к цели, так как цель уже определена, причем вполне формально.

Эти критерии описываются на понятном заказчику языке в виде готовых сценариев. Сценарии моделируют то, как проектируемая фича будет использоваться в дальнейшем. Если сценарий реализован и ожидаемый в нем результат может быт получен на практике, значит задача решена корректно и работу можно считать выполненной. Набор таких сценариев и называется приемочными тестами. Приемочные тесты фокусируются на поведении системы с точки зрения человека, а не на внутреннем устройстве и на технических деталях реализации.

Несмотря на общее название, этот подход относится ко вполне определенной части процесса – той, где происходит разработка требований и их формализация в спецификации. В данном процессе часто участвуют люди как со стороны бизнеса, так и с технической стороны, т.е. люди, обладающие разными компетенциями и взглядами на мир. Заказчики на интуитивном уровне понимают, что именно они хотят видеть в продукте, но сформулировать и перечислить требования кратко (и полно) могут далеко не все. Разработчики (представители исполнителя), в свою очередь, часто не знают, что именно забыл рассказать заказчик, и как это выяснить.
Для решения этих задач используется фреймворк Given – When – Then.

Фреймворк Given – When – Then

Смысл приемочного теста — показать, что произойдет с системой в конкретном сценарии. Для этого в сценарии должно быть описано, как выглядит система в начале теста, затем описывается какое-то действие, которое эту систему меняет. Это может быть воздействие снаружи, а может быть и какой-то внутренний триггер. В результате система немного меняет свое состояние, что и является критерием успешности. Важный момент: система рассматривается как черный ящик. Другими словами, формулируя тест, мы не знаем, как система устроена внутри и с чем она взаимодействует снаружи. Тут есть одна особенность. Иногда изменение системы недоступно для непосредственного наблюдения. Это означает, что саму приемку провести не получится. Выхода тут два — либо попытаться определить состояние косвенно, через какие-то соседние признаки, либо просто не использовать такой тест. Примером могут быть изменение полей в таблицах БД, отложенные изменения в каких-то недоступных файлах и т.д.

В юнит тестах используется шаблон Arrange – Act – Assert (AAA). Это означает, что в тестах должны быть явные части, отвечающие за подготовку данных — arrange, само действие, результат которого надо проверить – act, и собственно проверка, что реальность совпала с ожиданиями – assert. Для приемочных тестов используется подход Given – When – Then (GWT). Суть та же, только с другого ракурса.

- Given описывает что «дано», т.е. состояние системы в начальный момент времени
- When задает непосредственно триггер, который должен привести к результату. Чаще всего это какое-то действие пользователя.
- Then определяет результат этого действия, т.е. является критерием приемки.

Given часть может содержать в себе как одно, так и набор состояний. В случае, когда их несколько, эти состояния должны читаться через "И". Объединять какие-то состояния через "ИЛИ" можно, но тогда это будут два разных теста. Такое возможно для упрощения записи. Я рекомендую избегать этого до того момента, как все возможные комбинации состояний не будут описаны. Тогда можно быть уверенным, что ничего не забыто и слить несколько тестов в один для упрощения чтения и понимания. Это же справедливо и для Then — исходов, которые надо проверить может быть несколько. When должен быть один. Взаимовлияния триггеров лучше избегать.

GWT тесты вполне можно читать вслух: "Пусть (given) A и B, и C. Когда (when) случается D, то (then) получается E и F.". Их вполне можно использовать для документации или формулирования требований. Когда я говорю "читать", я не имею ввиду, что именно так

они и должны быть записаны. В реальности такие тесты получаются очень масштабными. Если их записать простым текстом, то потом взглянуть на них системно очень тяжело. А без системы легко пропустить какие-нибудь важные сценарии.

Очень важный момент: формат записи нужно выбирать тот, который наиболее подходит к вашей задаче, с которым удобнее работать. Никаких ограничений тут нет. Given, when, then — это общая структура записи, то есть то, что обязательно должно быть в тесте, а непосредственное представление может быть любым – хоть предложения, хоть таблицы, хоть диаграммы.

ATDD не диктует правила, а предоставляет фреймворк для того, чтобы составить свою спецификацию через примеры. Есть модель черного ящика, GWT и их комбинирование. Все остальное является применением этих механизмов на практике, часть из которых можно считать устоявшимися.

Пример

Для примера возьмем что-нибудь простое и понятное, например, светофор. Как можно описать требования к разработке светофора с помощью GWT нотации? Для начала нужно понять, что именно в светофоре можно назвать Given, что является When, а что Then.

За состояние светофора можно принять информацию о том, какая секция сейчас горит. Светофор переключается (меняет состояние) через какие-то промежутки времени. Значит триггером является таймер, точнее, срабатывание таймера. Результатом срабатывания триггера является переход в одно из состояний. Т.е. можно считать, что в примере со светофором Given и Then – один и тот же набор:

1. Горит красный
2. Горит красный и желтый
3. Горит зеленый
4. Зеленый мигает
5. Горит желтый

Опишем поведение светофора в нотации GWT:

1. Пусть горит **Красный**. Когда таймер срабатывает, тогда светофор переключается в режим одновременного **Красного и Желтого**.

2. Пусть горит **Красный и Желтый**. Когда триггер срабатывает, тогда светофор переключается в **Зеленый**.
3. Пусть горит **Зеленый**. Когда триггер срабатывает, тогда светофор переключается в **Зеленый мигающий**.
4. Пусть **Зеленый мигает**. Когда триггер срабатывает, тогда светофор переключается в **Желтый**.
5. Пусть горит **Желтый**. Когда триггер срабатывает, тогда светофор переключается в **Красный**.

Вот 5 сценариев, прочитав которые, можно понять, как работает светофор. Естественно, у светофора есть еще куча режимов, например, режим желтого мигающего (когда он неисправен), или ручной режим управления (например, в случае ДТП) и т.д. Но не будем усложнять иллюстрацию.

Описывать тесты словами мне кажется избыточным. Тем более, что меняется в них только название цвета. Тут лучше подойдет диаграмма состояний или простая таблица:

	Given	When	Then
1	Красный	Таймер	Красный + Желтый
2	Красный + Желтый	Таймер	Зеленый
3	Зеленый	Таймер	Зеленый мигающий
4	Зеленый мигающий	Таймер	Желтый
5	Желтый	Таймер	Красный

Пример показывает один из основных преимуществ приемочных тестов: они позволяют общаться с бизнес пользователями практически на их языке. Приятным бонусом идет готовый набор сценариев для тестирования и последующей автоматизации.

Обеспечение полноты

Нотация Given — When — Then структурирует процесс составления тестов и дает уверенность в том, что тесты описывают все аспекты поведения системы. Не нужно сидеть и постоянно спрашивать себя: "А какой сценарий я еще не описал?".

Итак, алгоритм такой:

1. Определить все состояния, которые могут быть заданы, т.е. все Given.
2. Определить все триггеры, т.е. When.
3. Определить все Then, что именно может произойти.
4. Теперь эти списки надо комбинаторно перемножить.
5. В результате получается набор тестов.

На каждом из этих этапов требуется участие заказчика или человека, который играет его роль, потому что именно он лучше всех представляет, что и как в итоге должно работать.

Почему полезно

Как уже было сказано, подобный подход, несмотря на свою избыточность, дает уверенность в том, что ни один из сценариев не будет пропущен. Это, пожалуй, главное преимущество такой формализации. Зачастую бизнес-пользователь видит процесс только в общих чертах и ему не видны детали. Я уверен, что вы постоянно слышите от заказчика или даже аналитика фразы типа: "Нам нужна такая-то фича, я все придумал, вот, смотри картинку", или "Тут нам нужна такая-то кнопка, у нас уже есть похожая функциональность в другом месте, сделай как там". Если до того, как начать разработку, сесть и прикинуть возможные варианты развития событий, то сразу всплывет очень много деталей, в которых, как известно, и кроется дьявол.

Подобный подход так же полезен и в случае, когда от аналитика приходит спека и ее нужно прочитать, дать свои оценки сложности и трудозатрат. При прочтении все детали ускользают, но если по ходу чтения вести конспект по форме GWT, то сразу становится понятно, какие сценарии плохо или неточно покрыты в требованиях и требуют уточнений.

Помимо анализа требований с целью разработки решения, GWT сценарии можно применять и для сбора требований. Предположим, что есть какая-то функциональная область и человек, который в ней разбирается, но время на общение с ним очень ограничено. Если подготовиться заранее и разобрать сценарии с помощью GWT фреймворка, то на самом интервью нужно будет узнать только то, что мы ничего не забыли из раздела Given, When и уточнить, что именно должно быть в разделе Then.

Есть специальные инструменты для автоматизации GWT сценариев, записанных в том

числе и на естественных языках. Пример — cucumber. Я с ними не работал, поэтому ничего кроме факта их существования рассказать не могу.

Подводные камни

Обратная сторона мощности GWT — избыточность. Предположим, что вы определили N штук given, M штук when и K штук then. В худшем случае количество тестов будет огромным – N *M* K. И с этим надо как-то жить. Это верхняя оценка сложности; в реальности далеко не все эти сценарии будут осуществимы, а часть из них будет дублировать друг друга, а еще часть можно пропустить ввиду низкого приоритета или очевидности.

Вторым недостатком можно указать формат. По моему опыту могу сказать, что GWT записи всегда стремятся к минимализму. Во время их разработки не хочется тратить времени на детальные описания, потому что зачастую сценарии похожи друг на друга. В результате получается тяжело читаемая структура. После некоторого перерыва для ее понимания приходится восстанавливать контекст, и заново вспоминать условные сокращения и записи. Это также затрудняет задачу передать кому-то документ для ознакомления, так как, скорей всего, для его прочтения потребуется сам автор.

Следующий недостаток объясняет, почему ATDD скорее относится к области формализации требований с бесплатным бонусом в виде тестовых сценариев, а не собственно тестирования. Такие сценарии не могут описать композитные (большие и сложные) сценарии. Тестирование идеального черного ящика в первую очередь основано на аксиоме его идеальности. В реальности ящики черными не бывают, они всегда взаимодействуют с чем-то снаружи себя, являясь при этом частью более сложной системы — продукта. Легко можно переусложнить требования, если попытаться включить в один документ сразу все связи внутри продукта. Такой набор приемочных тестов будет настолько огромным и сложным, что мало чем сможет помочь. Поэтому, в реальной жизни сквозные сценарии в качестве приемочных тестов не применяются.

Немного исторической справки

Если верить Википедии, то идея формулировать спецификации через конкретные сценарии была впервые описана Ward Cunningham в 1996 году, а сам термин specification by example ввел Martin Fowler в 2004 году. Дальнейшее развитие идеи формулируется в книге "Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing" от Gojko Adzic 2009 года. В 2011 он же выпустил еще одну книгу на эту тему: "Specification by Example: How Successful Teams Deliver the Right Software". Рекомендую эти книги для обращения к первоисточнику.

Теги: acceptance testing, ATDD, software development, requirements, разработка по, разработка требований

Хабы: Тестирование IT-систем, Анализ и проектирование систем, Управление разработкой, Развитие стартапа, Управление продуктом

27 Карма 0 Рейтинг

Михаил Белов @micb
Пользователь

Комментарии 5

Публикации

ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ

yard
12 часов назад

Неужели Banki.ru сливают ваши данные спамерам? Или как не угодить в ловушку микрозаймов

Средний 6 мин 8.9K

Кейс

+83 34 120

finindie
7 часов назад

Будет ли пенсия у родившихся в восьмидесятых?

Простой 10 мин 9K

+60 36 113

GeeksCat
12 часов назад

Blade Runner 2049 — это экранизация Набокова

Простой 8 мин 3.9K

Мнение

+39 19 21

Kopcheniy
17 часов назад

Эй конвертер, не шипи! Строптивые преобразователи платы для E-Ink экранов (Ч2)

Средний 12 мин 2K

Кейс

+39 19 1

Tirarex
16 часов назад

Идеальный компьютер, который мы потеряли: ретроспектива на Intel Compute Stick первого поколения

Простой 13 мин 4.6K

Обзор

+37 14 20

DAN_SEA
12 часов назад

Range Extender на NRF24L01+PA+LNA: обмен текстовыми сообщениями между устройствами там, где нет сотовой связи

Простой 11 мин 2.3K

Обзор

+34 37 21

ilkhom_cinecast
19 часов назад

Как мы снимали премиальные шины из багажника

7 мин 3.2K

Кейс

+25 16 3

artvi
19 часов назад

В поисках ПАК: импортозамещаем немецкое «железо» в российском ЦОД

Средний 5 мин 2.6K

Обзор

+24 14 3

fellow_pablo
13 часов назад

Эргономика рабочего места инди-разработчика, или как я избавился от боли в спине

Простой 5 мин 3.5K

Кейс

+23 27 24

Ciberst
15 часов назад

Система мета-сборки GN: краткий обзор и подходы

Средний 11 мин 485

Обзор

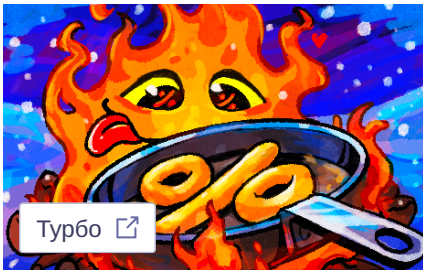
+19 9 1

Путь из бизнес-аналитика в программного роботизатора

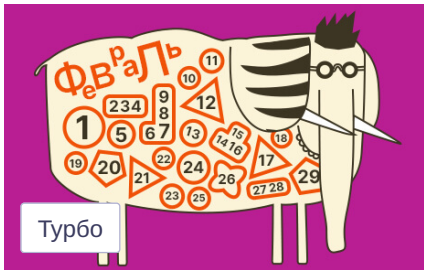
Турбо

Показать еще

МИНУТОЧКУ ВНИМАНИЯ



Топим снег скидками: промокодус в деле



Хабракалендарь, отворись! Какие IT-ивенты ждут нас в 2024

Глупым вопросам и ошибкам — быть! IT-менторство на ХК

СРЕДНЯЯ ЗАРПЛАТА В IT

Кошелек Дурова, или Telegram, который становится первым мировым супераппом

54K 118

Неужели Banki.ru сливают ваши данные спамерам? Или как не угодить в ловушку микрозаймов

8.9K 120

Где изучать Python в 2024. Бесплатные курсы, книги и ресурсы

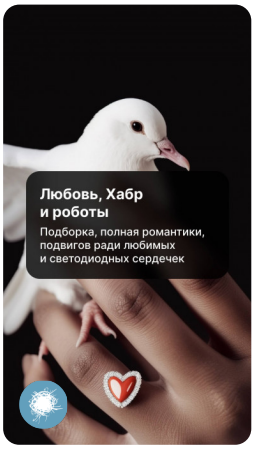
2.1K 0

Путь из бизнес-аналитика в программного роботизатора

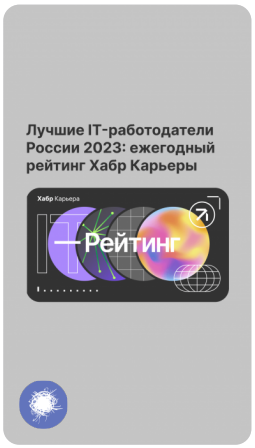
Турбо

ИСТОРИИ

Наука сна



Любовь, Хабр и роботы



Рейтинг IT-работодателей 2023

Прокачать коммуникацию

Перевернуть календарь и добавить событ

РАБОТА

Руководитель разработки
144 вакансии

Системный аналитик
583 вакансии

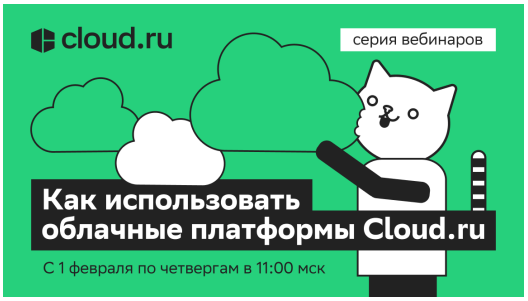
Тестировщик программного обеспечения
57 вакансий

Продакт менеджер

22 вакансии

Все вакансии

БЛИЖАЙШИЕ СОБЫТИЯ

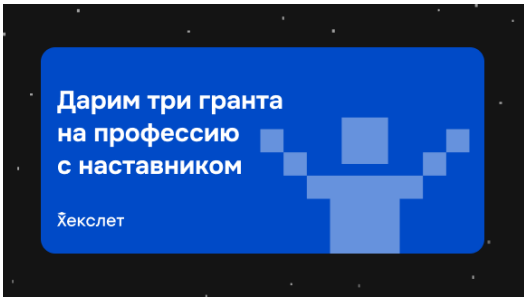


Как использовать облачные платформы Cloud.ru: серия демо-встреч по четвергам

1 – 29 февраля 11:00

Онлайн

Подробнее в календаре

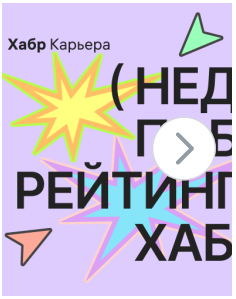


Конкурс грантов на обучение IT-профессиям от Хекслета

5 – 29 февраля

Онлайн

Подробнее в календаре



Неделя поб рейтинга Ха

12 – 18 фе

Онлайн

Подробнее в кал

Реклама

Ваш аккаунт

Войти
Регистрация

Разделы

Статьи
Новости
Хабы
Компании
Авторы
Песочница

Информация

Устройство сайта
Для авторов
Для компаний
Документы
Соглашение
Конфиденциальность

Услуги

Корпоративный блог
Медийная реклама
Нативные проекты
Образовательные программы
Стартапам

Настройка языка

Техническая поддержка