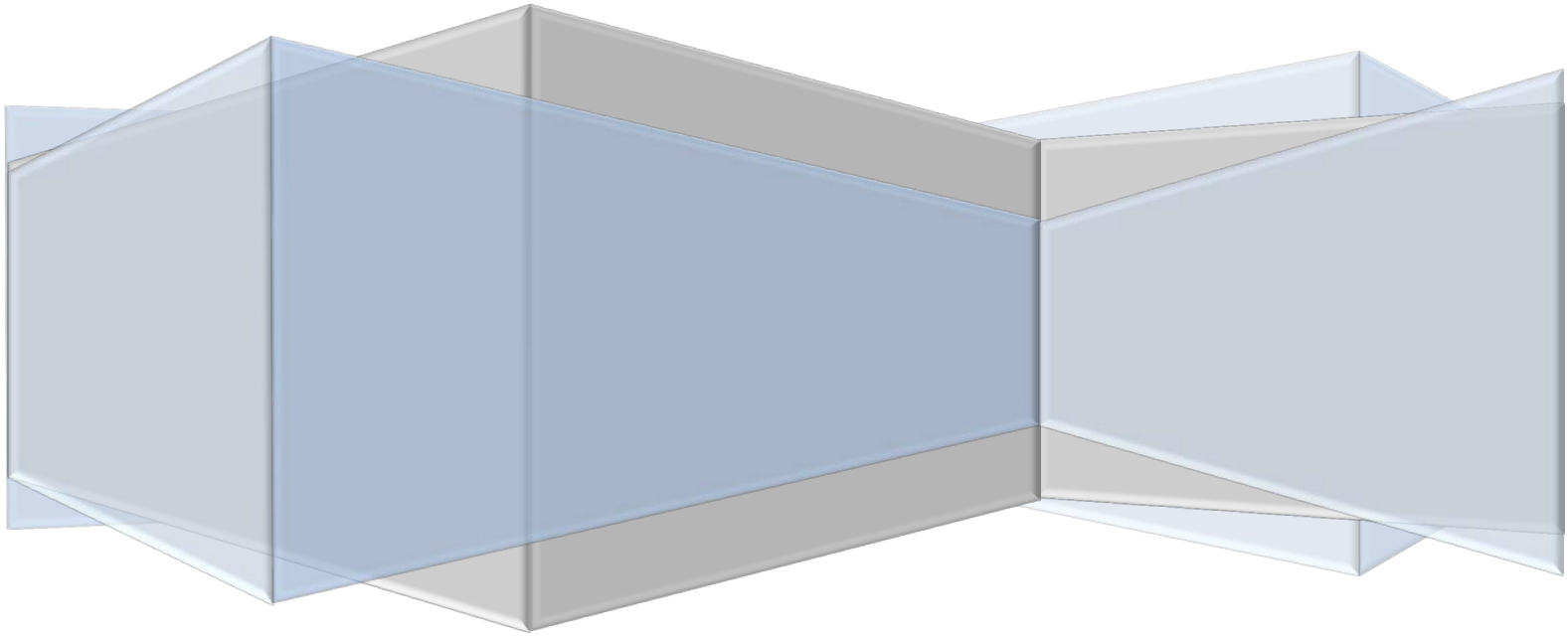
Записная книжка тест-дизайнера

Рикард Эдгрен

Оригинал книги



Перевод – Ольга Алифанова
http://software-testing.ru

Содержание

Введение

Эта книга описывает способы извлечения пользы из множества источников информации, необходимых для энергичного системного тестирования.

Я начал разбираться в этом вопросе глубже, когда в сто пятый раз ощутил, что существующие техники тест-дизайна не отражают мой подход после десятилетней работы с одним и тем же набором продуктов. Это вылилось в расширенную генерализацию идей, о которых я узнал благодаря коллегам[1].

Я собираюсь описать комплексное слияние тест-стратегии, тест-анализа, тест-дизайна и выполнения тестов – задачи, которые теоретически можно разделять, но на практике они переплетены. Книга не рассказывает о деталях отдельных тест-кейсов – тест-дизайн может выражаться в однострочных записях, чартерах или ментальной модели в вашей голове. Я пишу о том, как справиться со сложностями продукта, развивать навыки и обучаться по ходу дела.

Я хочу сделать упор на результатах тестирования, на широкой выборке, делающей возможными озарения [2]и нацеленной на полноценное освещение [3]важных областей. Возможно, книга лучше всего подойдет ручным тестировщикам, которые концентрируются на важных проблемах – людям, которые хотят выяснить важную информацию, а не удовлетворяются соответствием или несоответствием требованиям.

Книга рассказывает про методы тест-дизайна, которые долгое время применяются как мной, так и многими другими тестировщиками[4].

Суть

Учитесь, используя множество источников, генерируйте релевантные тест-элементы, синтезируйте достойные тест-идеи, и выполняйте тесты, не забывая про озарения.

Тестировщики могут следовать идеям социальной науки Grounded Theory: мы должны пользоваться множеством источников информации о субъекте (требования, спецификации, прототипы, код, баги, каталоги ошибок, информация службы поддержки, рассказы заказчика, ожидания пользователей, технологии, инструменты, модели, системы, цели качества, атрибуты качества, тест-техники, тест-идеи, разговоры с людьми, риски, возможности); мы должны уметь группировать и подходить к задачам креативно, и создавать теорию, состоящую из ряда идей, отражающих то, что важно протестировать, совместно с результатами.

Я ратую за то, чтобы тестировщики не ограничивались требованиями, понимали необходимость комбинирования разноплановой информации, смотрели как на целое, так и на детали одновременно. Эта книга предназначена для серьезных проектов в несколько релизов, и не имеет смысла в ситуациях, когда

---

[1] Я учился у всех своих коллег, но особенно у Хенрика Эмилссона и Мартина Янссона. Библиография отражает наиболее важные моменты. Хочется также поблагодарить рецензентов, которые потратили свое время на то, чтобы сделать ценные замечания: это Роберт Бергквист, Лиза Криспин и Мэтью Хоссер.

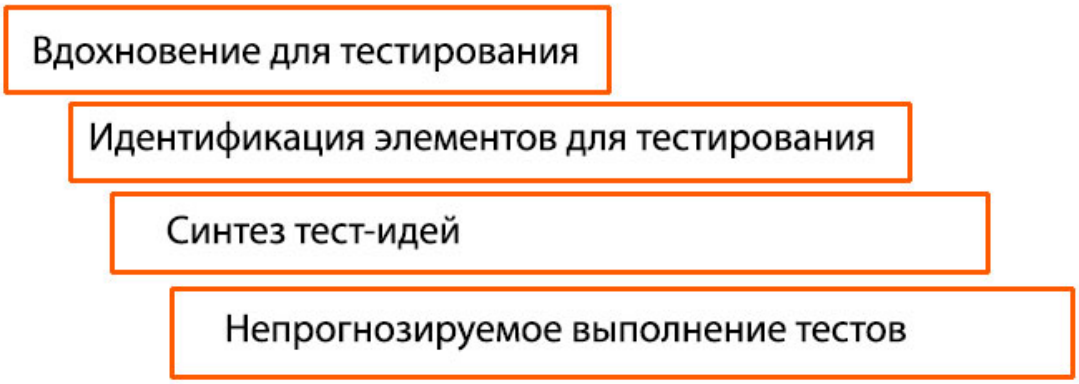
[2] Концепция "озарения" набирает популярность в тестировании. Впервые, насколько я знаю, об этом написал Джонатан-Бах в статье Session-Based Test Management – журнал "Software Testing and Quality Engineering", 11/00, http://www.satisfice.com/articles/sbtm.pdf. Также см. Rikard Edgren, *Testing is an Island, A Software Testing Dystopia*, EuroSTAR conference 2008 http://thetesteye.com/papers/redgren_testingisanisland.pdf.

[3] Об освещении областей - Rikard Edgren, *Is your testing saturated?* - http://thetesteye.com/blog/2010/04/is-your-testing-saturated/

[4] О работе тестировщика написано много статей, но Фиона Чарльз затронула основные моменты в статье *Modeling Scenarios using Data*, http://www.quality-intelligence.net/articles/Modelling%20Scenarios%20Using%20Data_Paper_Fiona%20Charles_CAST%202009_Final.pdf

ресурсы сильно ограничены, или когда вы абсолютно уверены, что техники, которыми вы пользуетесь – это идеальный выбор.

Это может показаться сложным, однако люди наделены феноменальными способностями.



Эти задачи зависят от навыка тестировщика – навыка, для выработки которого нужно время, а в лучшем случае – знание продукта и контекста проекта. Развитие этого навыка ускоряется при создании собственных "правил буравчика" и специфических эвристик тест-дизайна.

Grounded Theory

С моей точки зрения, за научную теоретическую базу тест-дизайна можно взять Grounded Theory [5](Корбин и Страусс), которая используется в социальных науках как качественный тщательный анализ феномена.

Ученые внимательно исследуют тему, собирают множество материала, документируют коды и концепции в заметках, комбинируют их в категории, и так рождается теория.

В отличие от традиционных естественных наук, тут нет гипотезы, от которой можно оттолкнуться, и это очень схоже с тестированием – было бы опасным полагать, что мы уже знаем все важные вопросы. Концепция Grounded Theory признает, что у нас есть проблема с выборкой, и что случайные счастливые открытия – это фактор преимущества. Такие теории продолжают развиваться, пока не достигают точки, в которой новая, меняющая что-то информация уже не находится – тогда теория насыщена.

Мне кажется, это хороший аналог тестирования, особенно потому, что он настаивает на креативном подходе исследователя.

Заимствуя идеи из другой области, нужно обдумать, нет ли у тестирования особых аспектов, делающих эти идеи неподходящими. Grounded Theory – это метод исследования, нацеленный на доказательство чего-либо, и все должны быть способны прийти к тем же самым выводам. Интервью в нем применяются чаще и шире, нежели в тестировании.

Я не пробовал применять GT к тестированию – она послужила источником вдохновения для описания моего тест-дизайна.

[5] Juliet Corbin and Anselm Strauss, *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, Second Edition*, SAGE Publications, Inc., Thousand Oaks 1998

Теория тест-дизайна

В отличие от традиционной концентрации на дизайне тест-кейсов, тест-дизайн в этой книге включает в основном тест-стратегию, тест-анализ, тест-дизайн и выполнение тестов, но не рекомендует разделять эти виды деятельности. Благодаря более комплексному подходу в голове тестировщика, одновременно размышляющего о разных вещах, рождаются интересные комбинации и возможности взаимодействия. Эти виды деятельности влияют друг на друга.

Пример соответствующей высокоуровневой тест-стратегии:

> *Мы создаем готовый продукт, который довольно дорог. Пользователи ожидают, что смогут пользоваться продуктом со своими данными, в своем окружении, и согласно своим нуждам. Мы бы хотели избежать патчей и прочих проблем, вызывающих имиджевые потери.*
>
> *Мы также хотим, чтобы вы помогали ускорить работу разработчиков; создавали для них легкие мишени.*

Низкоуровневые тест-стратегии (да, их множество) естественным образом придут к вам в процессе вашего изучения продукта, но вам, возможно, стоит проверить явные требования и провести чартерное или свободное исследовательское тестирование.

Тест-анализ – необыкновенно слабо проработанная область в литературе по тестированию. Думаю, это связано с тем, что традиционное/церемониальное тестирование в нем не нуждается – то, что должно быть протестировано, уже "полностью" описано в требованиях. Еще это связано с тем, что "*большая часть опубликованных руководств по тест-техникам основывается на юнит- и компонентных техниках*[6]". Это печально, так как куда более важные тест-решения принимаются, когда вы выбираете, какие тест-элементы учитывать и до какой степени; когда вы пытаетесь найти хорошие способы исследования какой-то области, и дизайн-методы для выявления важной информации о ПО с точки зрения пользователя. В то же самое время в этих процессах заключена самая суть исследовательского тестирования[7], и я тоже больше заинтересован в тестировании, а не в проверках[8].

Изобретение интеллектуальных способов тестирования на основании множества источников информации – это та часть тест-дизайна, которая завораживает меня больше всего.

Мне неинтересно создавать тест-кейсы – я считаю, что лучше создавать тест-идеи, как минимум на уровень выше тест-кейсов. Детали выполнения конкретных тестов часто отдаются на откуп тестировщику. Ожидаемый результат необязателен – зачастую мы не знаем, что планируем найти, и отчитываемся обо всем, что стоит упоминания.

В отличие от множества существующих техник тест-дизайна, мой подход концентрируется на поиске релевантных источников, их анализе, выборе стратегии и техники, а не на проработке деталей тест-кейса.

Мой подход частично схож с "предположением об ошибках" или "основанном на опыте" методом, но в основном он близок к исследовательскому тестированию, которое признает, что сам продукт и результаты его тестов должны использоваться для создания последующих тестов, если это имеет смысл. Если вы ищете

---

[6] Neil Thompson & Mike Smith, *The Keystone to Support a Generic Test Process: Separating the "What" from the "How"*, http://doi.ieeecomputersociety.org/10.1109/ICSTW.2008.46

[7] Основы исследовательского тестирования описаны в курсах Кема Кейнера и Джеймса Баха *Black Box Software Testing, Exploratory Testing*, http://www.testingeducation.org/BBST/exploratory/BBSTExploring.pdf, весна 2006.

[8] См. цикл статей Майкла Болтона, начиная с with *Testing vs. Checking*, http://www.developsense.com/blog/2009/08/testing-vs-checking/

информацию, используя множество источников, то ваша цель – обучение, а это ключевая часть исследовательского тестирования.

Возможно, этот текст – еще одна заявка на описание ключевой области исследовательского тестирования, однако этот тест-дизайн можно с тем же успехом использовать для более сценарного подхода, если тестировщик не хочет ограничиваться исключительно требованиями и спецификацией.

Важные моменты

Мы знаем, что полное тестирование – утопия, потом что даже если вы запустили весь код и все функции для всех атрибутов качества, вам не удастся проделать это со всеми возможными данными возможных пользователей, и учесть всех их нужды, окружения, и ощущения. Это означает, что у нас есть проблема выборки, и что мы создаем выборку, ориентируясь в основном на важные, значимые моменты.

Тест-дизайн должен порождать тест-идеи, которые при их воплощении (с озарениями в процессе или без них) покроют большую часть того, что важно (требования будут неплохим стартом). Это очень сложно – в ПО значимо множество вещей, и поэтому "*Тестирование – это невероятно креативная и интеллектуально сложная задача*"[9].

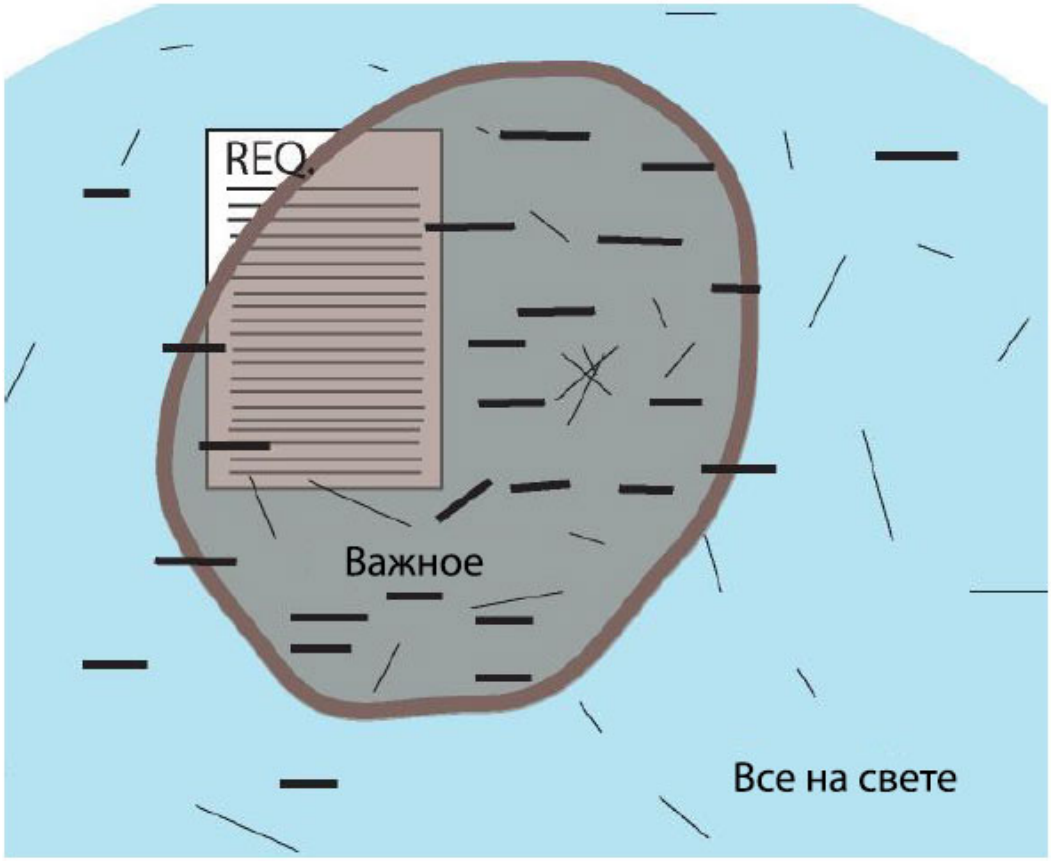
Я убежден, что в отдельных ситуациях вы или знаете, что важно, или нет. Поэтому базовая стратегия – это учиться, учиться, и еще раз учиться, и тогда вы будете знать, что же тут важно, чаще, чем не знать.

Ваш навык понимания того, что важно, будет развиваться со временем, а движим он любопытством и сотрудничеством. Если вы долго работаете над продуктом, используйте это как преимущество, изучая различные способы, которыми этот продукт приносит пользу – это долгое и веселое образовательное приключение. В пути вам помогает доменное знание, благодаря контактам с конечными пользователями концепции обретают плоть и кровь, воображение позволяет конструировать жизнеспособные сценарии и новые взаимодействия, критическое мышление помогает заметить пробелы, изучение технологий – понять, что можно сделать с их помощью (и что должно быть сделано). Все это – составные части мышления тестировщика[10].

[9] Страница 19 в Glenford Myers, *The Art of Software Testing*, Second Edition, John Wiley & Sons, Inc., Hoboken 2004 (впервые опубликована в 1979)

[10] Хорошее описание мышления тестировщика – в статье Майкла Болтона *Why Do Some Testers Find The Critical Problems?*, http://www.developsense.com/blog/2011/02/why-do-some-testers-find-the-critical-problems/

Тест-пространство можно рассматривать, как бесконечность, в которой находится коробка требований и "картошка[11]" продукта:



Располагая обоснованными знаниями о продукте и его контексте, вы можете варьировать свои действия, чтобы они привели к находке важной информации.

Чтобы распознавать важность, нужно принимать оценочные решения. В традиционном тестировании используется чересчур много количественных подходов, пришедших из естественных [12]наук. Компьютеры плохо разбираются в том, что важно – в этом хороши люди.

[11] См. статью Рикарда Эдгрена *In search of the potato…*, http://thetesteye.com/blog/2009/12/in-search-of-the-potato/
[12] Я предпочитаю определение Кема Кейнера *Тестирование как социальная наука,* Toronto Association of Systems and Software Quality, October, 2006., http://www.kaner.com/pdfs/KanerSocialScienceTASSQ.pdf

Вдохновение для тестирования

Часто (и по праву) говорят, что нужно учитывать куда больше, нежели явные требования, чтобы быть в состоянии хорошо тестировать[13].

Самая важная причина тому то, что на момент тестирования мы знаем о продукте больше, чем до его имплементации, и тестировщики рассматривают различные уровни детализации. Другие причины в том, что требования всегда неполны (они не были написаны Господом всемогущим, и список неявных требований почти бесконечен), некоторые вещи нужно опробовать, чтобы в них разобраться, и нет смысла тратить слишком много сил на фиксацию требований.

Мы также должны спросить себя, какую проблему пытается решить тестирование. Я думаю, в основном это ситуации вроде такой:

Мы знаем, что с продуктом, который мы создаем, возникнут проблемы. Нам нужна ваша помощь, чтобы убедиться, что релизный продукт может использоваться пользователями и не будет падать, и что он удовлетворительно поможет им решить их задачи.

А не такой:

Мы должны убедиться, что реальный продукт соответствует заявленному в требованиях.

Идея о множестве источников информации часто сопровождается примерами "иной информации" – например, чем-то вроде нужд пользователя, необходимости читать код, знания технологии, с которой работает ПО, или понимания характеристик качества в определенном контексте.

В разделе идей для тестирования есть подробный список[14], который можно рассматривать как "чеклист для создания собственного чеклиста". В нем не все всегда будет релевантным, и я рекомендую потратить минутку на каждую категорию. Подумайте о том, что важно, и решите для себя, какие источники информации надо рассматривать подробнее. Полученная вами информация может сразу дать вам хорошие идеи для тестов, указать на важные вещи, и со временем помочь с детализацией вашего тест-дизайна. Комплексное слияние помогает понять, что важно, во множестве ситуаций.

Использование множества источников информации, включая сам продукт и нужды пользователей, поможет вашим тестам быть лучше связанными с реальностью – тестирование будет тщательным.

---

[13] Один из лучших примеров: "Тестировщик, который рассматривает проектную документацию (явную спецификацию продукта) как единственный источник требований, вредит процессу тестирования". Уроки 32 и 33 в книге Кема Кейнера, Джеймса Баха и Брета Петтикорда, *Lessons Learned in Software Testing*, John Wiley & Sons, Inc., New York 2002. Другой пример – "Тестировщики должны базировать свои тесты на информации, которую они могут получить – и они могут получить много информации из источников, не включающих спецификацию". Кейм Кейнер, *The Ongoing Revolution in Software Testing*, Software Test & Performance conference, 2004, http://www.kaner.com/pdfs/TheOngoingRevolution.pdf

[14] Рикард Эдгрен, Мартин Янссон и Хенрик Эмильссон, *37 Sources of Test Ideas*, http://thetesteye.com/blog/2012/02/announcing-37-sources-for-test-ideas/

Использование множества моделей

Требования – это важная модель того, чего должно добиться ПО, однако это куда более сложный вопрос, выходящий далеко за рамки документированных требований. Вы должны относиться к требованиям критически, и, используя другие источники, попытаться найти неявные [15]значимые требования.

Если спецификация любого типа (концептуальная, техническая, функциональная, дизайнерская) существует, то это ценная информация. Вы также можете поискать спецификацию для другой функциональности, и в этом случае тест-спецификации могут быть наиболее интересными.

Сам код – тоже модель, и вы можете извлечь пользу из его чтения, или позволить тому, кто его читал, рассказать вам о нем. Обращайте особенное внимание на старый, новый, нестабильный, нечитабельный, прошедший ревью код.

Справка моделирует функциональность – если она написана с точки зрения пользователя.

Самая лучшая модель – это само ПО. Если вы можете увидеть, что оно делает, в живой природе – вы поймете, как взаимодействуют различные части. Используя ПО, вы всегда составляете его ментальную модель, даже если делаете это бессознательно.

У вас есть неявные ментальные модели, которыми необязательно возможно делиться с другими. Это нормально – спросите ваших друзей, как они представляют себе год. Некоторые никак его не представляют, но меж тем прекрасно понимают, что это значит. Ваши тесты и их результаты моделируют ваше тест-пространство, и, как и в случае любой модели, вы должны смотреть и вовнутрь, и вовне.

Будьте открыты для моделей других людей – тестировщики не идеальны, и как и все, делают ошибки.

Продукт и проект

Всегда существует какая-то история, и с шансами вы работаете с продуктом в течение множества релизов, и можете использовать данные прошлых прогонов тестов.

- Составили ли вы каталог ошибок для распространенного типа багов?
- Можете ли вы проставить теги багам и тикетам поддержки, чтобы использовать их для вдохновения?
- Каким образом вам удалось найти важные баги так рано, и как мимо вас проскользнули пропатченные баги?
- Как пользователи пытались решить свою проблему без вашего ПО?
- Используете ли вы ваши маркетинговые материалы как руководство по самым важным областям?
- Ищете ли вы в сети информацию о реальном использовании и проблемах, вызванных им?

Вы можете и должны получить много информации о функциональности и соответствующих ей рисках. Вы часть проекта, часть процесса, и вы знаете обо всех различных конечных продуктах.

У вас есть контекст выполнения тестов, который должен направлять ваш тест-дизайн.

- Когда вы можете к этому приступить, и сколько у вас времени?
- Каков опыт и специализация тестировщиков?
- Что будет тестироваться другими людьми?
- Какие новые типы тестирования нужно применить в этот раз?

[15] Хороший старт для неявных требоавний – на слайдах 8-9 в курсе Кема Кейнера и Джеймса Баха, Black Box Software Testing, Specification-Based Testing http://www.testingeducation.org/BBST/specbased/BBSTspecBased.pdf

- Готовы ли вы ко всегда происходящим неожиданностям?
- Возможно, стоит составить свободный, открытый план, который легко изменить.

Если вам повезло, вы создали или можете создать цели качества, служащие как направляющие для важных областей. Могут существовать и информационные цели, хорошо направляющие вашу деятельность.

Люди и навыки

Ваша команда и заинтересованные лица – обладатели большого количества знаний, уточните детали, когда появится возможность. Заслужите уважение разработчиков, и вы получите неформальные инсайдерские подсказки для поиска уязвимостей. Что вы знаете о нуждах пользователя, его знаниях, чувствах, расстройствах? Что ценно для пользователей, команды и других заинтересованных лиц?

Чтобы выяснить, что должно быть протестировано, надо использовать и свои навыки, и навыки команды:

- Знания, опыт, интуиция, субъективные ощущения.
- Исследования: изучайте и учитесь.
- Аналитическое мышление: моделируйте детали и общую картину, следуйте путям.
- Критическое мышление: ищите проблемы, риски и возможности.
- Креативность[16]: расширяйте спектр тестирования, генерируйте новые идеи, используйте латеральное мышление.
- Глаз Тестировщика: хочет видеть ошибки, видит много типов, смотрит во множестве мест, смотрит часто, концентрируется на важном, смотрит чужими глазами[17].

Больше вдохновения

Полезно много знать об актуальном и потенциальном использовании продукта. Используйте любой шанс, чтобы посетить пользователей и пообщаться с ними, выясните, что они сообщают службе поддержки, узнайте "настоящие" требования – какие проблемы пытаются решить пользователи?

Можете ли вы по-настоящему использовать продукт, находясь внутри проекта? Опробовать все на себе, пить свое домашнее шампанское? Использование реального ПО для достижения значимой цели – мое любимое вдохновение для тестирования.

Вы можете посмотреть на конкурентов – другие программные продукты, внутренний инструментарий, но также и аналоговые системы: как выполнялись похожие функции, пока не были компьютеризированы?

Чтобы понять нужды бизнеса, логику, информацию, знания, стандарты и законы, можно попробовать тестировать в паре с бизнес-экспертом.

Знание технологии означает, что вы знаете, как можно взаимодействовать с ПО, и что в нем важнее всего. Это также важно для понимания, как запускать тесты, и как сделать этот запуск эффективным. Использование своей основной платформы для частной работы – короткая дорожка, знание релевантных инструментов – необходимость.

Если вы много знаете о существующей тест-теории, вы можете использовать множество различных техник и подходов, и ваши шансы на нахождение важной информации будут выше. Классические техники тест-дизайна

[16] Рикард Эдгрен, *Where Testing Creativity Grows*, EuroSTAR conference 2007, http://thetesteye.com/papers/where_testing_creativity_grows.pdf
[17] Рикард Эдгрен, *The Eye of a Skilled Software Tester*, The Testing Planet, March 2011 – Issue 4, http://wiki.softwaretestingclub.com/w/file/fetch/39449474/TheTestingPlanet-Issue4-March2011.pdf

могут нечасто быть подходящим вариантом (кроме неявного деления на классы эквивалентности, которое используется повсеместно), но в правильной ситуации они могут быть мощным подспорьем.

Вдохновляйтесь общими тест-идеями вроде быстрых тестов, туров, мнемоник, эвристик, характеристик качества, хитростей[18], атак[19], чеклистов, и другим знанием из книг, курсов[20], блогов[21], форумов, сайтов, статей[22], конференций и разговоров.

Не используйте идеи, не оценив, важны ли они, и релевантны ли контексту вашего продукта. со временем вы сможете создавать собственные списки триггеров тест-идей, которые легко трансформируются в полезные тесты для реальных проектов.

Двойная работа?

Можно возразить, что использование широкого списка источников вдохновения для тестирования дублирует работу аналитиков требований. Это отчасти верно, и может быть менее затратным в случае наличия приложений к требованиям, подробнее описывающих атрибуты и предпочтения, и собранную мотивационную информацию[23].

В то же время тестирование рассматривает такие детали, до которых требования никогда не снисходят. Требования не укажут время ответа для абсолютно всех действий во всех ситуациях, а ручные тестировщики будут постоянно неявно оценивать производительность, если им разрешено или сказано это сделать.

Тестировщикам (и разработчикам) нужно понимать настоящие требования (а не сам документ), чтобы сделать отличную работу. Это потребует некоторой дупликации, однако даст более богатое совместное понимание благодаря разнообразию источников.

К тому же, проделывая это упражнение, тестировщики получат ряд источников для использования в качестве оракулов [24]в ходе оценки интересного поведения под тестами.

[18] Много советов доступно на http://www.quicktestingtips.com
[19] Джеймс Уиттакер, *How to break Software: A Practical Guide to Testing*, Addison-Wesley, Boston 2002
[20] Отличный бесплатный курсовой материал доступен на http://www.testingeducation.org (в основном от Кема Кейнера)
[21] Блог-портал для старта: http://www.testingreferences.com
[22] Колонки Майкла Болтона в Better Software вдумчивы и вдохновляют: http://www.developsense.com/publications.html
[23] Отличная книга про требования: Дональд Гауз, Джеральд Вайнберг, *Exploring Requirements: Quality Before Design*, Dorset House Publishing Co. 1989
[24] Для более внятного списка эвристик оракулов см. Бах/Болтон, HICCUPPS (F) - http://www.satisfice.com/rst.pdf

37 источников тест-идей

Мы рекомендуем постоянно собирать идеи для тестов из разнообразных информационных источников. Посмотрите на нижеуказанные моменты, и подумайте о ценностях, рисках, возможностях; найдите возможность быстро покрыть значимые вещи.

Продукт	1. **Возможности.** Первые и очевидные идеи для тестов имеют дело с тем, что продукт предположительно должен делать. Хороший старт – это требования, примеры и другие спецификации, или список функций, созданный на основании имеющегося ПО. Однако старайтесь также выявить неявные требования, недокументированные ожидания пользователей. Будьте начеку, не пропустите нежелательные возможности. 2. **Режимы падений.** ПО может упасть множеством способов, поэтому задавайте вопросы "что, если" для генерирования идей тестов, выявляющих работу с внутренними/внешними, ожидаемыми/неожиданными, (не)намеренными, реалистичными/спровоцированными отказами. Бросьте вызов устойчивости системы, сломаться может любой объект или компонент. 3. **Модели.** Модель состояний помогает выявить идеи тестов состояний, переходов и путей. Карта анатомии системы показывает, что может быть протестирована, и может выявить взаимодействия. Создайте свою модель, используя структуры вроде SFDPOT из Эвристической Модели Тест-Стратегии. Визуальную модель проще показать команде, и моделирование, как правило, дает более глубокое понимание и новые идеи. Множество богатых моделей даст наилучшие идеи для тестов. 4. **Данные.** Идентифицировав намеренные и ненамеренные данные (какой-то шум есть всегда), вы получите хороший старт для ряда идей для тестов. Следуйте за простыми и сложными данными в приложении, побывайте внутри и вне границ, бросьте вызов типам и форматам данных, используйте CRUD (Создание, Чтение, Обновление, Удаление), исследуйте зависимости, и посмотрите на данные в разных местах. 5. **Окружения.** Продукт – это не остров, поэтому совместимость с окружениями (железом, ОС, приложениями, конфигурациями, языками) – одна из множества важных проблем тестирования. Не забудьте и про действия рядом с вашим продуктом. Понимая общую картину, вы получите надежные тест-идеи, которые нельзя получить, рассматривая изолированную функциональность. **6. Белый ящик.** Применив разрушительное мышление тестировщика к взглядам разработчиков на архитектуру, дизайн и код, вы можете бросить вызов допущениям и найти дешевые в исправлении ошибки. Обращайте особое внимание на решения и пути, которые могут быть непонятными с точки зрения черного ящика. Покрытие кода – штука небесполезная, она может помочь найти еще не протестированные вещи. **7. История продукта.** Старые проблемы с шансами возникнут в новых формах. Поищите по вашему баг-трекеру или в системе поддержки, или же создайте каталог ошибок, запоминайте критические падения и анализ их причин. Используйте старые версии вашего ПО, как источник вдохновения и оракул. **8. Слухи.** Обычно о качестве и проблемах много говорят, и некоторые разговоры вредят продукту и организации. Используйте слухи в качестве идей для тестирования. Ваша миссия – уничтожить слухи или же доказать, что они верны. **9. Реальный продукт.** Взаимодействуя с продуктом, вы получите множество идей о том, что подвержено ошибкам, взаимосвязано, вызывает интерес. Если вы можете опробовать продукт на себе, вы куда лучше поймете, что в продукте значимо. Если "Качество – это ценность для какого-то человека", то с шансами этот человек – "я". **10. Технологии.** Зная внутреннюю кухню технологии, с которой раотает ваше ПО, вы можете увидеть проблемные области и вещи, которые могут пойти не так; понять возможности и аспекты безопасности, то, какие параметры надо менять, и когда. Вы можете опробовать правильные вариации и поговорить о технологиях с разработчиками. **11. Конкуренты.** Рассматривая разные решения похожих проблем, вы можете напрямую получить идеи для тестов, а также чувство того, в каких характеристиках заинтересованы конечные пользователи. Могут существовать и домашние решения (например, таблицы Excel), и аналоговые решения тех же проблем – все это источник вдохновения. Можете ли вы получить интересные идеи для тестов из поддержки ваших конкурентов, "ЧаВо" или другого материала?
Бизнес	**12. Цель.** Общее предназначение продукта даст вам цели для ваших тест-идей. Задайте несколько дополнительных "почему", чтобы выявить истинные цели. Это даст вам широкую и благодатную площадку для старта, и вы сможете быстро найти очень важные проблемы. **13. Бизнес-задачи.** Каковы самые значимые задачи для компании (и на уровне подразделения)? Есть ли требования, противоречащие этим задачам? Знаете ли вы общую картину, видение продукта и драйверы ценностей? **14. Имидж продукта.** Запланированное поведение и характеристики продукта могут быть явными или неявными, находящимися внутри умов тех, кто разрабатывал или пользовался продуктом. Если вы знаете и можете продемонстрировать угрозы имиджу продукта – например, противоречие маркетинговым материалам – вы сможете писать убедительные баг-репорты. **15. Бизнес-знания.** Если вы знаете цель продукта и контекст, в котором он работает, вы поймете, принесет ли он ценность пользователям. Если вы не можете завладеть этими знаниями, скооперируйтесь с тем, кто знает потребности, логику и окружения. **16. Юридические аспекты.** Надо ли вам учитывать контракты, штрафы и другие законодательные обязательства? Что будет стоить компании дороже всего в юридическом плане? Есть ли у вас юрист, который может дать вам советы, чего стоит избежать?

Команда	**17. Креативные идеи.** Все продукты уникальны и требуют тестов, которых доселе не существовало. Пользуйтесь техниками латерального мышления (Шесть Шляп от Эдварда де Боно, провокативными операциями, методом от противного, рандомной стимуляцией, Google Goggles), чтобы создать креативные тесты. Метафоры и аналогии – хороший способ стартовать в новом направлении. **18. Внутренние коллекции.** Используйте или создайте списки вещей, которые часто бывают значимы в вашем контексте. Некоторые называют это паттернами качества/тестирования, а некоторые создают специфичные для продукта списки быстрых тестов. 19. **Вы сами.** Вы пользователь. Вы можете быть заинтересованным лицом. Вы значимы. Получите преимущество от ваших сильных сторон благодаря опыту, навыкам, знаниям и осведомленности о проблемах. Используйте ваше субъективное мнение и ощущения, чтобы понять, что тут значимо. И не забудьте признать свои слабости и слепые зоны.
Проект	**20. Бэкграунд проекта.** Причины для существования проекта – источник множества решений, и стоит знать об истории предыдущих (схожих) проектов, дабы провести эффективное тестирование. **21. Информационные цели.** Жизненно важно понимать явные и неявные цели тестирования. Если вы не можете их получить, создайте ваши собственные цели качества, которые направляют идеи для тестирования любой фичи. **22. Проектные риски.** С рядом сложных вещей в проекте тестирование может помочь разобраться. Вам надо знать, с какой функциональностью проблемы у разработчиков – вы скорректируете свое расписание в зависимости от рисков, которые нужно снизить в первую очередь. **23. Тест-артефакты.** Для дальнейшего тестирования можно пользоваться не только вашими собственными тест-идеями, логами и результатами – попробуйте использовать результаты тестов с других проектов, отчеты о бета-тестировании, оценку удобства использования, результаты тестирования третьими сторонами, и так далее. На какие вопросы вы хотите иметь возможность ответить в статусных отчетах? **24. Долг.** Короткие дорожки, которыми мы ходим, часто вызывают постоянно растущий долг. Это может быть проектный долг, управленческий долг, технический долг, долг ПО, долг по тестированию, называйте это как угодно. Если команда следит за списком своих долгов, вы можете сопоставить с этим списком свои идеи для тестов. **25. Разговоры.** Неформальная информация, которую вы получаете от других людей, может содержать более важные по сравнению со спецификациями вещи. Многие могут помочь вам с вашим тест-дизайном, кто-то лучше разбирается в значимости, а что-то могут упомянуть вскользь. Если разработчики знают, что вы можете найти интересные штуки, они дадут вам инсайдерскую информацию о сомнительных частях приложения. Набор вопросов к разработчику может быть невинным "что, с твоей точки зрения, надо протестировать" или "какую часть твоего кода тебе больше понравилось писать?" **26. Контекстный анализ.** Что еще в текущей ситуации должно повлиять на то, что вы тестируете, и как? Знаете ли вы о силах рынка и драйверах проекта? Есть ли что-то изменившееся, что должно привести к новым путям тестирования? Что тестируют другие? Какие силы и слабости у проекта и его команды? **27. Множество конечных продуктов.** Тестировать нужно много чего: сам продукт, инсталляционный пакет, программные интерфейсы, расширения, код и комментарии, свойства файлов, справку, другую документацию, релизные заметки, файлы Readme, маркетинговые и тренировочные материалы, демо-версии, и так далее. Все это также содержит информацию, которая может послужить источником вдохновения. **28. Инструменты.** Если что-то можно сделать очень быстро, то попробовать – хорошая идея. Инструменты – не просто средство достижения цели, они могут также использоваться как стартовая точка анализа.

Заинтересованные лица	**29. Характеристики качества.** Характеристики качества всегда важны для успеха проекта, однако эта область может быть как легко достижимой, так и трудной и критичной. Наше определение включает возможности, надежность, удобство использования, харизму, безопасность, производительность, IT-руемость, совместимость, поддерживаемость, тестируемость, ремонтоспособность, портируемость, и множество подкатегорий. Многие из этих характеристик можно использовать как непрерывный поток тест-идей в уме, свободно выполняя их, и будучи наготове распознать нарушения. **30. Продуктовые страхи.** То, о чем действительно волнуются заинтересованные лица, сильнее рисков и не нуждается в приоритезации – оно нуждается в тестировании. Сложно верифицируемые, но полезные для тестирования страхи: потеря имиджа, неверные решения, ущерб, людям не понравится ПО. У разных людей страхи различаются, выясните, что будет самым важным. **31. Сценарии использования.** Пользователи хотят достичь чего-то или ощутить что-то, используя ПО, поэтому создайте тесты, которые разнообразными путями имитируют последовательности в поведении продукта, а не его изолированные характеристики. Чем больше достойных доверия паттернов использования вы знаете, тем реалистичней будет ваша работа. Попробуйте также эксцентричные тесты мыльной оперы для расширения тестового покрытия. **32. Полевая информация.** Помимо знания о проблемах, с которыми столкнулись пользователи, их окружениях, нуждах и чувствах, уделите время пониманию ваших пользователей как в режиме ошибок, так и в режиме успеха. Поговорите с конечными пользователями, предпродажной подготовкой, продажниками, маркетологами, консультантами, специалистами поддержки, или, что еще лучше, немного поработайте там. 33. **Пользователи.** Подумайте о разных типах пользователей (люди, которых вы знаете, персоны), различных нуждах, различных ощущениях, различных ситуациях. Выясните, что им нравится, а что нет, что они используют совместно с вашим ПО. Разыграйте сцену в тест-лаборатории, в которой тестировщики будут изображать различных пользователей – к каким тест-идеям это приведет? Лучший вариант, конечно, нефильтрованная информация напрямую от пользователей из их контекста. Помните, что два разных пользователя могут по-разному думать об одной и той же области.
Внешние факторы	**34. Публичные коллекции.** Извлеките пользу из общих или специализированных списков багов, ошибок кода или тест-идей. Создавая свой чеклист, подходящий для вашей ситуации, попробуйте эти: • Appendix A of Testing Computer Software (Кейнер, Фолк, Нгуен) • Boris Beizer Taxonomy (Отто Винтер) • Shopping Cart Taxonomy (Гири Вийярагаван) • Testing Heuristics Cheat Sheet (Элизабет Хендриксон) • Это еще не конец (Майкл Хантер) Почерпните хитрости и техники тестирования из книг, блогов, конференций, ищите эвристики тест-дизайна или создайте подходящие лично для вас. **35. Стандарты.** Найдите релевантные бизнес-стандарты, законы и требования. Прочтите и вникните в стандарты пользовательского интерфейса, безопасности, политик. Есть статьи, описывающие, как сломать даже что-то соответствующее стандартам – можно ли включить эти способы в идеи для тестов? **36. Справка.** Разнообразные справочные материалы – хороший источник оракулов и тест-вдохновения. К примеру, это атлас для географического продукта. Общие разноплановые знания могут быть вам полезны, а Википедии может быть достаточно, чтобы быстро разобраться в статистическом методе. 37. **Поиск.** Использование Google и других способов – хороший способ найти то, что вы ищете, и то, о нужде в чем в и не подозревали (приятная неожиданность).

Рикард Эдгрен, Мартин Янссон и Хенрик Эмильссон - thetesteye.com v1.0

Вдохновлено работой Сабурина " 10 Sources for Testing Ideas",эвристикой HICCUPPS(F) Баха/Болтона и трудами Кейнера

Анализ

Аналитическая [25]часть тест-дизайна отвечает на вопрос, *что нам надо протестировать.* Это включает идентификацию и изучение различных информационных источников, а также искусство выяснения, что важно или может быть важным.

Я хочу остановиться на естественном инстинкте тестировщика, подталкивающем его к разбиению информации о продукте на элементы, которые можно применить в тестировании. Это могут быть детали требований, инсайты от разговора с заказчиком, слоганы с веб-сайта – это длинный список, как мы убедились в предыдущей главе.

Кейнер называет это **обучением**, Майкл Болтон (и Джеймс Бах) в дополнение используют термин "факторинг" – "Факторинг – это процесс анализа объекта, события или модели с целью определить элементы, из которых он состоит[26]".

У Эдварда Де Боно есть общая техника латерального мышления – **фракционирование**, и она явно включает креативный аспект.

"Мы не стараемся найти реальные компоненты ситуации – мы стараемся создать эти компоненты[27]".

В тестировании эта концепция включает не только математическую факторизацию[28], анализирующую и ищущую точные компоненты целого.

В тестировании мы скорее ищем элементы, которые полезны, содержат что-то, что мы хотим протестировать с целью поиска важной информации о продукте. Мы можем создавать необоснованные искусственные разделения, потому что они нам помогают. Если сценарий использования включает Firefox 3.6, мы можем добавить все прочие браузеры и платформы, которые считаем релевантными и нужными для тестирования; мы автоматически думаем о настройках браузеров.

Мы добавим то, что мы знаем или думаем, что знаем, используя наше понимание значимых вещей, чтобы эти элементы синтезировались в полезные тест-идеи. Мы также будем искать множество решений, и важно тут не то, что факторы совокупно складываются в информационный источник – важнее тут то, что вы хотите тестировать полезные элементы, и если они друг другу противоречат – это совершенно нормально.

Концентрируйтесь не только на том, что важно, но и на том, что может быть важным, или способно генерировать другую важную информацию. Вы должны создавать тесты, **достойные тестирования**, и типичным примером могут быть ситуации, когда вы не знаете, важно это или нет, но хотите убедиться путем тестирования.

Эвристики анализа

Анализ широкого и релевантного тестового пространства – это трудный для освоения и оттачивания навык; он связан с пониманием целого, деталей, и значимого. Однако существует множество эвристик [29]и устоявшихся

[25] Анализ можно также назвать факторингом, потому что анализ в тестировании = Факторинг + Латеральное мышление + Понимание, что важно + Множество решений.
[26] Майкл Болтон, презентация *Confirmation Bias*, http://www.developsense.com/presentations/2010-04-ConfirmationBias.pdf
[27] Страница 135 в книге Эдварда Де Боно *Lateral Thinking - Creativity Step by Step*, Harper & Row, New York 1990 Perennial edition
[28] Wikipedia: Factorization, http://en.wikipedia.org/wiki/Factorization

правил, которыми можно пользоваться для усиления взаимосвязанных процессов анализа и дизайна. Вот несколько примеров:

1. Как можно быстрее (ASAP) – маленький кусочек информации о системе может сильно повлиять на тестирование (а обратная связь может сильно повлиять на разработку).
2. Копайте глубже – Требования или риски – неплохой старт, но это всего лишь начало пути.
3. Хм? Серьезно? – вы можете что-то не понимать, ваше понимание может быть неверным, истина может быть не важна, или может значить больше, чем вы думаете[30].
4. Эвристика "У Мэри был барашек" – возьмите каждое слово, вникните в него, оспорьте его, используйте синонимы, антонимы и комбинации[31].
5. Оспаривайте все, особенно это заявление – в случае с важными вещами бросьте вызов утверждениям.
6. Спрашивайте людей о деталях, пояснениях и ожиданиях[32].
7. Анализ противоречий [33]– ищите противоречия, дыры и то, что можно неправильно понять.
8. Эвристика "если только не…" – возьмите любое утверждение о вашем продукте, процессе или модели, и добавьте к нему "если только не…". Посмотрите, куда это вас приведет[34].
9. Применение бизнес-знаний – изучите бизнес, а если не можете, поработайте с тем, у кого эти знания есть.
10. Суждение о важности – думаете ли вы, что это важно? Думает ли так кто-то еще? Может ли это стать важным? Будет ли пользователь [35]огорчен или выведен из себя проблемой?
11. Что, если? – мозговой штурм о том, что может произойти, или что можно будет испытать.
12. Латеральное мышление – искусственное фракционирование, аналогии, генерация альтернатив, противоположностей, рандомная стимуляция – все полезное подойдет[36].
13. Триггеры тест-идей – ознакомьтесь с информацией или ситуациями, которые, по вашему мнению, могут генерировать хорошие идеи.
14. Приближение/отдаление – перейдите выше или ниже на уровень детализации, исследуйте содержимое.

---

[29] Широкое определение эвристик включает что угодно, помогающее вам решить проблему – к примеру, каждый элемент в чеклистах в этой книге – это эвристика. Тут я фокусируюсь на способах размышления об аналитической части вашего тест-дизайна.

[30] Эвристика Баха/Болтона для критического мышления в курсе *Rapid Software Testing*, http://www.satisfice.com/rst.pdf

[31] Глава 10 в статье Mind your meaning книги Donald C. Gause/Gerald M. Weinberg, *Are Your Lights On? How to Figure Out What the Problem REALLY Is*, Dorset House Publishing Co. 1990 (впервые опубликовано в 1982)

[32] "Вопросы вне контекста" перечислены в книге *Exploring Requirements* (Gause/ Weinberg), и у Майкла Болтона есть статья *Context-Free Questions for Testing* на http://www.developsense.com/blog/2010/11/context-free-questions-for-testing/

[33] Слайды 73-78 в презентации Кема Кейнера *Developing Skills as an Exploratory Tester*, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL, November 2006, http://www.kaner.com/pdfs/ExploratorySkillsQAI2007.pdf

[34] Эвристика "Если только не" описана Майклом Болтоном, http://www.developsense.com/blog/2007/03/white-glove-heuristic-and-unless/

[35] Понимайте "пользователя" в широком смысле, если хотите найти проблемы в областях вроде поддерживаемости, тестируемости и ремонтопригодности.

[36] Эдвард Де Боно - *Lateral Thinking - Creativity Step by Step*, 1990 Perennial edition

15. Новые связи – скомбинируйте разнообразные знания важным образом, сопоставьте детали и общую картину, что нуждается в дополнительном изучении?
16. Конкретизация обобщений – сделайте идеи, пришедшие из общих тестов, таксономий, чеклистов (Хендриксон[37], Хантер[38], Кейнер[39], Сабурин[40], Software Quality Characteristics) контекстно-зависимыми.
17. Создайте рисунок, диаграмму или таблицу – проще поделиться с другими, может спровоцировать новые идеи.
18. Загадочное молчание – "Если что-то интересное или важное не описано и не задокументировано, оно может быть не продумано до конца, или дизайнер может скрывать проблемы[41]".
19. Разнообразие – подумайте об этом разным образом[42].
20. Эвристика "Длинный поводок" – "Дайте себе отвлечься… потому что вы никогда не знаете, что вы найдете… но периодически сверяйтесь со своим статусом по отношению к миссии[43]".
21. Нырнуть и уйти – начните с самой сложной части, посмотрите, что произойдет, бросьте, когда захотите[44].
22. Эвристика Румсфельда [45]– исследуйте известные неизвестные, подумайте над неизвестными неизвестными.
23. Не останавливаться – когда вы нашли то, что искали, подумайте еще – возможно, за углом ждут находки получше.
24. Всегда делайте паузы – анализ (и дизайн) – это длительная деятельность, делайте паузы всегда и никогда.
25. Сделано другими – выясните, какие типы тестирования проводятся другими людьми, их можно пропустить или затронуть лишь слегка[46].
26. Контекстный анализ – выясните, какие факторы в нынешней ситуации могут направить ваше тестирование[47].

---

[37] Элизабет Хендриксон, *Test Heuristics Cheat Sheet*, http://testobsessed.com/wp-content/uploads/2011/04/testheuristicscheatsheetv1.pdf
[38] Майкл Хантер, Это еще не конец
[39] Приложение A, Common Types of Errors Cem Kaner, Jack Falk, и Hung Q. Nguyen, *Testing Computer Software, Second Edition*, John Wiley & Sons, Inc., New York 1999. Доступно по адресу http://www.testingeducation.org/BBST/testdesign/Kaner_Common_Software_Errors.pdf
[40] Роберт Сабурин, *10 Sources of Testing Ideas*, http://www.amibugshare.com/articles/Article_10_Sources_of_Testing_Ideas.pdf
[41] Воркшоп Кема Кейнера и Джеймса Баха, *Risk-Based Testing: Some Basic Concepts*, QAI Managers Workshop, QUEST Conference, Chicago 2008, http://www.kaner.com/pdfs/QAIRiskBasics.pdf
[42] Много хороших примеров в статье Джеймса Баха и Майкла Болтона, *Exploratory Testing Dynamics* v2.2, http://www.satisfice.com/blog/wp-content/uploads/2009/10/et-dynamics22.pdf
[43] Эвристика Баха/Болтона для критического мышления в курсе *Rapid Software Testing*, http://www.satisfice.com/rst.pdf
[44] Страницы 100-103 в книге Джеймса Баха *Secrets of a Buccaneer-Scholar*, Scribner, New York 2009
[45] Оригинальная цитата Румсфельда: "Есть известные известные. Есть вещи, про которые мы знаем, что мы их знаем. Мы также знаем, что есть известные неизвестные, то есть мы знаем, что есть вещи, которые мы не знаем. Но есть и неизвестные неизвестные – мы не знаем, что мы о них не знаем".
[46] Хороший пример: "Одна техника – проверка единичных переменных и их комбинаций на граничных значениях – часто хорошо применяется в юнит-тестах и низкоуровневых интеграционных тестах. Они гораздо эффективнее системных тестов. Если разработчик тестирует именно таким образом, системные тестировщики должны концентрироваться на других рисках и других техниках" – Кем Кейнер, статья в блоге *Updating some core concepts in software testing*, http://www.satisfice.com/kaner/?p=11

27. Фрейминг [48]тестирования – привяжите свое тестирование к миссии тестирования, и узнайте об этой связи больше.
28. Мои домыслы – объясните свое тестирование и оспорьте его[49].

Эта деятельность окружена пониманием того, что значимо. Это очень важно для получения хороших тест-элементов, а выполняя эту деятельность, вы достигнете еще большего понимания. Это позитивная спираль и с толком потраченное время.

Невидимый анализ

Большую часть времени анализ будет стремительно происходить у вас в голове – вы заметите что-то важное и немедленно среагируете идеей для теста (зафиксированного письменно или выполненного). Это вполне нормально, документированный анализ нужен только там, где требуется ревью этого процесса.

Если вы потренируетесь, вы сможете делать это быстрее, а также будете лучше делиться своими мыслями.

Креативность очень важна – чем больше вы знаете о продукте и думаете о нем, тем больше элементов и идей у вас появится. Составьте их список, включая всякие безумства – их всегда можно удалить. Как сказал Роберт Сабурин, *"Соберите все идеи для тестов, которые можете найти! По моему опыту, лучше сознательно пропустить тест, чем пропустить его, потому что ваше время истекло, или вы просто про него забыли[50]!"*

Заметки

Для себя – но в первую очередь для окружающих – можно писать короткие заметки в ходе выявления важных вещей благодаря разнообразным источникам информации. Это позволит инспектировать процесс, возможно, даже воссоздать его, а еще это мощный инструмент передачи знаний.

Вы можете использовать кодирование – к примеру, описать тэги – во всех ваших релевантных документах, чтобы иметь возможность вернуться к ним и перекомбинировать информацию[51].

Типичные области, для которых подходят (и многим полезны) периодически обновляемые заметки:

- Наши источники тест-идей
- Пользовательские конфигурации
- (Пользовательские) обходные пути
- Корневые причины проблем из службы поддержки
- Найденные в ПО XXX баги
- Наши наиболее важные характеристики качества

---

[47] Хорошее руководство – раздел Project Environment section в статье Джеймса Баха, *Heuristic Test Strategy Model*, http://www.satisfice.com/tools/satisfice-tsm-4p.pdf

[48] Майкл Болтон, *Test Framing*, http://www.developsense.com/resources/TestFraming.pdf

[49] Алан Ричардсон, статья в блоге *Challenge your assumptions and presuppositions to identify useful variation*, http://www.eviltester.com/index.php/2008/04/18/challenge-your-assumptions-and-presuppositions-to-identify-useful-variation/

[50] Роберт Сабурин, *Just-in-time testing*, EuroSTAR 2010, http://www.amibugshare.com/courses/Course_Just_In_Time_Testing.zip

[51] Если вы хотите углубиться в вопрос, попробуйте пробную версию инструмента качественного анализа Atlas.ti. Вы можете использовать любой текстовый инструмент с функциями поиска, вики или блога.

- Тесты, которые нужно прогнать в первую очередь
- Типичные баги в юнит-тестах.

Возможно, вы сможете уговорить аналитика требований делиться заметками, чтобы получить детали и стремления, которые нельзя выразить в формате требований. Сделайте пользователей более живыми; когда требования формулируются словами, теряется много информации.

Характеристики качества

Характеристики качества [52]описывают атрибуты, которые дают преимущество большей части программных продуктов[53]. Они могут использоваться как для продукта целиком, так и для его частей. *Целое состоит из частей. Качество части определяется целым.*

Это общие характеристики, служащие богатым источником триггеров для идей по тестированию любого приложения. Некоторые из них вам не подходят, некоторые легко удовлетворить, а некоторые очень важны и сложны. См. материал для печати на следующих страницах для подробного списка вдохновляющих концепций в областях Возможностей, Надежности, Удобства использования, Харизмы, Безопасности, Производительности, IT-руемости, Совместимости, Поддерживаемости, Тестируемости, Ремонтопригодности, Портируемости. Никаких чисел к этим описаниям не дается, метрики опасны[54], так как скрывают то, что на самом деле важно.

Пройдитесь по этому списку с коллегами или заинтересованными лицами – они помогут вам сфокусироваться, и в то же самое время получат представление о сложности продукта (и его тестирования). Обсудите с командой каждый случай конфликта характеристик.

Это разбиение на категории может также быть стартовой точкой более целенаправленного нефункционального [55]тестирования, с полезным добавлением специализированного тестирования и облегченных [56]методов.

---

[52] В книгу включена обновленная версия постера http://thetesteye.com/posters/TheTestEye_Software-quality-characteristics-1-0/ (авторы – Рикард Эдгрен. Хенрик Эмильссон, Мартин Янссон).

[53] Чересчур длинное определение качества ПО может быть сформулировано так:
"Качество ПО – это **больше**, нежели **воспринимаемая** сумма **релевантных** характеристик качества (таких, как мощность, надежность, удобство использования, харизма, безопасность, производительность, IT-руемость, совместимость, возможность сопровождения, тестируемость, поддерживаемость, портируемость) для множества **разных** людей.

[54] Более мягкий тезис – "Невалидные метрики опасны", http://www.context-driven-testing.com/

[55] "Нефункциональное" – странное слово, однако определение Кейнера "парафункциональное" мало распространено, как и "тестирование по характеристикам".

[56] Посмотрите тег lightweight на http://thetesteye.com/blog - там представлены простые идеи нефункциональных тестов для всех.

Характеристики качества ПО

Пройдитесь по списку и подумайте о вашем продукте/функциях. Добавьте специфики в ваш контекст, и трансформируйте лист под свои нужды.

Возможности. *Может ли продукт выполнять ценные функции?*

- *Полнота:* все важные, желаемые конечными пользователями функции доступны.
- *Точность:* любой вывод или расчет в продукте верны и представлены значимыми цифрами.
- *Эффективность:* выполняет свои действия эффективно (не делая того, чего не должен делать).
- *Интероперабельность:* различные функциональности взаимодействуют наилучшим образом.
- *Многопоточность:* способность выполнять несколько параллельных задач и работать одновременно с другими процессами.
- *Агностицизм данных:* поддерживает все возможные форматы данных, и умеет обращаться с шумом.
- *Расширяемость:* третьи лица или заказчики могут добавлять функциональность или менять поведение.

Надежность. *Можете ли вы доверять продукту в различных и сложных ситуациях?*

- *Стабильность:* продукт не должен вызывать падения, необработанные исключения или ошибки сценариев.
- *Отказоустойчивость:* продукт достойно справляется со всеми предвиденными и непредвиденными ошибками.
- *Стрессоустойчивость:* как система справляется с превышением различных пределов?
- *Восстанавливаемость:* возможно ли восстановиться и продолжить использование продукта после фатальной ошибки.
- *Целостность данных:* все типы данных остаются неприкосновенными во всех областях продукта.
- *Безопасность:* продукт не будет вредить людям или их ценностям.
- *Восстановление после катастрофы:* что будет, если произойдет что-то очень, очень плохое?
- *Добросовестность:* является ли поведение продукта понятным, предсказуемым, достойным доверия?

Удобство использования. *Легко ли использовать продукт?*

- *Предоставление возможности*: продукт приглашает раскрыть свои возможности.
- *Интуитивность*: то, что продукт может сделать, легко понять и объяснить.
- *Минимализм*: в содержании и внешнем виде продукта нет ничего избыточного.
- *Изучаемость*: научиться пользоваться продуктом можно быстро и легко.
- *Запоминаемость*: если вы научились что-то делать, вы не забудете, как это делается.
- *Открытость*: информацию о продукте и его возможности можно изучить, исследуя пользовательский интерфейс.
- *Операбельность*: опытный пользователь может очень быстро выполнять распространенные действия.
- *Интерактивность*: состояния продукта и возможности взаимодействия с приложением (через графический интерфейс или API) просты для понимания.
- *Контролируемость*: пользователь чувствует, что работа продукта у него под контролем.
- *Ясность*: все указано явно, детально, на понятном языке, и не оставляет места сомнениям.
- *Ошибки*: сообщения об ошибках информативны, ошибку сложно совершить и легко исправить, если она совершена.
- *Соответствие*: поведение и дизайн одинаковы во всем продукте.
- *Настраиваемость*: настройки и поведение по умолчанию можно изменять ради большей гибкости.

- *Доступность*: продукт может использовать максимально возможное количество людей, и он соответствует применимым стандартам доступности.
- *Документация*: есть "Помощь", которая действительно может помочь и соответствует функциональности.

Харизма. *Есть ли она у продукта?*

- *Уникальность:* продукт выделяется, в нем есть что-то, чего больше нет ни у чего.
- *Удовлетворенность:* как вы себя ощущаете после использования продукта?
- *Профессионализм:* есть ли у продукта необходимый налет профессионализма и ощущение соответствия своей задаче?
- *Привлекательность:* все ли типы аспектов продукта привлекательны для глаз и прочих органов чувств?
- *Любопытство:* заинтересуются ли пользователи, будут ли пробовать сделать то, что можно сделать при помощи продукта?
- *Завороженность:* "подсядут" ли пользователи на продукт, будет ли им весело, войдут ли они в состояние потока, уделят ли ему максимум внимания в ходе использования?
- *Хайп:* должен ли продукт использовать новейшие и наилучшие технологии и идеи?
- *Ожидания:* продукт превосходит ожидания и отвечает нуждам, о существовании которых вы даже не знали.
- *Настроение:* правильное ли настроение у продукта и информации в нем, говорит ли он с вами на правильном языке и в нужной стилистике?
- *Прямота:* сильны ли (первые) впечатления?
- *История:* есть ли захватывающие истории о зарождении, создании или использовании продукта?

Безопасность. *Защищен ли продукт от нежелательного использования?*

- *Аутентификация:* идентификация продуктом пользователей.
- *Авторизация:* как продукт обращается с тем, что могут видеть и делать авторизованные пользователи.
- *Приватность:* способность не демонстрировать защищенные данные неавторизованным пользователям.
- *Дыры безопасности:* продукт не должен приглашать воспользоваться уязвимостями социальной инженерии.
- *Секретность:* продукт ни при каких условиях не должен раскрывать информацию о базовых системах.
- *Неуязвимость:* способность выдерживать попытки взлома.
- *Чистота от вирусов:* продукт не будет передавать вирусы или казаться вирусом.
- *Устойчивость к пиратству:* невозможность нелегально копировать и распространять продукт или код.
- *Соответствие:* стандарты безопасности, которых придерживается продукт.

Производительность. Достаточно ли быстро продукт работает?

- *Мощность:* множество продуктовых ограничений в разных обстоятельствах (к примеру, медленная связь).
- *Использование ресурсов:* грамотное использование памяти, хранилища и других ресурсов.
- *Время отклика:* скорость, с которой (по ощущениям) выполняются действия.
- *Доступность:* система доступна для использования, когда должна быть доступной.
- *Скорость обработки:* способность продукта обрабатывать множество операций.
- *Выносливость:* может ли продукт долго выдерживать нагрузку?
- *Обратная связь:* хороша ли обратная связь от системы или пользователя?

- *Масштабируемость:* насколько хорошо продукт расширяется, уменьшается и масштабируется горизонтально?

IT-руемость. *Легко ли установить, поддерживать и вести продукт?*

- *Системные требования:* способность запускаться на поддерживаемых конфигурациях и справляться с различными окружениями или отсутствующими компонентами.
- *Инсталлируемость:* продукт можно установить на желаемых платформах с необходимыми объемами доступного пространства.
- *Обновления:* простота обновления на новую версию без потери конфигурации и настроек.
- *Деинсталляция:* все ли файлы (кроме пользовательских или системных) и прочие ресурсы удаляются при деинсталляции?
- *Конфигурация:* можно ли конфигурировать инсталляцию различными способами или путями для удобства пользователя?
- *Возможность развертывания:* продукт может выкатываться отделом IT для различных (ограниченных) типов пользователей и окружений.
- *Поддерживаемость:* легко ли обеспечить пользовательскую поддержку продукта и его артефактов?
- *Тестируемость:* насколько эффективно продукт может быть протестирован пользователями в релизе?

Совместимость. *Насколько хорошо продукт взаимодействует с ПО и окружениями?*

- *Совместимость с оборудованием:* продукт можно использовать совместно с применимыми конфигурациями компонентов оборудования.
- *Совместимость с операционной системой:* продукт может запускаться на желаемых версиях операционной системы, и его поведение типично для нее.
- *Совместимость с приложениями:* продукт и его данные работают с другими приложениями, которыми, скорее всего, будут пользоваться потребители.
- *Конфигурационная совместимость:* способность продукта подстраиваться под конфигурации окружения.
- *Обратная совместимость:* может ли продукт делать все то, что могла делать предыдущая версия?
- *Прямая совместимость:* сможет ли продукт использовать артефакты или интерфейсы будущих версий?
- *Устойчивость:* влияние на окружение, к примеру, энергоэффективность, отключения, режим энергосбережения, дистанционный доступ.
- *Удовлетворение стандартам:* продукт удовлетворяет применимым стандартам, нормативам, законам или этике.

Внутренние характеристики качества ПО

Конечные пользователи не сталкиваются с этими характеристиками напрямую, однако эти характеристики могут быть важными для успешных продуктов.

Возможность сопровождения. *Можно ли обеспечить сопровождение использования продукта и его проблем?*

- *Идентификаторы:* легко ли определить части продукта и их версии, или специфичные ошибки?
- *Диагностика:* можно ли получить детали происходящих с пользователями ситуаций?
- *Разрешение проблем:* легко ли выявить проблему (к примеру, через лог) и получить помощь?
- *Дебаг:* можете ли вы наблюдать внутренние состояния ПО при необходимости?

- *Гибкость:* возможность использовать продукт большим количеством способов, нежели это было задумано изначально.

Тестируемость. *Легко ли проверять и тестировать продукт?*

- *Отслеживаемость:* продукт логирует действия на подходящих уровнях и в удобном формате.
- *Контролируемость:* возможность независимо устанавливать состояния, объекты или переменные.
- *Наблюдаемость:* возможность наблюдать за тем, что нужно протестировать.
- *Отслеживаемость:* может ли продукт сообщать, что он делает и как у него дела?
- *Изолируемость:* возможность тестировать часть продукта отдельно.
- *Стабильность:* изменения в ПО контролируются, и не слишком часты.
- *Автоматизация:* есть ли в продукте публичные или скрытые программные интерфейсы, которыми можно пользоваться?
- *Информация:* тестировщики могут изучить то, что должно быть изучено.
- *Аудитопригодность:* можно ли валидировать продукт и его результаты?

Поддерживаемость. *Можно ли незатратно поддерживать и расширять продукт?*

- *Гибкость:* возможность изменять продукт согласно требованиям заказчиков.
- *Расширяемость:* легко ли будет добавлять функциональность в будущем?
- *Простота:* код не сложнее необходимого и не мешает тест-дизайну, прогону тестов и оценке.
- *Читабельность:* код адекватно документирован, его легко читать и понимать.
- *Прозрачность:* легко ли разобраться в базовых структурах?
- *Модулярность:* код разделен на управляемые блоки.
- *Способность к рефакторингу:* довольны ли вы юнит-тестами?
- *Анализируемость:* возможность искать причины дефектов или другой важный код.

Портируемость. *Разрешено ли перемещать продукт в различные окружения и языки?*

- *Переиспользуемость:* можно ли повторно использовать части продукта в других проектах?
- *Адаптивность:* легко ли заставить продукт поддерживать другое окружение?
- *Совместимость:* удовлетворяет ли продукт распространенным интерфейсам или официальным стандартам?
- *Интернационализация:* продукт легко переводить.
- *Локализация:* все ли части продукта подогнаны под нужды целевой культуры/страны?
- *Устойчивость пользовательского интерфейса:* будет ли продукт хорошо выглядеть после перевода?

Рикард Эдгрен, Хенрик Эмильссон, Мартин Янссон – thetesteye.com v1.1

Синтезирование тест-идей

Процесс синтезирования тест-идей [57]очень трудно описать. Он включает в себя множество источников информации, понимание того, что важно, немного креативности для создания хитроумных идей тестов и эффективных способов их выполнения.

Самый простой способ – это взять требования, перефразировать каждый пункт, и при желании добавить деталей, используя разбиение на классы эквивалентности. Наилучший способ – это пользоваться множеством источников информации и генерировать достойные прогона идеи тестов, которые с хорошим шансом будут эффективными. Невозможный способ – скомбинировать всю важную информацию всеми доступными путями.

Лучше использовать каждый важный элемент, и каждую, с вашей точки зрения, значимую комбинацию. Вам помогут ревью, реальный подсказывающий тесты продукт, а также скорость тестов – это даст вам наилучшие идеи для вашей конкретной ситуации. Совместная командная работа над ментальной картой не займет много времени.

Я рекомендую записать идеи тестов – как минимум высокоуровневые, - чтобы их можно было легко использовать при обсуждениях и планировании. Если ревью еще не проведено, или ПО уже доступно – гораздо быстрее будет фиксировать полезные тесты после их выполнения, одновременно с результатом.

Не пытайтесь покрыть все, это невозможно. Стремитесь к широте своих тестов, и рассчитывайте на неожиданные открытия, которые помогут вам найти необходимую для успешного продукта информацию. Некоторые идеи тестов будут генерироваться на лету, во время выполнения тестов, когда вы своими глазами увидите, что на самом деле можно сделать с ПО.

Не останавливайтесь, наткнувшись на полезную идею. Подумайте еще, и, возможно, вы найдете способы получше, или же обнаружите новые решения старых проблем.

Тут применяются классические техники: классы эквивалентности, анализ граничных значений, модели состояний, деревья классификаций, поток данных[58], а также прямые заявления о том, что вы хотите протестировать[59], и что угодно, что вы считаете полезным.

В целях тренировки могут пригодиться классификации – на основе рисков, на основе спецификации, доменное тестирование. Однако в реальности куда полезнее быстро переключаться между ними и создавать собственные методики для имеющегося разнообразного исходного материала.

Тем не менее, вашему вниманию предлагается альтернативная классификация идей для тестов:

---

[57] Термин "тест-идея" предложен Брайаном Мариком. Определение Мартина Янссона (http://thetesteye.com/blog/2010/10/discussion-around-the-content-of-a-test-proposal/) практически идеально: "суть теста, выраженная минимальным количеством слов и отражающая самые важные его аспекты" (правда, это исключает визуальные идеи).
Однострочные тест-идеи: см. Рикард Эдгрен, More and Better Test Ideas, EuroSTAR 2009 conference, http://www.thetesteye.com/papers/redgren_moreandbettertestideas.pdf

[58] Две хороших книги - Lee Copeland, *A Practitioner's Guide to Software Test Design*, Artech House Publishers, Boston, 2003 и Torbjörn Ryber, *Essential Software Test Design*, Fearless Consulting Kb, 2007

[59] Для техники тест-дизайна, не использующей никакой поясняющей техники, нет устоявшегося названия, см. обсуждение на http://thetesteye.com/blog/2010/12/test-design-technique-name-competition/ (хотя мне кажется, что "Человеческая техника тест-дизайна" – подходящее название.

Непрерывные идеи тестов

Непрерывные идеи тестов нельзя взять и завершить – они продолжают жить, пока не получена вся возможная релевантная информация. Хороший пример [60]– это такая характеристика качества, как стабильность: чем больше вы тестируете, тем больше вы узнаете о стабильности продукта. Для важных характеристик вы, возможно, пользуетесь не только непрерывными, идущими в фоновом режиме тестами, но в качестве дополнения они очень полезны и ресурсоемки.

Другой пример – это свободное непрерывное тестирование удобства использования: в ходе тестирования тестировщик держит в уме релевантные качества продукта, и в случае каких-либо нарушений они будут замечены и зафиксированы.

Классические идеи тестов

Классические идеи тестов имеют дело с конкретикой и могут быть записаны в формате "убедиться, что…". Они могут отражать требования или другую известную тестировщикам информацию.

Не поймите меня неправильно, это важные идеи. Классические техники тест-дизайна с явными ожидаемыми результатами могут быть критически важными для покрытия всего необходимого в центральных частях функциональности, и вы должны знать, как ими пользоваться. Для функционального тестирования можно рассматривать отдельные изолированные возможности, или изучать всю систему целиком[61].

Чтобы не повредить ревью и отчетности, опасайтесь гранулярности: лучше использовать один тест вида "убедиться в правильном обращении с невалидным вводом (пустой ввод, пробелы, строка, с ударениями, Unicode, особые символы ASCII, длинная строка, неверные разделители)", чем дюжину тестов. В некоторых ситуациях вы можете даже использовать фразу "Убедиться, что все явные требования выполнены", и сфокусироваться на более интересных тест-идеях.

Воспользуйтесь также возможностью разобраться, какие тесты лучше подходят для автоматизации, и какие безопаснее всего тестировать автоматически, при помощи инструментов, и вручную.

Комбинаторные идеи тестов

Многие проблемы не возникают в изоляции (поэтому юнит-тесты очень хороши, но далеки от идеального решения), и поэтому тестирование стремится использовать функции, настройки и данные совместно, разным образом и в разных последовательностях.

Сценарное тестирование [62]– один из способов это сделать, а попарного тестирования может вполне хватить[63], если нет смысла выяснять самые распространенные, подверженные ошибкам, важные или наилучшие комбинации (тестировщик, знающий продукт, может сходу рассказать вам, где нельзя пренебречь тестированием взаимодействия).

Комбинаторные идеи тестов совместно используют множество тест-элементов, потому что мы полагаем (или, наоборот, не знаем), что эти элементы могут влиять друг на друга.

---

[60] Не уверен, но думаю, что непрерывная деятельность может быть связана **только** с характеристиками качества.

[61] Вдохновляющий список 41 системного аспекта можно найти в работе Joris Meerts, *Functional Testing Heuristics - A Systems Perspective*, http://www.testingreferences.com/docs/Functional_Testing_Heuristics.pdf

[62] Отличный обзор от Кема Кейнера - *An Introduction to Scenario Testing*, 2003. http://www.kaner.com/pdfs/ScenarioIntroVer4.pdf

[63] Хороший тщательный анализ попарного тестирования: James Bach, Patrick Shroeder, *Pairwise Testing: A best Practice that Isn't,* http://www.testingeducation.org/wtst5/PairwisePNSQC2004.pdf

Необычные идеи тестов

Требования часто пишутся так, чтобы их можно было "протестировать" (обычно это означает "верифицировать"). Это может запросто превратиться в квантифицированные требования в форме контракта, которые не описывают реальные нужды и желания.

Риск упустить важное можно снизить при помощи тестирования, которое необязательно направлено на опровержение гипотезы. *"Вы один из тех немногих, кто детально изучит продукт целиком до того, как он выйдет в релиз*[64]*".*

Необычные тест-идеи открыты, основаны на чем-то интересном, к ним нельзя применить дихотомию "Прошел/Упал", и они по большей части направлены на исследование и изучение с целью найти полезную информацию.

Подойдя к делу творчески, вы можете придумать оригинальные способы решения тест-проблем. Удобство использования можно протестировать, спросив любого постоянного пользователя о его ощущениях. Наилучшей оценкой харизмы может быть просмотр десятка альтернативных дизайн-макетов. Если вам доступно сто вариантов действия, вы можете попытать счастья, взглянув только на пять случайно выбранных. Неизвестную область можно просмотреть по верхам, надеясь на случайные счастливые открытия.

Эти тесты особенно важны для получения обратной связи, так как одна необычная идея может привести к двум другим, получше. У вас также есть шанс отбросить их как нерелевантные.

Визуальные идеи тестов

Не все можно выразить словами, а многие вещи гораздо лучше объясняются при помощи изображений. Попробуйте визуально изобразить ваши тесты – это также может помочь при генерации новых идей и осмыслении продукта.

Примеры таких тестов: модели состояний, изображения архитектуры, представление технологии, взаимоотношения между тест-элементами, различные вдохновляющие изображения.

Как и все прочие, визуальные идеи тестов трансформируются в любой другой тип и могут изменяться при надобности.

Что и как?

Большинство тест-идей, скорее всего, будет концентрироваться на том, что тестировать. Некоторые будут также указывать, как тестировать. Это очень выгодно – к примеру, если вы пишете, что собираетесь использовать Xenu's Link Checker, а кто-то сообщит вам, что есть куда более подходящий инструмент. Иногда переход от идеи теста к реальному тесту долог и труден, и это провоцирует доверие к исполнителю тестов (которым можете быть вы сами) и стремление не упускать возможности.

Держите в голове "что" и "как", но расценивайте их как единое целое – это позволит создавать более эффективные и инновационные тесты, использующие синергию разных областей. Используйте целостный подход: копаясь в деталях, вы должны держать в уме общую картину, систему в целом.

Эти категории сформулированы искусственно и используются только для того, чтобы лучше объяснить происходящие процессы. Когда вы уже достаточно знаете, забудьте про категории и перестройте свои

[64] Страница viii в книге: Cem Kaner, Jack Falk, Hung Quoc Nguyen, *Testing Computer Software, Second Edition*, John Wiley & Sons, Inc., New York 1999

модели тест-дизайна так, как удобнее вам, вашим коллегам, и образу мышления в вашей компании. Убедитесь, что вы меняете и пополняете свои тест-идеи по мере того, как узнаете о продукте больше. Проведите ревью своих идей, и у вас родятся вопросы, гипотезы и ответы.

Эвристики тест-дизайна

При дизайне тестов все средства хороши. Пользуйтесь любыми источниками информациями, отклоняйтесь от проторенных дорог, и полагайтесь на инстинкт, подсказывающий, что важно протестировать.

Используйте также эти основные правила, если они применимы.

1. Эвристики тест-анализа могут использоваться, как эвристики дизайна.
2. "Как можно позднее" – детализируйте ваши тесты как можно позднее. Ситуация будет меняться, и у вас будет больше информации[65].
3. "Различные полумеры" – "лучше протестировать побольше на хорошем уровне, чем идеально выполнить один-два типа тестирования[66]".
4. Чем быстрее, тем лучше – вы получите больше информации о продукте.
5. "Автоматизируйте 100% тестов, которые должны быть автоматизированы[67]" – вы лучше разберетесь в этом вопросе, выполнив тест как минимум один раз.
6. Эвристика "Без завитушек[68]" – 1) простые тесты могут очень вам помочь, 2) вам не требуется использовать модные техники тестирования, особенно там, где это не нужно.
7. Усложняйте – 1) сложные тесты могут быть очень эффективными, 2) нельзя пренебречь предосторожностями, создавая сложные тесты[69].
8. Бесплатное покрытие – при помощи одного теста можно покрыть множество различных вещей. Используйте непрерывные идеи тестов для характеристик качества. Держите их в уме и замечайте, если этим характеристикам что-то угрожает.
9. Подходящая гранулярность – подумайте об уровне детализации ваших тестов, чтобы облегчить их пересмотр.
10. Самое репрезентативное [70]– берите за образец самое распространенное, подверженное ошибкам, всеобъемлющее, важное.
11. Принцип важности – сфокусируйте большинство тестов на основной задаче, однако не забывайте и про остальные, а также про возможные ошибки.
12. Счастливый случай – самые лучшие тесты помогают найти то, о важности чего мы и не подозревали.
13. Можно выбросить – не бойтесь отказываться от тестов, в релевантность которых вы больше не верите.
14. Я узнаю это, когда увижу – поработайте с ПО, и вы почувствуете, что в нем важно, что подвержено ошибкам, и как это тестировать.
15. Интуиция – "Доверяйте вашей интуиции, думая о сложно предсказуемых вещах в условиях недостатка информации[71]".

---

[65] Robert Sabourin, EuroSTAR 2010, *Just-in-time testing*, http://www.amibugshare.com/courses/Course_Just_In_Time_Testing.zip

[66] Урок 283 в книге: Cem Kaner, James Bach, Bret Pettichord, *Lessons Learned in Software Testing*, John Wiley & Sons, Inc., New York 2002

[67] Их зачастую меньше, чем кажется, из-за стоимости поддержания и неявного регресса, который у вас уже есть. Цитата – Алан Пейдж, страница 104 в книге *Beautiful Testing*, O'Reilly Media Inc., Sebastopol 2010

[68] Рикард Эдгрен, статья *No Flourishes and New Connections Heuristics*, http://thetesteye.com/blog/2011/06/no-flourishes-and-new-connections-heuristics/

[69] Из переписки с Хенриком Эмильссоном. Надо отметить, что все эти эвристики – не новость: они многократно использовались тестировщиками.

[70] К примеру, урок 50 в книге Cem Kaner, James Bach, Bret Pettichord, *Lessons Learned in Software Testing*, John Wiley & Sons, Inc., New York 2002

16. Пять примеров из книги "Lessons Learned in Software Testing" – "Тестируйте на границах, тестируйте все сообщения об ошибках, тестируйте отличную от разработчика конфигурацию, прогоните нудные в настройке тесты, избегайте избыточных тестов[72]".
17. Соответствие информационным целям – можно ли подстроить тесты под различные информационные задачи?
18. Хорошие тесты – мощные, валидные, ценные, достойные доверия, вероятные, не избыточные, мотивирующие, выполнимые, поддерживаемые, повторяемые, важные, легкие в оценке, поддерживают разрешение проблем, подходящей сложности, объяснимые, малозатратные[73], легкие для понимания, позволяют совершать случайные открытия.
19. Изменение стратегии – если ваше тестирование не дает новой интересной информации, настало время менять стратегию.
20. Использование где-то еще? – можно ли применить эту отличную идею к другим областям?
21. Кто может это сделать? – вы можете включать тест-идеи, которые не можете выполнить самостоятельно.
22. Разнообразие ревью – если при ревью нет замечаний и дополнений, попросите того, кто мыслит иначе.
23. Непрочтенный тест-дизайн – если разработчики не читают ваши высокоуровневые тесты, спросите их, почему.
24. Любимая техника – ваша лучшая техника [74]тест-дизайна может неидеально подходить к ситуации, но она может стать той, которая даст наилучший результат.
25. Полезным будет любой навык, связанный с исследовательским тестированием, тестированием на основе опыта, и предположениями об ошибках.
26. Мета-вопросы – для сложных проблем тест-дизайна можно использовать вопросы вроде CIA Phoenix Checklist[75].

Тесты, достойные прогона

В какой-то момент вам нужно решить, какие именно тесты прогонять и оценивать. Это может быть сделано или обдумано в ходе анализа, синтеза или выполнения тестов. Возможно, вы заняты этим постоянно, и я полагаю, что вы выберете тесты, которые считаете наиболее важными. Это не магия – этот выбор включает множество аспектов: риск, скорость, возможность, перспективы, история, обеспечение случайных открытий, ценные свойства.

Не стоит тратить на приоритезацию тестов много времени – кажется естественным провести черту под названием "достойно тестирования".

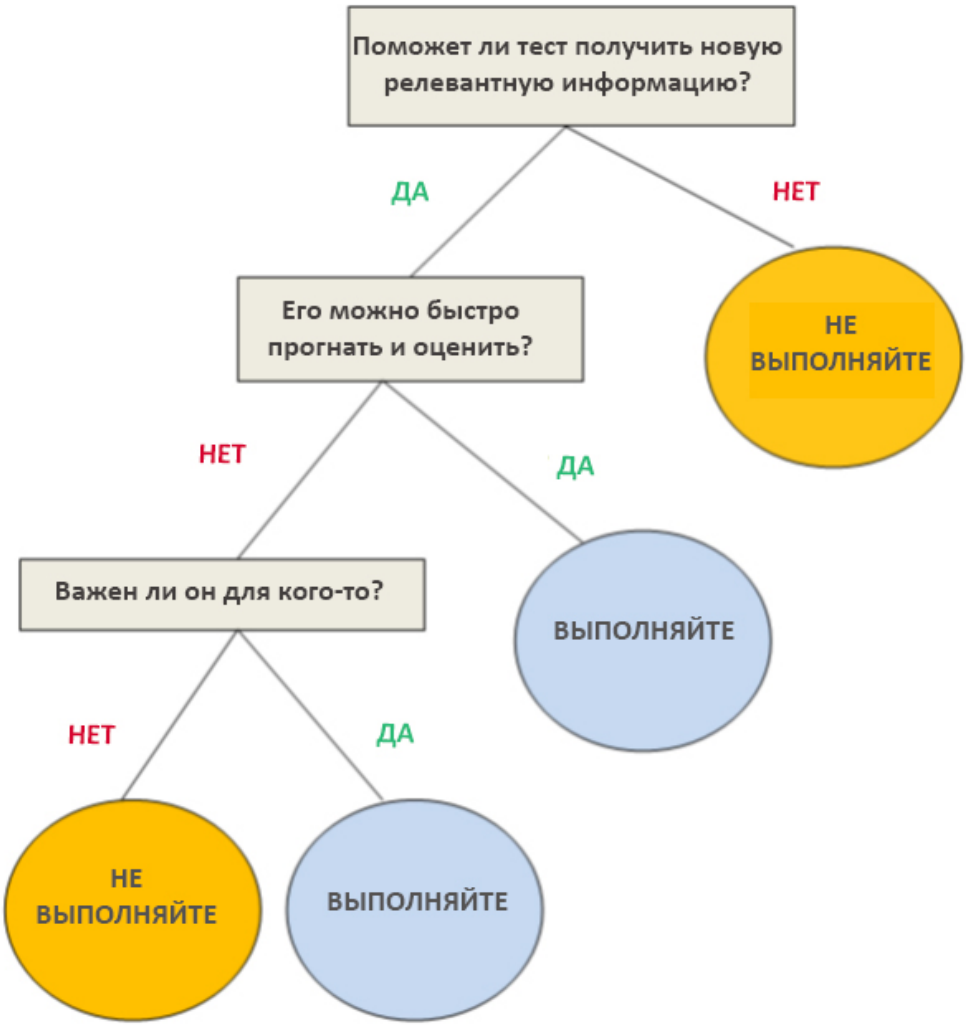
Тест достоин прогона, если мы считаем, что информация, которую мы можем получить, стоит потраченного времени вне зависимости от рисков, требований, подходов к тестированию, и т. д.

В книге "Adaptive Thinking[76]" Герд Джигерензер описывает свою теорию (исходный автор – Герберт Саймон), что интуиция основана на эвристиках, и что мы используем быстрые и практичные деревья решений, чтобы принимать хорошие **решения** в **сложном** мире и при **ограниченной** информации.

---

[71] Gerd Gigerenzer, *Gut Feelings: The Intelligence of the Unconscious*, Penguin Group, New York 2007
[72] Урок 38 в книге Cem Kaner, James Bach, Bret Pettichord, *Lessons Learned in Software Testing*, John Wiley & Sons, Inc., New York 2002
[73] Кем Кейнер, *What is a Good Test Case?*, STAR East, May 2003, http://www.kaner.com/pdfs/GoodTest.pdf
[74] Хороший обзор техник тест-дизайна: глава 3: Testing Techniques, Cem Kaner, James Bach, Bret Pettichord, *Lessons Learned in Software Testing*, John Wiley & Sons, Inc., New York 2002
[75] Страницы 139-141 в книге Michael Michalko, *Thinkertoys: a handbook of creative-thinking techniques*, Ten Speed Press, Berkeley 2006

Чтобы объяснить свои резоны и сделать вашу интуицию прозрачнее, можете попробовать использовать дерево решений [77]для сортировки тестов[78]:



Порядок и содержание ваших вопросов могут варьировать. Я, например, сначала оцениваю скорость, а потом важность, потому что я хочу иметь шанс найти то, о важности чего мы и не подозревали. С другой стороны, вопрос о релевантности может привести к удалению отличных тестов, но это нормально. Мы не можем выполнить абсолютно все тесты – есть вопрос баланса и оправданности потраченных на прогон усилий[79].

Цель этой книги – помочь тестировщикам выбрать из всего богатства возможностей использования ПО наиболее достойные тесты.

---

[76] Gerd Gigerenzer, *Adaptive Thinking: Rationality in the Real World*, Oxford University Press, New York 2000

[77] Это не идеальный метод, но он может дать наилучшие результаты, согласно книге: Gerd Gigerenzer, *Gut Feelings: The Intelligence of the Unconscious*, Penguin Group, New York 2007

[78] Больше про сортировку тестов: Robert Sabourin, *What Not to Test*, Better Software magazine November/December 2004, http://www.amibugshare.com/articles/Article_What_Not_To_Test.zip

[79] Эрик Якобсон, статья *Eight Things You May Not Need To Test*, http://www.testthisblog.com/2012/01/eight-things-you-may-not-need-to-test.html

Выполнение тестов

Часть тест-дизайна лучше отложить на этап выполнения тестов. Имея под рукой продукт, вы можете лучше принимать решения о деталях, а также поймете, какие вариации и отклонения будут быстрыми и полезными. Вы найдете часть нужной вам информации, а также информацию, о необходимости которой вы и не подозревали. Вы узнаете, какие области подвержены ошибкам, что можно быстро протестировать, о чем стоит узнать больше, что на самом деле можно делать с ПО, как все взаимосвязано, и, возможно, самое важное: что можно улучшить.

Используйте эту информацию в тест-дизайне, пересоздайте и перегруппируйте ваши тесты, вернитесь к источникам информации, которые оказались более важными, чем вы думали изначально.

Вам также понадобится множество детализированных вариаций для выявления возможных проблем, и проектирование и выполнение дополнительных тестов для создания хорошего отчета об ошибке.

Степень свободы

Это покрывается современным определением исследовательского тестирования:

"Исследовательское тестирование – это стиль тестирования, который ставит во главу угла личную свободу и ответственность конкретного тестировщика за постоянную оптимизацию ценности своей работы путем восприятия относящегося к тестированию обучения, тест-дизайна, выполнения тестов и интерпретации результатов тестов как взаимоподдерживающих видов деятельности, выполняющихся параллельно в ходе всего проекта[80]*".*

Если тестирование – это бесконечная проблема выборки, не стоит себя ничем ограничивать. Разнообразие требует свободы и способности мгновенно ухватиться за хорошую возможность.

При высокой степени свободы вы можете менять тест-окружение по необходимости, и максимизировать необходимое вам разнообразие путем вариаций и отклонений. Вы можете пользоваться неконтролируемыми окружениями и тестировщиками, потому что они дадут вам информацию, о нужности которой вы и не подозревали.

Эвристики выполнения тестов

Существует множество эвристик выполнения тестов. Тут я привожу те, которую считаю связанными с тест-дизайном[81].

1. Все эвристики анализа и дизайна могут использоваться для выполнения тестов.
2. Базовая эвристика – если оно существует, я хочу его протестировать[82].
3. Быстро разобраться с важными проблемами – старайтесь начать с поиска наиболее важных проблем[83].
4. Благодатный старт – вначале проверьте, хорошо ли работают простые, крупные участки.

---

[80] Кем Кейнер, статья *Defining Exploratory Testing*, http://www.satisfice.com/kaner/?p=42

[81] Много эвристик для прекращения тестирования – в блоге Майкла Болтона: http://www.developsense.com/blog/2009/09/when-do-we-stop-test/

[82] Продолжение: "(Исключая ситуацию, когда у меня есть дела поважнее"). Джеймс Бах, статья *Should Developers Test the Product First*, http://www.satisfice.com/blog/archives/54

[83] Урок 5 в книге: Cem Kaner, James Bach, Bret Pettichord, *Lessons Learned in Software Testing*, John Wiley & Sons, Inc., New York 2002

5. Широкая структуризация – если А работает хорошо (или плохо), можно пропустить Б.
6. Фоновая сложность – используйте "документацию", которая сложнее необходимого[84].
7. Сделай еще кое-что – дополнительно сделайте что-нибудь склонное к ошибкам, популярное, или характерное для пользователя. Не раздумывайте чересчур сильно, просто сделайте что-нибудь и посмотрите, что будет.
8. Эвристика мертвой пчелы – если я поменяю файл с данными, и он перестанет ронять тестируемое приложение, то следующий шаг – поменять его на предыдущий вариант и снова посмотреть на падение[85].
9. Шумовая полоса – если продукт делает нечто странное, то, возможно, дело идет к катастрофе[86].
10. Вприпрыжку – делайте что-нибудь намеренно сложным образом[87].
11. Пользовательские ошибки – изучите ненамеренные (типичные) ошибки.
12. Вживание в роль – **станьте** "настоящим" пользователем[88].
13. Бисекция [89]– если происходит что-то странное, удалите "половину", пока не найдете необходимые ингредиенты.
14. Эвристика опрокидывания [90]– попытайтесь извлечь пользу из непредусмотренного использования.
15. Базовая матрица конфигураций – идентифицируйте несколько платформ, покрывающих границы[91].
16. Одновременное выполнение тестов – прогоните несколько тестов одновременно для получения новых взаимодействий и идей.
17. Эвристика дежавю – столкнувшись с сообщением об ошибке, вызовите его еще раз (если оно не точно такое же, код – ваш подозреваемый[92]).
18. Ищите во множестве мест – результаты тестов можно интерпретировать в разных локациях.
19. Проблема не здесь – может ли найденная проблема существовать в других местах?
20. Наития – "Доверяйте вашим инстинктам. Проведите тест, который кажется многообещающим[93]".
21. Парное тестирование – совместное выполнение тестов стимулирует мыслить иначе и быстрее.
22. Помощник тестировщика – может ли мне прямо сейчас пригодиться инструмент?
23. Динамика полярностей [94]– переключайтесь между осторожным и быстрым, игривым и серьезным, объективным и субъективным, и т. п. подходами.
24. Прямо сейчас – "какой наилучший возможный тест я могу провести прямо сейчас[95]?"
25. Эмоции – используйте свои чувства[96].
26. Свежий взгляд найдет ошибку – смотрите на новое, давайте другим смотреть на ваше[97].

---

[84] Рикард Эдгрен, статья *Background Complexity and Do One More Thing Heuristics*, http://thetesteye.com/blog/2011/03/background-complexity-and-do-one-more-thing-heuristics/
[85] Джеймс Бах, статья *Dead Bee Heuristic*, http://www.satisfice.com/blog/archives/39
[86] Джеймс Бах, статья *Testing Heuristic: Rumble Strip*, http://www.satisfice.com/blog/archives/8
[87] Джеймс Бах, статья *Three New Testing Heuristics*, http://www.satisfice.com/blog/archives/467
[88] Роб Ламберт, статья *There's method in the madness*, http://thesocialtester.posterous.com/theres-method-in-the-madness
[89] Включена как фича в программу Джеймса Баха Perlclip, http://www.satisfice.com/tools/perlclip.zip
[90] Рикард Эдгрен, статья *Flipability Heuristic*, http://thetesteye.com/blog/2011/05/flipability-heuristic/
[91] Рикард Эдгрен и Хенрик Эмильссон, см. статью *BCM – Basic Configuration Matrix*, http://thetesteye.com/blog/2008/05/bcm-basic-configuration-matrix/
[92] Биджей Роллисон, статья *Boundary Testing – Hidden Loops and the Deja Vu Heuristic*, http://www.testingmentor.com/imtesty/2009/11/13/boundary-testing-hidden-loops-and-the-deja-vu-heuristic/
[93] Страница 6 в книге Cem Kaner, Jack Falk, and Hung Q. Nguyen, *Testing Computer Software, Second Edition*, John Wiley & Sons, Inc., New York 1999
[94] Много примеров полезных динамик в статье Джеймса Баха, Джона Баха, Майкла Болтона: *Exploratory Testing Dynamics* v2.2, http://www.satisfice.com/blog/wp-content/uploads/2009/10/et-dynamics22.pdf
[95] Джеймс Бах, статья *Exploratory Testing Explained*, http://www.satisfice.com/articles/et-article.pdf
[96] Майкл Болтон, *Lightning Talk on Emotions and Oracles*, http://www.developsense.com/2007/05/lightning-talk-on-emotions-and-oracles.html
[97] Урок 43 в книге Cem Kaner, James Bach, Bret Pettichord, *Lessons Learned in Software Testing*, John Wiley & Sons, Inc., New York 2002

27. Импровизируйте[98]!
28. Проблемная (не)определенность – проблемы есть всегда, но могут быть не важными.
29. Эвристика "Я могу ошибаться" – если я нахожу множество проблем, возможно, проблема в моей модели.

Быстрые тесты

Добавьте в ваш прогон немного быстрых и не всегда полезных тестов (взято у Кейнера/Баха[99]).

30. Тест ботинка – найдите поле ввода, переведите в него курсор, положите ботинок на клавиатуру и уйдите на обед.
31. Граничное тестирование – тестируйте на границах, потому что неверное кодирование границ – распространенная ошибка.
32. Атаки Уиттакера [100]– Ввод: принудительные сообщения об ошибках, значения по умолчанию, исследование данных, переполнение буфера. Ищите взаимодействия, повторяйте многократно. Вывод: различный, неверный, изменение свойств, обновление экрана. Изучите сохраненные данные и расчеты. Наполните или повредите файловую систему, проверьте невалидные и недоступные файлы.
33. Тестирование вмешательства – отмена, пауза, замена, прекращение, назад/далее, противоборство, удаление, выход из системы.
34. Протестируйте недавние изменения – возможны неожиданные побочные эффекты.
35. Исследуйте взаимоотношения данных – используйте зависимости, отслеживайте данные, вмешивайтесь, удаляйте.
36. Тур вариативности – варьируйте как можно больше во всех возможных направлениях.
37. Тур сложности – ищите наиболее сложные функции и данные, создавайте сложные файлы.
38. Тур образца данных – пользуйтесь любым доступным образцом везде, где это возможно. Чем сложнее, тем лучше.
39. Непрерывное использование – не перезагружайте систему при тестировании. Оставляйте окна и файлы открытыми. Повышайте использование диска и памяти в надежде, что система со временем самостоятельно завяжется узлами.
40. Правки – установите параметр на определенное значение, а затем, позднее, измените значение на что-нибудь еще, не перезагружая и не воссоздавая документ или структуру данных.
41. Куча мала – запустите много одновременных процессов и множество одновременно существующих состояний.
42. Подсечка – начните использовать функцию, когда система находится в подходящем состоянии, а затем на полпути измените состояние системы на неподходящее.
43. Наследие сообщений об ошибках – вызовите сообщения об ошибках. Тщательно тестируйте после того, как они закрылись.
44. Кликобезумие – тестирование, конечно, намного шире "стука по клавиатуре", но эта фраза родилась не просто так. Попробуйте постучать по клавиатуре[101]. Попробуйте покликать везде.
45. Множество экземпляров – запустите несколько копий приложения одновременно. Открывайте одни и те же файлы.
46. Взаимодействия функций – выясните, при каких условиях отдельные функции взаимодействуют или делятся данными. Ищите взаимозависимости. Пройдитесь по ним. Нагрузите их.

---

[98] Джонатан Коул, статья *Getting Started with Exploratory Testing - Part 2*, http://www.kohl.ca/blog/archives/000186.html

[99] Эти тесты – цитаты или интерпретации из http://www.testingeducation.org/BBST/riskbased/BBSTRisk2005.pdf. Многие были собраны на воркшопах LAWST, и см. также статью Джеймса Баха *Rapid vs. Hyper-Rapid Testing*, http://www.satisfice.com/blog/archives/9 и курс *Rapid Software Testing*, http://www.satisfice.com/rst.pdf

[100] Джеймс Уиттакер, *How to break Software: A Practical Guide to Testing*, Addison-Wesley, Boston 2002

[101] Используйте клавиши, которые часто значат что-то еще – например, F1, Ctrl+P, Alt+03

47. Дешевые инструменты! – учитесь использовать InCtrl5, Filemon, Regmon, AppVerifier, Perfmon, Task Manager, Threadhijacker, Zed Attack Proxy, Color Oracle (все они бесплатны).
48. Ресурсный голод – прогрессивно снижайте память и другие ресурсы, чтобы продукт грациозно деградировал (или неуклюже рухнул).
49. Игра "Так сказано в книге" – поищите онлайн-справку или руководство пользователя, и найдите инструкции для чего-нибудь интересного. Сделайте это, и на основании этого импровизируйте.
50. Безумные конфигурации – измените конфигурацию операционной системы нестандартным или необычным образом до или после установки продукта[102].
51. Гроккинг – найдите часть продукта, которая создает большие объемы данных или очень быстро выполняет какую-либо операцию.

Интерпретация

Что угодно – интерпретация. Цель тестирования – это не просто проставить "пройдено/упало", это мало что значит[103]. Вы должны быть открыты к важной информации и искать ее широким бреднем. Иногда необходимо начать тест, не имея ожидаемого результата или оракула под рукой, пользуясь вместо ними своим здравым смыслом и любопытством для определения деталей проблемы или поиска полезной и важной информации. Часто вам потребуется проводить последующие тесты для понимания, что значил результат предыдущих. Вы можете искать во многих местах, дабы избежать ошибок интерпретации.

Вы можете фиксировать все, что, с вашей точки зрения, важно для получателей. Тестировщики могут найти баги, риски, странные моменты, проблемы, артефакты, интересности, тесты, зависимости, вопросы, ценности, подходы, идеи, улучшения, обходные пути, модели, связи[104].

Если вы знаете несомненные идеальные способы, то, конечно, используйте их как оракулы.

Счастливые случайности

Даже при наилучшем тест-дизайне вы не сможете выяснить все значимое о продукте. Однако при открытом и заинтересованном выполнении тестов вы можете воспользоваться счастливыми случайностями – тестирование переполнено ими.

Если этого с вами не происходит, воспользуйтесь этой отличной практикой[105].

52. *Будьте открыты счастливым случайностям* – самые важные вещи часто находятся, когда вы ищете что-то другое. Если бы мы точно знали, где расположены все баги, нам бы хватило автоматизации или детальных тест-сценариев (или мы бы вообще не заморачивались с тестами). Поэтому убедитесь, что смотрите на картину в целом, везде, где только можно. Странное поведение может быть как нерелевантным, так и ключевым для получения важной информации. Тестирование всегда подразумевает выборку, и счастливые случайности -ваш друг и спаситель.

Такие случайности – это не удача, но немного удачи в них присутствует.

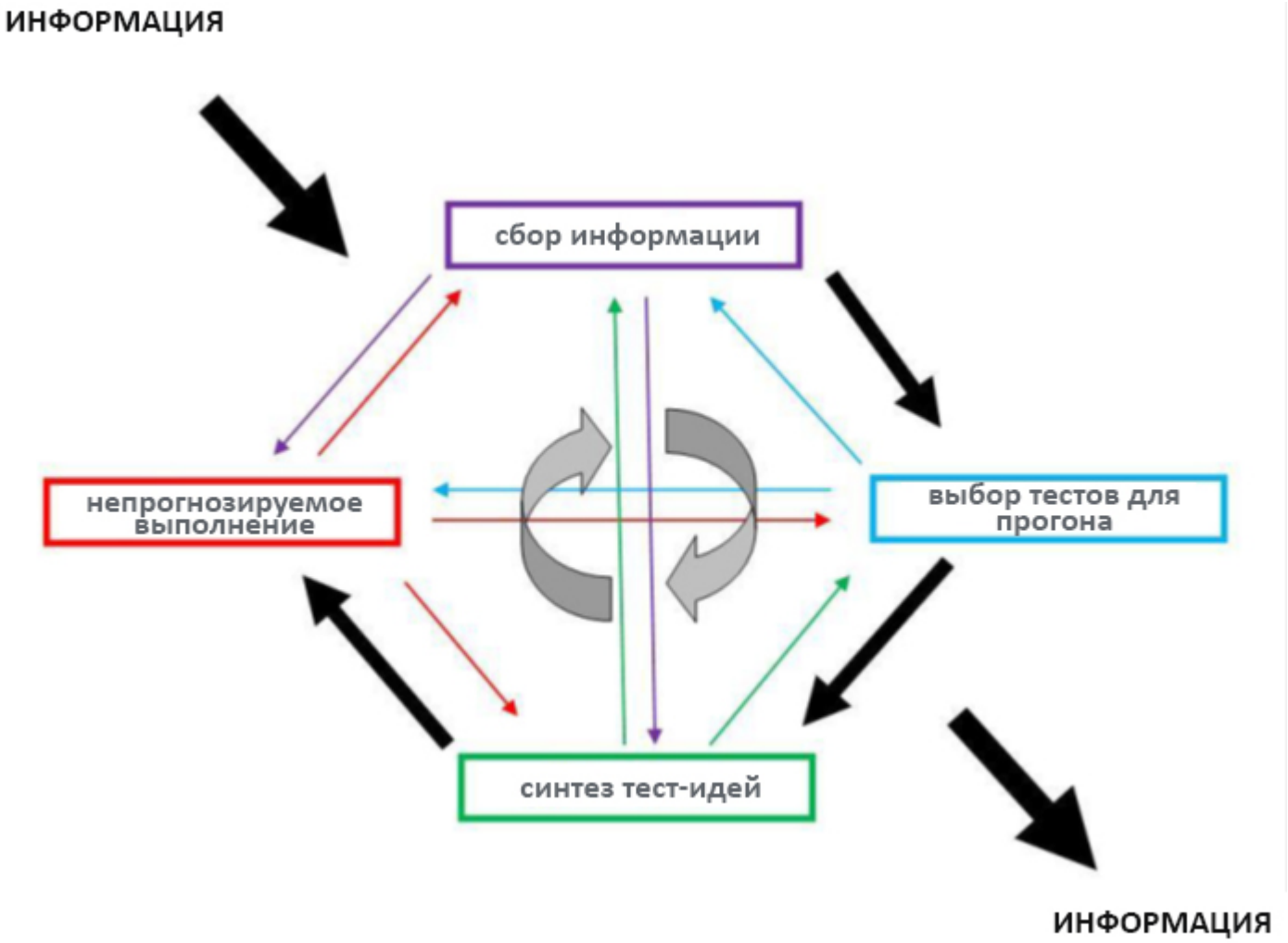
---

[102] См. также статью Рикарда Эдгрена An Error Prone Windows Machine http://thetesteye.com/blog/2008/04/an-error-prone-windows-machine/ - там есть важный тест (установка и/или запуск пользователем с ограниченными правами)
[103] Рикард Эдгрен, статья *Addicted to Pass/Fail?*, Tea-time with Testers, August 2011 – Issue 5, http://issuu.com/teatimewithtesters/docs/tea-time_with_testers_august_2011__year_1__issue_v
[104] Статья Джеймса Баха *What Testers Find* (с комментариями), http://www.satisfice.com/blog/archives/572 Развитие темы Майклом Болтоном: http://www.developsense.com/blog/2011/03/more-of-what-testers-find/ и http://www.developsense.com/blog/2011/04/more-of-what-testers-find-part-ii/
[105] Рикард Эдгрен, статья *Exploratory Testing Best Practices**, http://thetesteye.com/blog/2010/09/exploratory-testing-best-practices/

Заключение

Вышеописанные фазы тест-дизайна в реальности не происходят линейно. Они могут идти во всех направлениях, растягиваться на несколько релизов, и в них будет множество изменений и новых идей по мере интерпретации результатов тестирования.

Процесс скорее выглядит так:



Тестирование добавляет качество к информации о продукте

Самый неформальный тест-дизайн может неимоверно быстро происходить в голове опытного тестировщика, думающего о полностью новой идее в ходе исследовательского тестирования, или же тест-дизайн может быть основной для скриптов, кейсов и сценариев, которые тщательно исследованы и спроектированы заранее. Хорошее тестирование использует множество различных методов.

Идите по следу, ищите во многих местах, с разными перспективами; варьируйте многое или малое, прислушивайтесь к своим ощущениям, используйте известные вам данные, не перезагружайте систему, сделайте ее простой, сложной, импровизируйте!

Покрытие

Странно писать о тест-дизайне, не сказав ни слова о покрытии, однако репрезентативная модель покрытия как минимум крайне сложна, и ее невозможно получить в количественном формате, не удалив информацию

о том, что важно (а именно такого покрытия нужно добиваться). Возьмем, к примеру, покрытие кода – можно спросить, что важнее – тестировать код или полезность продукта.

Предложенный в книге тест-дизайн генерирует скорее качественную информацию, а она субъективна и чувствительна к тому, что важно, так же, как и пользовательские отношения с продуктом. Вы можете отчитаться о том, что протестировано, а что нет, и что важного вы нашли, не упоминая проценты покрытия.

С точки зрения тестирования у моделей покрытия две цели:

1. Идентификация того, что тестировать.
2. Принятие решения о завершении тестирования.

Как можно видеть из предыдущих глав, моделей и подходов для определения, что тестировать, великое множество. Сомнительно, что слияние всех этих и других моделей в многомерные модели покрытия или модель моделей принесет пользу. Что касается вопроса, когда остановиться – мы можем посмотреть на него с другой стороны:

Выполнение тестов может продолжаться до тех пор, пока не перестанет приносить новую, релевантную информацию. Измените стратегию или остановитесь: тестирование насыщено.

Если бы мне пришлось выбирать модель тестового покрытия, я бы стремился покрыть все, кажущееся важным.

Изъяны

Мне кажется, что в этой маленькой книжке много полезной информации, однако она далека от совершенства. Вот наиболее важные известные мне проблемы:

1. Первая претенциозная коллекция эвристик тест-дизайна, которую точно можно улучшить.
2. Не для всех – слишком много теории, высокая плотность, для многих нерелевантна.
3. Стиль тестирования "пройдено/упало" ожидается/требуется во множестве контекстов, где не будут работать альтернативные подходы.
4. Много двойной работы, особенно при фрагментированных зонах ответственности.
5. Полноценный живой пример обогатил бы голые идеи.

Although I believe this little book has a lot of useful information, it is far from perfect. These are the most important problems I know of:

Завершение

Тестировщики должны освободиться от ограничений тест-кейсов, основанных исключительно на требованиях и нацеленных только на верификацию. Они должны создавать множество типов тестов, видеть картину целиком, чтобы разглядеть важные детали.

Тесты не будут истинным отражением реальности, но, по моему мнению, у них неплохой шанс выявить реальные грядущие взаимоотношения между пользователем и ПО.

Прозрачность вашего тест-дизайна – основа отчетности.

Следует также стремиться к разнообразию задействованных людей. Это означает не только смешанную команду тестирования – это также значит, что другие роли должны различным образом участвовать в тестировании. Это нужно не потому, что они не доверяют тестировщикам, а потому, что все они стремятся

создать отличный продукт, и одна голова хорошо, а две лучше. По той же причине вы можете подключаться на ранних стадиях проекта и помочь со сбором, анализом и пониманием требований.

Я думаю, что я раскрыл наиболее важные аспекты тест-дизайна, известные мне. Это неполная информация, и не вся она будет для вас релевантна. Я надеюсь, что все, кто дочитал до конца, почерпнули хотя бы несколько новых идей. Если это не так – надеюсь, вы напишете свою статью и пришлете ее мне.

Библиография

Большая часть контента основана на опыте. Частично я обсуждал это с коллегами и другими тестировщиками. Я старался расставлять ссылки на книги, статьи, курсы или блоги там, где это было применимо.

Книги

James Bach, *Secrets of a Buccaneer-Scholar*, Scribner, New York 2009

Edward deBono, *Lateral Thinking - Creativity Step by Step*, 1990 Perennial edition

Lee Copeland, *A Practitioner's Guide to Software Test Design*, Artech House Publishers, Boston, 2003

Juliet Corbin and Anselm Strauss, *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*, Second Edition, SAGE Publications, Inc., Thousand Oaks 1998

Donald C. Gause/Gerald M. Weinberg, *Exploring Requirements: Quality Before Design*, Dorset House Publishing Co. 1989

Donald C. Gause/Gerald M. Weinberg, *Are Your Lights On? How to Figure Out What the Problem REALLY Is*, Dorset House Publishing Co. 1990

Gerd Gigerenzer, *Adaptive Thinking: Rationality in the Real World*, Oxford University Press, New York 2000

Gerd Gigerenzer, *Gut Feelings: The Intelligence of the Unconscious*, Penguin Group, New York 2007

Adam Goucher and Tim Riley, *Beautiful Testing*, O'Reilly Media Inc., Sebastopol 2010

Cem Kaner, Jack Falk, and Hung Q. Nguyen, *Testing Computer Software*, Second Edition, John Wiley & Sons, Inc., New York 1999

Cem Kaner, James Bach, Bret Pettichord, *Lessons Learned in Software Testing*, John Wiley & Sons, Inc., New York 2002

Michael Michalko, *Thinkertoys: a handbook of creative-thinking techniques*, Ten Speed Press, Berkeley 2006

Glenford Myers, *The Art of Software Testing*, Second Edition, John Wiley & Sons, Inc., Hoboken 2004 (originally published 1979)

Torbjörn Ryber, *Essential Software Test Design*, Fearless Consulting Kb, 2007

James Whittaker, *How to break Software: A Practical Guide to Testing*, Addison-Wesley, Boston 2002

Статьи

James Bach, *Exploratory Testing Explained*, http://www.satisfice.com/articles/et-article.pdf

James Bach, *Heuristic Test Strategy Model*, http://www.satisfice.com/tools/satisfice-tsm-4p.pdf

James Bach, Jon Bach, Michael Bolton, *Exploratory Testing Dynamics*, http://www.satisfice.com/blog/wp-content/uploads/2009/10/et-dynamics22.pdf

James Bach, Patrick J. Schroeder, *Pairwise Testing: a Best Practice That Isn't*, http://www.testingeducation.org/wtst5/PairwisePNSQC2004.pdf

Jonathan Bach, *Session-Based Test Management*, http://www.satisfice.com/articles/sbtm.pdf

Michael Bolton, *Test Framing*, http://www.developsense.com/resources/TestFraming.pdf

Michael Bolton, Better Software columns, http://www.developsense.com/publications.html

Fiona Charles, *Modeling Scenarios using Data*, http://www.quality-intelligence.net/articles/Modelling%20Scenarios%20Using%20Data_Paper_Fiona%20Charles_CAST%202009_Final.pdf

Rikard Edgren, Henrik Emilsson and Martin Jansson, *Software Quality Characteristics*, http://thetesteye.com/posters/TheTestEye_SoftwareQualityCharacteristics.pdf

Rikard Edgren, Henrik Emilsson and Martin Jansson, *37 Sources for Test Ideas* , http://thetesteye.com/posters/TheTestEye_SourcesforTestIdeas.pdf

Rikard Edgren, *Addicted to Pass/Fail?, http://issuu.com/teatimewithtesters/docs/tea-time_with_testers_august_2011__year_1__issue_v*

Rikard Edgren, *More and Better Test Ideas*, http://www.thetesteye.com/papers/redgren_moreandbettertestideas.pdf

Rikard Edgren, *Testing is an Island, A Software Testing Dystopia*, http://thetesteye.com/papers/redgren_testingisanisland.pdf

Rikard Edgren, *The Eye of a Skilled Software Tester*, http://wiki.softwaretestingclub.com/w/file/fetch/39449474/TheTestingPlanet-Issue4-March2011.pdf

Rikard Edgren, *Where Testing Creativity Grows*, http://thetesteye.com/papers/where_testing_creativity_grows.pdf

Elisabeth Hendrickson, *Test Heuristics Cheat Sheet*, http://testobsessed.com/wp-content/uploads/2011/04/testheuristicscheatsheetv1.pdf

Michael Hunter, *You Are Not Done Yet*, http://www.thebraidytester.com/downloads/YouAreNotDoneYet.pdf

Cem Kaner, *The Ongoing Revolution in Software Testing*, http://www.kaner.com/pdfs/TheOngoingRevolution.pdf

Cem Kaner, *What is a Good Test Case?*, http://www.kaner.com/pdfs/GoodTest.pdf

Cem Kaner, *An Introduction to Scenario Testing*, http://www.kaner.com/pdfs/ScenarioIntroVer4.pdf

Joris Meerts, *Functional Testing Heuristics - A Systems Perspective*, http://www.testingreferences.com/docs/Functional_Testing_Heuristics.pdf

Robert Sabourin, *10 Sources of Testing Ideas*, http://www.amibugshare.com/articles/Article_10_Sources_of_Testing_Ideas.pdf

Robert Sabourin, *What Not to Test*, http://www.amibugshare.com/articles/Article_What_Not_To_Test.zip

Neil Thompson & Mike Smith, *The Keystone to Support a Generic Test Process: Separating the "What" from the "How"*, http://doi.ieeecomputersociety.org/10.1109/ICSTW.2008.46

Курсы

James Bach, Michael Bolton, *Rapid Software Testing*, http://www.satisfice.com/rst.pdf

Michael Bolton, *Lightning Talk on Emotions and Oracles*, http://www.developsense.com/2007/05/lightning-talk-on-emotions-and-oracles.html

Michael Bolton, *Confirmation Bias*, http://www.developsense.com/presentations/2010-04-ConfirmationBias.pdf

Cem Kaner, *Developing Skills as an Exploratory Tester*, http://www.kaner.com/pdfs/ExploratorySkillsQAI2007.pdf

Cem Kaner & James Bach, *Risk-Based Testing: Some Basic Concepts*, http://www.kaner.com/pdfs/QAIRiskBasics.pdf

Cem Kaner & James Bach, *Black Box Software Testing, Exploratory Testing*, http://www.testingeducation.org/BBST/exploratory/BBSTExploring.pdf

Cem Kaner, *Software Testing as a Social Science*, http://www.kaner.com/pdfs/KanerSocialScienceTASSQ.pdf

Cem Kaner & James Bach, *Black Box Software Testing, Specification-Based Testing* http://www.testingeducation.org/BBST/specbased/BBSTspecBased.pdf

Cem Kaner & James Bach, *Black Box Software Testing, Risk-Based Testing* http://www.testingeducation.org/BBST/riskbased/BBSTRisk2005.pdf

Cem Kaner & Rebecca Fiedler, *Black Box Software Testing, Test Design* http://www.testingeducation.org/BBST/testdesign/BBSTTestDesign2011pfinal.pdf

Robert Sabourin, *Just-in-time testing*, http://www.amibugshare.com/courses/Course_Just_In_Time_Testing.zip

Блоги

Blog portal http://www.testingreferences.com

Michael Bolton, http://www.developsense.com/blog/

Context-Driven Testing, http://www.context-driven-testing.com/

Rikard Edgren, Henrik Emilsson, Martin Jansson, http://thetesteye.com/blog/

Cem Kaner, http://www.satisfice.com/kaner/

Jonathan Kohl, http://www.kohl.ca/blog/

Rob Lambert, http://thesocialtester.posterous.com/

Quick Testing Tips, http://www.quicktestingtips.com