КЛАССИФИКАЦИЯ ВИДОВ ТЕСТИРОВАНИЯ

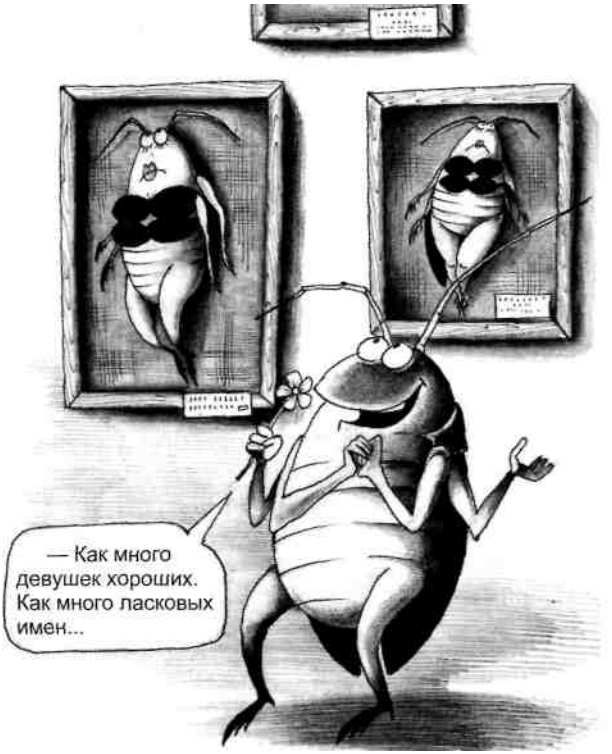
- ПО ЗНАНИЮ ВНУТРЕННОСТЕЙ СИСТЕМЫ
- ПО ОБЪЕКТУ ТЕСТИРОВАНИЯ
- ПО СУБЪЕКТУ ТЕСТИРОВАНИЯ
- ПО ВРЕМЕНИ ПРОВЕДЕНИЯ ТЕСТИРОВАНИЯ
- ПО КРИТЕРИЮ "ПОЗИТИВНОСТИ" СЦЕНАРИЕВ
- ПО СТЕПЕНИ ИЗОЛИРОВАННОСТИ ТЕСТИРУЕМЫХ КОМПОНЕНТОВ
- ПО СТЕПЕНИ АВТОМАТИЗИРОВАННОСТИ ТЕСТИРОВАНИЯ
- ПО СТЕПЕНИ ПОДГОТОВКИ К ТЕСТИРОВАНИЮ

Любая классификация составляется по определенному признаку, например:

- **по полу** люди делятся (классифицируются) на мужчин и женщин;
- **по наличию** кошки люди делятся на тех, у кого кошка есть, и тех, у кого ее нет;
- **по росту** люди делятся на группы в зависимости от количества сантиметров от земли до макушки (например, один будет в группе "181 см", а другой — в группе "185 см").

Один и тот же субъект может быть одновременно элементом бесчисленного количества классификаций, при этом прекрасно себя чувствовать и не испытывать никаких угрызений совести. Например, дебошир и романтик Сева Б. может одновременно

- **быть** мужчиной,
- **иметь** кошку и
- **вырасти** до 175 см.


— Как много девушек хороших. Как много ласковых имен...

Немедленная польза от классификаций в отношении видов тестирования заключается в том, что упорядоченная и обобщенная информация легче воспринимается, усваивается и запоминается.

Замечу, что видов тестирования существует огромное количество и мы не будем пытаться объять необъятное, а поговорим об основных видах, которых, впрочем, и так хватит с лихвой для любого интернет-проекта.

Сначала перечислим, потом объясним. Объяснения призваны дать общее понимание каждого из элементов, в то время как последующие разговоры это понимание расширят и углубят.

Формат изложения:

Классификация по этому признаку
состоит из следующих элементов.

Перечисляем:

1. **По знанию внутренностей системы:**
   - черный ящик *(black box testing);*
   - серый ящик *(grey box testing);*
   - белый ящик *(white box testing).*
2. **По объекту тестирования:**
   - функциональное тестирование *(functional testing);*
   - тестирование интерфейса пользователя *(UI testing);*
   - тестирование локализации *(localization testing);*
   - тестирование скорости и надежности *(load/stress/performance testing);*
   - тестирование безопасности *(security testing);*
   - тестирование опыта пользователя *(usability testing);*
   - тестирование совместимости *(compatibility testing).*
3. **По субъекту тестирования:**
   - альфа-тестировщик *(alpha tester);*
   - бета-тестировщик *(beta tester).*
4. **По времени проведения тестирования:**
   - **до** передачи пользователю — альфа-тестирование *(alpha-testing);*
     - тест приемки *(smoke test, sanity test* или *confidence test);*
     - тестирование новых функциональностей *(new feature testing);*

- регрессивное тестирование *(regression testing);*
- тест сдачи *(acceptance or certification test);*

• **после** передачи пользователю — бета-тестирование *(beta testing).*

5. По критерию "позитивности" сценариев:

• позитивное тестирование *(positive testing);*
• негативное тестирование *(negative testing).*

6. По степени изолированности тестируемых компонентов:

• компонентное тестирование *(component testing);*
• интеграционное тестирование *(integration testing);*
• системное (или энд-ту-энд) тестирование *(system or end-to-end testing).*

7. По степени автоматизированности тестирования:

• ручное тестирование *(manual testing);*
• автоматизированное тестирование *(automated testing);*
• смешанное/полуавтоматизированное тестирование *(semi automated testing).*

8. По степени подготовки к тестированию:

• тестирование по документации *(formal/documented testing);*
• эд хок-тестирование *(ad hoc testing).*

Объясняем:

1. По знанию внутренностей системы

• черноящичное тестирование *(black box testing);*
• белоящичное тестирование *(white box testing);*
• сероящичное тестирование *(grey box testing).*

Кстати, в отношении четких дефиниций, водоразделов и прочих академических штучек до сих пор идут споры.

ЧЕРНЫЙ ЯЩИК *(black box)*

Должен признаться, что лучшие мгновения моего студенчества прошли не в аудиториях моей альма-матер, не в залах библиотек, а в пивной Коптевского рынка, куда мы с Балмашновым, Гнездиловым, Дебдой, Ермохиным, Илюхиным, Карповым, Назаровым,

Осмоловским, Сапачевым и Тарасовым вламывались с тубусами наперевес и за вечер выполняли недельный план по продажам.

Основным элементом Коптевской пивной того времени была так называемая автопоилка, т.е. аппарат, принимающий жетон и выдающий пол-литра того, что мы тогда считали пивом.

Так вот если перевести манипуляции с автопоилкой на компьютерный язык, то

- жетон был **вводом,**
- пиво — **выводом,**
- щель для жетона и носик для пива — **интерфейсом пользователя,** а
- механизм автопоилки, обменивающий жетон на пиво, — **черным ящиком,** так как мы **не знали** (и для сохранения аппетита не хотели знать), **как был устроен изнутри** тот столь необходимый для студента аппарат.

В отношении ПО черный ящик, т.е. **область незнания,** — это не что иное, как тестируемые части бэк-энда (например, код программиста, схема базы данных), составляющие невидимый пользователю **виртуальный мост,** который соединяет

- **фактический ввод** (шаги) и
- **фактический вывод** (фактический результат).

Признаки подхода "Черный ящик":

1. **Тестировщик не знает, как устроен виртуальный мост.**
2. **ИДЕИ для тестирования идут от предполагаемых паттернов** *(pattern* — образец) **поведения пользователей.** Поэтому подход "Черный ящик" также называют **поведенческим.**

Разберем первый признак.

1. ТЕСТИРОВЩИК НЕ ЗНАЕТ, КАК УСТРОЕН ВИРТУАЛЬНЫЙ МОСТ

С одной стороны,

тестировщик имеет преимущество перед программистом, т.е. автором кода. Давайте будем честны перед собой: **мы часто принимаем желаемое за действительное.** Особенно это касается того, что мы создали сами, например воображаемого образа любимого человека. Сколько раз каждый из нас заводил романы с абсолютно неправильными, несовместимыми и нередко вредными для нас

людьми и утешал себя, что *it's o'k* — притрется, пригладится и поймется. Как показывает жизнь, притирки, пригладки и понимания ни к чему хорошему не приводят, как и попытки заставить программиста произвести функциональное тестирование.

Вот перевод постинга на одном из форумов по тестированию:

"Программист не должен проводить тестирование, и давать релизу зеленый свет. Нужно, чтобы кто-то независимый (человек/отдел) был ответствен за поиск багов и уполномочен "доставать " программиста до тех пор, пока баг не будет починен. Дело в том, что я как программист знаю свой код, и если я сам провожу тестирование, то обязательно буду делать допущения, что какие-либо части кода работают по умолчанию и их можно не проверять. С другой стороны, мои тесты основаны на моем понимании того, как работает код, и не учитывают реальных и порой абсолютно нелогичных вещей, которые будут делаться пользователями с моим кодом. С третьей стороны, у меня на тестирование нет времени, и я не понимаю, почему должен проводить тестирование, если за него платят тестировщикам ".

Реальность — это мир, пропущенный через призму субъективного восприятия. Например, каждый родитель свято верит, что его ребенок самый умный, талантливый и перспективный. Код — это дитя программиста, и **в своей реальности** программист нередко воспринимает код как априорно непогрешимый.

Вот вам легенда про призму восприятия:

Когда на пути в Индию корабли Колумба остановились перед одним из Карибских островов, индейцы... этих кораблей не увидели, потому что их призмы восприятия не пропускали образы, абсолютно чуждые тем предметам и явлениям, с которыми они и их предки существовали бок о бок на протяжении тысячелетий. И лишь шаман, учуяв что-то неладное, несколько дней пристально всматривался в горизонт, пока наконец не отделил романтические силуэты испанских фрегатов от привычных океана и неба и не сказал своей пастве: "Опа! Корабли Колумба " (тут, конечно, все сразу настроили свои призмы, увидели не замеченные раньше корабли, деловито погрузили в лодки свиней и поехали менять их на бусы).

Идея, думаю, понятна. Программист пишет, тестировщик тестирует, Филипп Филиппыч оперирует, Айседора Дункан танцует, и никаких разрух.

Итак, блэк бокс-тестировщику, знающему лишь то, для **чего** был написан код (т.е. функциональности), а не **как** он был написан, легче смотреть на тестирование с точки зрения пользователя, для удовлетворения чаяний которого весь софтверный сыр-бор и начался.

С другой стороны,

блэк бокс-тестирование ведется вслепую, так как ни одна из частей виртуального моста неизвестна. Следствием этого может стать ситуация, когда для вещи, проверяемой одним тест-кейсом, пишется несколько тест-кейсов.

Итак, **в случае с черным ящиком тестировщик не знает, как устроен виртуальный мост,** и это может быть как полезно, так и вредно для дела.

Разберем второй признак.

2. ИДЕИ ДЛЯ ТЕСТИРОВАНИЯ ИДУТ ОТ ПРЕДПОЛАГАЕМЫХ ПАТТЕРНОВ *(pattern* — образец) ПОВЕДЕНИЯ ПОЛЬЗОВАТЕЛЕЙ

То, что мы называли вводом (шагами), на самом деле является двумя вещами, которые так же неотрывно связаны, как судьбы Ромео и Джульетты. Речь идет о

сценариях и
данных для сценариев.

Исполнение тестирования может проходить как при наличии, так и без тест-кейсов. Так вот в обоих случаях **сценарий** *(scenario)* — **это последовательность ДЕЙСТВИЙ для достижения фактического результата.**

Пример сценария

1. *Открой www.main.testshop.rs.*
2. *Кликни линк "contact us".*

Если исполнение тестирования идет по тест-кейсам, то можно сказать, что сценарий тест-кейса — это совокупность шагов тест-кейса.

Данные для сценариев, или просто "данные", — это конкретные ЗНАЧЕНИЯ ВВОДА, используемые для достижения фактического результата.

Пример данных

1. *Открой www.main.testshop.rs.*
2. *Введи текст "Затоваренная бочкотара" в поле поиска.*
3. *Нажми кнопку "Искать".*

В последнем примере шаги 1 —3 (включительно) были сценарием, а "Затоваренная бочкотара" — данными.

*Еще один **пример данных***

При закрытии счета в одном из интернет-магазинов на последней странице пользователь должен ответить, почему он закрывает счет. Ему дается список из 20 вопросов, и напротив каждого вопроса размещен квадрат, куда можно поставить галочку (checkbox). Так вот если пользователь поставит галочку напротив строк "Служба поддержки" и "Медленная доставка" и нажмет на кнопку "Закрыть счет", то данными будет текст "Служба поддержки " и " Медленная доставка".

Совместим знания о сценариях и данных со вторым признаком подхода "Черный ящик".

Предполагаемые паттерны поведения пользователей — это те **сценарии и данные,** которые, **как мы ожидаем,** будут реализовываться и вводиться пользователями.

Основные источники предполагаемых паттернов поведения пользователей могут быть:

а) **напрямую взяты из спека.**

Пример

Пункт 12 спека #9548 говорит: "Если на странице с регистрационной формой пользователь не указал свой е-мейл, то после нажатия на кнопку "Зарегистрироваться" показывается та же страница, но с сообщением об ошибке: "Пожалуйста, введите ваш е-мейл" и с изменением шрифта имени текстового поля "Е-мейл:" на красный цвет".

Напишем тест-кейс.

ИДЕЯ: "Сообщение об ошибке, если при регистрации не указан е-мейл".

Сценарий:

1. *Открой wvwv.main.testshop.rs/register.htm.*
2. *Заполни все текстовые поля кроме "Е-мейл:" действительными данными (поле "Е-мейл:"должно быть пустым).*
3. *Нажми на кнопку "Зарегистрироваться".*

Ожидаемый результат 1:
Страница регистрации.

Ожидаемый результат 2:
Сообщение об ошибке "Пожалуйста, введите ваш е-мейл".

Ожидаемый результат 3:
Шрифт имени поля "Е-мейл:" изменен на красный цвет.

***Кстати,** данными для сценария из последнего примера послужили две вещи: 1) действительный ввод всех полей, кроме е-мейла (мы предполагаем, что лицо, исполняющее тест-кейс, знает легитимные значения ввода), и 2) пустое поле для е-мейла. Значение ввода "" — это тоже вид данных.*

Давайте для простоты в дальнейшем использовать ***термин "сценарий" в качестве собирательного образа, т.е. самого сценария и данных, используемых в нем;***

б) **найдены путем эксплоринга.**

Иногда "брожение" по сайту является лучшим источником для понимания того, как реальный пользователь будет **с** ним обращаться;

в) **получены путем применения методики черноящичного тестирования** *(black box testing methodology)*.

Примеры: впереди будет много примеров;

г) **подарены интуицией.**

Помните, как у Конан Дойля было сказано об инспекторе Лестрейде? Примерно так: "Но была единственная вещь, которая мешала ему стать настоящим сыщиком, — у него не было чутья".

А чем мы не сыщики? **Интуиция не менее важна для настоящего профессионала-тестировщика, чем прикладные знания и опыт работы;**

д) присоветованы программистом или продюсером.

Общение, общение и еще раз общение. Самое дорогое — это информация, и общение — один из главных ее источников. Продюсер, программист и тестировщик дают путевку в жизнь одной и той же функциональности, но каждый смотрит на нее со своей колокольни, и если нам, тестировщикам, получить мнения товарищей с двух других колоколен, то можно узнать потрясающе полезные вещи;

е) **др.**

Например, мы прочитали статью в Интернете, давшую классную идею для сценария.

Итак, мы разобрались со вторым признаком подхода "Черный ящик".

Обобщаем.

При подходе "Черный ящик" тестировщик не основывает идеи для тестирования на знании об устройстве и логике тестируемой части бэк-энда. Идеи формируются путем предпо-

ложений о сценариях, которые будут реализовываться и применяться пользователями. Такие сценарии называются паттернами поведения пользователей.

БЕЛЫЙ ЯЩИК *(white box)*

также известен под именами Стеклянный ящик *(glass/clear box),* Открытый ящик *(open box)* и даже Никакой ящик *(no box).*

В отличие от "Черного ящика" **при подходе "Белый ящик" тестировщик основывает идеи для тестирования на знании об устройстве и логике тестируемой части бэк-энда.**

Таким образом, при белоящичном тестировании сценарии создаются с мыслью о том, чтобы протестировать определенную часть бэк-энда, а не определенный паттерн поведения пользователя.

> ***Пример** из жизни*
>
> *Допустим, нужно протестировать проходимость нового российского внедорожника.*
>
> *При подходе "Черный ящик" тестировщик садится за руль, выезжает за кольцевую — в объятия подмосковной осени, находит непролазную канаву, заезжает в нее и пытается выбраться, т.е. он проделывает вещи, которые с большой вероятностью будут проделаны основными пользователями таких машин — охотниками, рыболовами и рэкетирами.*
>
> *При подходе "Белый ящик" тестировщик открывает капот и видит, что установлена система полного привода фирмы "Джапан моторз", модель RT6511. Тестировщик **знает,** что проходимость внедорожника зависит именно от RT6511 и ее слабое место — это эффективность при езде по снегу. Что делает тестировщик? Правильно! Выезжает на белую сверкающую гладь русского поля и насилует джип в свое удовольствие.*

Последний пример не только служит иллюстрацией разницы в подходах, но и показывает, что использование методик обоих подходов количественно и качественно увеличивает покрытие возможных сценариев.

Идем дальше.

> ***Постановка мозгов***
>
> *Покрытие возможных сценариев — это одна из частей архиважнейшей концепции, называемой **тестировочное покрытие.***
>
> *Забудем на минуту о ПО вообще и о тестировании в частности.*
>
> *Представим себе шахматную доску, состоящую из 64 клеток. Единственная фигура, присутствующая на доске, — белый король. Допустим,*

*каждая возможная позиция короля записана на отдельной карточке: "Поставь белого короля на такую-то клетку". Следовательно, у нас есть 64 карточки, или 100% **теоретически** возможных вариантов расположения короля. Если мы будем перемещать короля в соответствии с позициями на карточках, то, последовательно перелистав все карточки, добьемся 100%-й **практической** реализации предписаний, указанных на карточках.*

Теперь усложним задачу и представим, что у нас есть шахматная доска, количество клеток на которой так велико, что не поддается подсчету. Допустим, что, согласно лишь нам известной логике, в голову нам ударило выбрать лишь 20 позиций, которые мы опять же зафиксировали на карточках. Теперь вопрос: покрывают ли 20 карточек 100% теоретически возможных вариантов расположения короля? Нет. Можем ли мы на 100%о практически реализовать предписания, указанные на 20 карточках? Да.

Обратно к тестированию ПО.

***Тестировочное покрытие** (test coverage) состоит из двух вещей:*

а. Покрытие возможных сценариев.
б. Покрытие исполнения тест-кейсов.

***Покрытие возможных сценариев** — это в большинстве случаев абстрактная величина, так как в большинстве же случаев невозможно даже подсчитать, сколько понадобится тест-кейсов, чтобы обеспечить 100%-ю проверку ПО (например, попробуйте подсчитать количество **всех** теоретически возможных тест-кейсов для тестирования Майкрософт Ворда-2003).*

*Другими словами, **в большинстве случаев** покрытие возможных сценариев нельзя представить как процентное отношение сценариев, зафиксированных в тест-кейсах, ко всем теоретически возможным сценариям.*

Покрытие возможных сценариев может увеличиться либо уменьшиться путем прибавления либо отнятия уникального тест-кейса, т.е. тест-кейса,

- *который тестирует реальный сценарий использования ПО и*
- *который не является дубликатом другого тест-кейса.*

***Покрытие исполнения тест-кейсов** — это **всегда величина конкретная,** и выражается она процентным отношением исполненных тест-кейсов к общему количеству тест-кейсов. Допустим, тест-комплект для тестирования функциональностей спека #1243 "Новые функциональности корзины" состоит из 14 тест-кейсов, и если 7 из них исполнены, то покрытие исполнения тест-кейсов равно 50%>.*

Возвращаемся к нашим ящикам.

Симбиоз использования подходов "Черный ящик" и "Белый ящик" увеличивает **покрытие возможных сценариев**

- **количественно,** потому что появляется большее количество тест-кейсов;

- **качественно,** потому что ПО тестируется принципиаль но разными подходами: с точки зрения пользователя ("Черный ящик") и с точки зрения внутренностей бэк-энда ("Белый ящик").

В реальной жизни белоящичное тестирование проводится либо самими программистами, написавшими код, либо их коллегами с программистской квалификацией того же уровня. Кстати, юнит-тестирование, о котором мы говорили, — это часть *white box-mec*тирования.

СЕРЫЙ ЯЩИК *(gray/grey box)*

Это подход, сочетающий элементы двух предыдущих подходов, это

- *с одной стороны,* тестирование, ориентированное на пользователя, а значит, мы используем паттерны поведения пользователя, т.е. **применяем методику "Черного ящика";**
- *с другой* — **информированное тестирование,** т.е. мы знаем, как устроена хотя бы часть тестируемого бэк-энда, и активно **используем** это знание.

*Ярчайший **пример***

Допустим, мы тестируем функциональность "регистрация":

- *заполняем все поля (имя, адрес, е-мейл и т.д.) и*
- *нажимаем кнопку "Зарегистрироваться".*

Следующая страница — подтверждение, мол, дорогой Иван Иваныч, поздравляем, вы зарегистрированы.

*Теперь **вопрос:** если мы видим страницу с подтверждением регистрации, то значит ли это, что регистрация была успешной? **Ответ:** нет, так как процесс регистрации с точки зрения нашей системы включает не только подтверждение на веб-странице, но и создание записи в базе данных,*

*т. е. **вывод, который стоит проверить, состоит из***

- ***страницы с подтверждением и***
- ***новой записи в базе данных.***

Откуда мы почерпнем знание о логике создания записей в базе данных при регистрации? Например, из технической документации (документ о дизайне/архитектуре системы, документ о дизайне кода), общения с программистом, самого кода.

Как видно из последнего примера, подход "Серый ящик" — это дело хорошее, жизненное и эффективное. Деятельность большинства профессиональных тестировщиков интернет-проектов протекает именно в разрезе сероящичного тестирования.

Пара мыслей вдогонку.

1. Когда мы говорим о поведенческом тестировании, то это не значит, что тестировщик ограничен набором действий, совершаемых пользователем. Во многих случаях специально написанный код используется для облегчения тестирования или для того, чтобы вообще сделать его возможным.

> ***Пример***
>
> *При разговоре о формальной стороне тест-кейса мы проверяли баланс кредитной карты до и после покупки на странице www.main.testshop.rs /<четыре_последних_цифры_карты>/balance.htm. В реальности пользователь проверяет баланс кредитной карты на сайте кредитной организации, выдавшей эту карту (например, www.wellsfargo.com), а страница balance.htm является* ***специальным кодом, написанным для тестирования*** *с использованием несуществующих кредитных карт.*
>
> ***Кстати,*** *тот факт, что тестировщик использует информацию веб-страницы balance.htm, не означает, что он понимает логику работы кода, отвечающего за списание денег со счета.*

2. Как мы видели на примере с регистрацией, выводом, который нужно было проверить для реального тестирования, послужила не только страница с подтверждением, но и запись в базе данных. **Так как ожидаемый вывод — это ожидаемый результат наших тест-кейсов, то огромное значение для эффективности тестирования имеет поиск именно того ожидаемого результата, который реально подтвердит, что код работает.** Так, если бы в том же самом примере ожидаемым результатом была только страница с подтверждением, то проверка базы данных была бы лишь тратой времени.

2. По объекту тестирования

- Функциональное тестирование *(functional testing);*
- Тестирование интерфейса пользователя *(UI testing);*
- Тестирование локализации *(localization testing);*
- Тестирование скорости и надежности *(load/stress/ performance testing);*
- Тестирование безопасности *(security testing);*
- Тестирование опыта пользователя *(usability testing);*
- Тестирование совместимости *(compatibility testing).*

ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ *(functional testing)*

Уже говорили и еще будем много говорить.

ТЕСТИРОВАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ
(UI (читается как "ю-ай") *testing)*

Это тестирование, при котором проверяются элементы интерфейса пользователя. Мы рассмотрим все основные элементы веб-интерфейса при разговоре о системе трэкинга багов.

Важно понимать разницу между

тестированием **интерфейса** пользователя и
тестированием **с помощью интерфейса** пользователя.

***Пример** первого*

Проверяем максимальное количество символов, которые можно напечатать в поле "Имя" на странице "Регистрация", т.е. проверяем, отвечает ли конкретный элемент интерфейса, называющийся "однострочное текстовое поле" (textbox), требованию спецификации, которая указывает на максимальное количество символов, которое в этом поле можно напечатать.

***Пример** второго*

Тестируем бэк-энд и с помощью интерфейса создаем транзакцию покупки, т.е. мы использовали интерфейс пользователя как инструмент для создания транзакции.

ТЕСТИРОВАНИЕ ЛОКАЛИЗАЦИИ
(localization testing)

Многогранная вещь, подразумевающая проверку множества аспектов, связанных с **адаптацией** сайта для пользователей из других стран. Например, тестирование локализации для пользователей из Японии может заключаться в проверке того, не выдаст ли система ошибку, если этот пользователь на сайте знакомств введет рассказ о себе символами *Kanji,* а не английским шрифтом.

ТЕСТИРОВАНИЕ СКОРОСТИ И НАДЕЖНОСТИ
(load/stress/performance testing)

Это проверка поведения веб-сайта (или его отдельных частей) при одновременном наплыве множества пользователей.

У каждого, кто пользуется Интернетом, есть опыт ожидания, когда, например, кликаешь на линк и следующая страница медленно высасывает из тебя душу, загружаясь ну оче-е-е-е-нь долго.

Плохой перформанс (скорость работы) — это основная беда российских интернет-проектов.

Менеджмент, который экономит на подобном тестировании, в итоге, как правило, глубоко сожалеет об этом, так как современный интернет-пользователь — это существо ранимое и нервное, если сайт работает медленно, с перебоями или не работает совсем, так как не справляется с наплывом посетителей, то современный интернет-пользователь идет куда? Правильно, на сайт конкурента, тем более что физически никуда идти или ехать не надо, а надо лишь набрать "даблюдаблюдаблю точка адрес конкурента точка ком ".

Тестирование скорости и надежности — это отдельная техническая дисциплина, за хорошее знание которой получают очень большие деньги в иностранной валюте.

Как правило, целью такого тестирования является обнаружение слабого места *(bottleneck)* в системе. Под системой подразумеваются все компоненты веб-сайта, включая код, базу данных, "железо" и т.д.

В моей практике был случай, когда из-за того, что один из запросов к базе данных был составлен громоздко (с точки зрения обработки этого запроса системой), одна интернет-компания потеряла много пользователей, так как в течение нескольких дней сайт то работал, то не работал, и никто не мог понять, what the heck is going on ("что за фигня "), пока один из программистов не встрепенулся и не исправил код. Прошу заметить, что функционально старый код работал прекрасно, но с точки зрения перформанса он никуда не годился.

Скорость и надежность веб-сайта профессионально проверяется специальным ПО, которое легко может стоить под 100 тыс. долл. (например, *Silk Performer* от *Segue* или *Load Runner* от *Mercury Interactive).*

Упомянутое ПО служит:

с одной стороны, для генерации наплыва пользователей,

с другой — для измерения скорости, с какой веб-сайт в среднем отвечает каждому из "наплывших" и *с третьей* — для последующего анализа полученных данных.

ТЕСТИРОВАНИЕ БЕЗОПАСНОСТИ *(security testing)*

Одна из знакомых моего друга несколько лет назад наотрез отказывалась пользоваться Интернетом. На вопрос "почему? " она неизменно отвечала, что боится хукеров, чем неизменно вызывала у окружающих смех до икоты, так как на самом деле она имела в виду хакеров (hacker — в современном значении киберпреступник, hooker — девушка легкого поведения).

Шутки шутками, а киберпреступность *(cyber crime)* — это целая криминальная индустрия, доходы ежегодно измеряются миллиардами долларов, которые соответственно теряют корпорации и честные каптруженики.

Тестирование безопасности — это множество вещей, суть которых заключается в том, чтобы усложнить условия для кражи — кражи данных, денег и информации.

> ***Например,*** *в одной из систем интернет-платежей есть специальный отдел, который профессионально занимается взламыванием... своего же веб-сайта и получает премии за каждую найденную ошибку в системе обеспечения безопасности.*

ТЕСТИРОВАНИЕ ОПЫТА ПОЛЬЗОВАТЕЛЯ *(usability testing)*

Призвано объективно оценить опыт пользователя *(user experience),* который будет работать с разрабатываемым интерфейсом.

Каждый из нас иногда ломает голову над тем, как исполнить желаемое на том или ином сайте. Поясню.

Допустим, вы идете на сайт сети пиццерий и хотите найти пиццерию, ближайшую к вашему дому. Если интерфейс сделан с заботой об опыте пользователя (user friendly interface), то мы быстро найдем вверху (header) и/или внизу (footer) страницы хорошо заметный линк "restaurant locator" либо "store locator" (месторасположение ресторана).

Вопрос: почему такой линк должен быть вверху или внизу страницы и называться именно так?

Ответ: да потому, что это своего рода конвенция, и пользователь, ищущий ближайшую к дому пиццерию, ожидает увидеть линк в этих местах и с таким названием.

При юзабилити-тестировании также проверяется интуитивность интерфейса. Я видел некоторые "гениальные" интерфейсы, которые словно были созданы с целью не допустить достижения страницы, на которой можно оплатить товар.

Еще одним примером ужасного юзабилити является ставшее популярным размещение, скажем, на новостных сайтах больших мигающих баннеров справа от текста, после минуты чтения новостей на таком сайте создается впечатление, что побывал на угарной дискотеке.

Юзабилити-тестирование часто проводится путем привлечения группы потенциальных пользователей с целью собрать впечатления от работы с системой.

В добрые доткомовские времена, на рубеже тысячелетий представители интернет-компаний запросто ловили на улицах Сан-Франциско праздношатающихся разгшъдяев и платили им 50 долл. за час работы со свежеиспеченным веб-сайтом. Those were the Ways, my friend... Those were the days... (непереводимо).

Зачастую опыт пользователя тестируется самими продюсерами во время написания спека и создания макетов. Есть также профессиональные юзабилити-инженеры.

ТЕСТИРОВАНИЕ СОВМЕСТИМОСТИ *(compatibility testing)*

Это проверка того, как наш веб-сайт взаимодействует с

- "железом" (например, модемами) и
- ПО (браузерами/операционными системами) наших пользователей.

Пример

Много лет назад, когда Netscape Navigator все еще использовался, а Виндоуз была еще в 98 версии, мы нашли такой баг:

*"Краткое **описание:***

> *"Проблема совместимости: Win'98 перезагружается при входе в систему с Netscape Navigator версии X.X"*

Описание и шаги *для воспроизведения проблемы:*

1. *Открой www.main.testshop.rs с помощью Netscape Navigator версии X.X, установленной на Win'98 (можно использовать машину из тест-лаборатории).*
2. *Введи "rsavin-testuser11@testshop.rs" в поле "Имя пользователя" и "121212" в поле "Пароль".*
3. *Нажми на кнопку "Вход".*

Баг: *Win'98 начинает перезагружаться.*
Ожидаемый результат: *вход в систему.*

Комментарий:
баг воспроизводится только при таком сочетании браузера и ОС".

Из примера почерпнем по крайней мере три вещи:

- при тестировании было найдено такое сочетание браузера/операционной системы, при котором существовал фатальный баг, из-за которого пользователь не только не смог бы войти в *www.main.testshop.rs,* но и терял бы всю свою несохраненную работу;
- проблемы, связанные с совместимостью между веб-сайтом и браузером/ОС, реальны и могут вести к серьезным багам;
- можно (и нужно) создать тест-лабораторию с наиболее **популярными** сочетаниями браузер/ОС, установленными на компьютерах наших пользователей.

Как найти эти популярные сочетания? Очень просто — покопайтесь в Интернете и поищите статистику о пользовании браузеров и ОС.

Что дальше? Дальше включаем компы с популярными ОС, запускаем на них популярные браузеры и исполняем наши тест-кейсы.

Тестирование с разными браузерами называется **кросс-браузер-тестированием** *(cross-browser testing).*

Тестирование с разными ОС называется **кросс-платформ-тестированием** *(cross-platform testing).*

Примером *тестирования совместимости вашего сайта и "железа" является ситуация, когда полноценное пользование вашим сайтом возможно только при наличии видеокарты определенного типа, например поддерживающей технологию DirectX версии X.X. Здесь мы можем, например, протестировать, каков будет опыт пользователя, если у того на машине установлена устаревшая и неподдерживаемая видеокарта (кстати, такое тестирование будет называться* ***негативным,*** *но об этом позднее).*

За исключением тех случаев, когда тест-кейсы специально созданы для тестирования совместимости, я не рекомендую указывать

в них детали, например, по типу и версии браузера, так как типы и особенно версии меняются. Как мы помним, излишняя детализация приводит к трате времени на поддержание тест-кейсов.

3. По субъекту тестирования

- альфа-тестировщик *(alpha tester);*
- бета-тестировщик *(beta tester).*

АЛЬФА-ТЕСТИРОВЩИК *(alpha tester)*

Это сотрудники компании, которые профессионально или непрофессионально проводят тестирование: тестировщики, программисты, продюсеры, бухгалтеры, сисадмины, секретарши. В стартапах накануне релиза нередко все работники, включая Харитоныча, сидят по 16 часов кряду, пытаясь найти непойманные баги.

БЕТА-ТЕСТИРОВЩИК *(beta tester)*

Это нередко баловень судьбы, который не является сотрудником компании и которому посчастливилось пользоваться новой системой до того, как она станет доступна всем остальным. За бета-тестирование иногда даже платят деньги (вспомните пример с 50 долл. в час за юзабилити-тестирование).

4. По времени проведения тестирования

ДО передачи пользователю — альфа-тестирование *(alpha testing):*

- тест приемки *(smoke test, sanity test* или *confidence test);*
- тестирования новых функциональностей *(new feature testing);*
- регрессивное тестирование *(regression testing);*
- тест сдачи *(acceptance* или *certification test),*

ПОСЛЕ передачи пользователю — бета-тестирование *(beta testing)*

О "До передачи пользователю — альфа-тестирование *(alpha testing)"* мы еще поговорим.

О "После передачи пользователю — бета-тестирование *(beta testing)"* уже говорили.

5. По критерию "позитивности" сценариев

- **позитивное** тестирование *(positive testing);*
- **негативное** тестирование *(negative testing).*

Начнем со второго.

Пример

Допустим, что имя файла с банковскими транзакциями должно иметь определенный формат:

bofa_< YYYYMMDD>_ach. txt,

где YYYY — это год в полном формате (2005), MM — это месяц в полном формате (01 — январь), DD — это день в полном формате (01 — первое число месяца).

Этот файл служит в качестве ввода для программы process transactions, которая ежедневно в 23:00

автоматически "забирает" его из директории /tmp/input_files/,
анализирует (parse) его и
вставляет данные из него в базу данных.

Предположим, что из-за ошибки кода, генерирующего файл, имя файла от 18 января 2004 г. будет не

- *bofa_20040t18_ach.txt (processtransactions ожидает именно и буквально это имя), а*
- *bofa_2004118_ach.txt.*

Какая реакция должна быть у программы process_transactions, если она не может найти файл?

Ответ на этот вопрос может быть найден в спеке, где, например, может быть указано, что в ситуации, когда файл не найден, process_ transactions посылает соответствующему дистрибутивному списку е-мейл:

- *с предметом (e-mail subject)* ***"Ошибка: файл ввода для process transactions отсутствует"*** *и*
- *содержанием (e-mail body)* ***"Файл bofa_20040118_ach.txt отсутствует в директории /tmp/input_files/".***

Если спек не предусматривает возможности возникновения такой ситуации, то мы как тестировщики должны ее предусмотреть и создать тест-кейс с соответствующим сценарием.

Итак, **сценарий, проверяющий ситуацию, связанную с**

- **потенциальной ошибкой** *(error)* пользователя и/или
- **потенциальным дефектом** *(failure)* в системе,

называется негативным.

***Пример** ошибки пользователя*
Ввод недействительных данных в поле "Имя" на странице регистрации.

***Пример** дефекта в системе*
Вышеуказанный пример о неправильной генерации имени файла.

Создание и исполнение тест-кейсов с негативными сценариями называется НЕГАТИВНЫМ тестированием.

Далее.

Позитивные сценарии — это сценарии, предполагающие нормальное, "правильное"

- **использование** и/или
- **работу системы.**

*Первый **пример** позитивного сценария*
*Ввод **действительных** данных в поле "Имя" на странице регистрации.*

*Второй **пример** позитивного сценария*
*Проверка работы системы, когда имя файла имеет **правильный** формат: bofa_20040l 18_ach.txt.*

Создание и исполнение тест-кейсов с **позитивными сценариями называется ПОЗИТИВНЫМ тестированием.**

Несколько мыслей вдогонку:

а. Как правило, **негативное тестирование находит больше багов.**
б. Как правило, негативное тестирование — вещь более слож ная, творческая и трудоемкая, так как спеки описывают, как должно работать, когда "усе в порядке", а не как долж но работать в ситуациях с ошибками или сбоями.
в. Если есть позитивные и негативные тесты как часть тест-комплекта, то **позитивные тесты исполняются в первую очередь.** Логика:
В большинстве случаев целью создания функционачьности является возможность реализации именно позитивных сценариев, т.е. работоспособность позитивных сценариев более приоритетна, чем работоспособность негативных сценариев.
г. Существуют спеки, полностью посвященные тому, как долж на себя вести система при ошибках/дефектах. Следователь но, все тестирование такого спека будет негативным.

д. Два полезных термина:

- **обращение с ошибкой/дефектом** *(error handling /failure handling)* — это то, как система реагирует на ошибку/дефект;
- **сообщение об ошибке** *(error message)* — это информация (как правило, текстовая), которая выдается пользователю в случае ошибки/сбоя.

*Маленький **примерчик** вдогонку*

Правильность сообщений об ошибке является намного более серьезной вещью, чем может показаться, при рассуждениях об этом в теории. Например, сегодня я попытался купить по Интернету новую книгу Харуки Мураками:

- *добавил книгу в корзину на одном из сайтов,*
- *вбил номер кредитки в соответствующие поля веб-страницы и*
- *нажал кнопку "Купить".*

Мне выдается сообщение об ошибке: так, мол, и так, проверьте, пожалуйста, номер своей кредитной карты, дорогой пользователь. Я проверяю — все в порядке: и номер карты, и срок действия. Нажимаю "Купить" еще раз — го же сообщение об ошибке. Пробую вбить информацию по другой карте — то же самое. Начиная с этого момента, успешное осуществление акции покупки новой книги Харуки Мураками стало для меня делом принципа. Звоню в службу поддержки, и мне говорят

— А вы, кстати, поставили галочку в чек-бокс (check box), что согласны
с нашим соглашением?
-Нет.
—А вы поставьте и попробуйте нажать на кнопку "Купить".
—Ставлю, пробую, работает.
—Ну вот и славненько. Чем-нибудь еще можем быть полезны?
—Ничем. Thank you.

В итоге я потерял 15 минут своего времени, а веб-сайт потерял меня как пользователя, так как "ложечки нашлись, а осадок остался". Все из-за неверного сообщения об ошибке.

6. По степени изолированности тестируемых компонентов

- **компонентное** тестирование *(component testing);*
- **интеграционное** тестирование *(integration testing);*
- **системное** (или энд-ту-энд) тестирование *(system or end-to-end testing).*

Сначала краткие и емкие определения, а затем иллюстрации.

Компонентное тестирование *(component testing)* — это тестирование на уровне логического компонента. И это **тестирование самого логического компонента.**

Интеграционное тестирование *(integration testing)* — это тестирование на уровне двух или больше компонентов. И это тестирование **взаимодействия** этих двух или больше компонентов.

Системное (или энд-ту-энд) тестирование *(system or end-to-end testing)* — это проверка всей системы от начала до конца.

Теперь иллюстрации кратких и емких определений.

Допустим, программисту поставлена задача написать код, который бы находил **полные имена и е-мейлы пользователей,** потративших больше 1000 долл. в нашем онлайн-магазине с момента регистрации. Таким пользователям должен быть отправлен **е-мейл с подарочным сертификатом,** использование которого до 17 ноября включительно предоставит 5%-ю скидку на любую разовую покупку.

> ***Кстати,*** *для добротного тестирования данной функциональности нужно написать гораздо больше тест-кейсов, чем я приведу, но сейчас наша задача — это понять*
>
> *суть каждого из трех рассматриваемых видов тестирования и разницу между ними.*

КОМПОНЕНТНОЕ ТЕСТИРОВАНИЕ

Для начала выделим три компонента, которые мы протестировали бы:

1. Создание файла с полными именами, е-мейлами и номерами сертификатов.
2. Рассылка пользователям е-мейлов.
3. Правильное предоставление скидки вышеуказанным пользователям.

Проверяем.

Компонент 1

Проверяем, что создается файл нужного формата

- с полными именами и е-мейлами пользователей, потративших > 1000 долл., и
- номером сертификата для каждого из этих пользователей.

Это позитивное тестирование.

Мы также должны проверить, не затесались ли в наш файл

пользователи, потратившие < 1000 долл.

Это негативное тестирование, связанное с потенциальным дефектом в коде, отвечающем за выбор правильных пользователей.

Компонент 2

Допустим, код первого компонента, который должен был создать для нас файл, не работает. Мы не отчаиваемся, а просто вручную, не ропща на судьбу, создаем файл установленного формата с взятыми с потолка

- е-мейлами,
- полными именами пользователей и
- номерами подарочных сертификатов.

Этот файл мы "скармливаем" программе рассылки е-мейлов и проверяем, что правильные е-мейлы доходят до пользователей из файла(позитивное тестирование).

Компонент 3

Как мы помним, компонент 1 не работает. Что делать?

Сертификат — это как некий код, например *"UYTU764587657"*. который нужно ввести во время оплаты, и если сертификат действительный, то итоговая сумма к оплате уменьшается.

В данном случае можно попросить программиста, чтобы тот помог сгенерировать легитимные номера сертификатов. Когда номера сертификатов имеются в наличии, можно, например, проверить, работает ли подарочный сертификат только один раз (позитивное тестирование) или его можно использовать для двух или более транзакций (негативное тестирование, воспроизводящее ошибку пользователя, использующего сертификат более одного раза). Также нужно будет проверить размер скидки (5%) (позитивное тестирование) и действительность сертификата:

до 17 ноября (позитивное тестирование), 17
ноября (позитивное тестирование) и
после 17 ноября (негативное тестирование, воспроизводящее ошибку пользователя, использующего просроченный сертификат).

***Кстати,** в случаях когда тестирование связано со сроками (например, сроком истечения сертификата), мы, естественно, не ждем до 17 ноября, а просто меняем системное время тест-машины на нужное время или меняем значение времени в базе данных. Естественно, что такие изменения вы должны предварительно согласовать с коллегами, которые работают на той же тест-машине или с той же базой данных.*

Важный момент: хотя по спеку все три компонента и взаимосвязаны, из-за проблем в коде у нас получилось **компонентное тестирование** в чистом виде. Другими словами, мы тестировали сами компоненты, а не **связь** между ними.

Тестирование связи между компонентами называется **интеграционным тестированием.**

ИНТЕГРАЦИОННОЕ ТЕСТИРОВАНИЕ

У нас есть три связи между компонентами:

а) между 1-м и 2-м компонентами;
б) между 2-м и 3-м компонентами;
в) между 1-м и 3-м компонентами.

Подробности:

а. Компонент 1 генерирует файл со списком
   - е-мейлов и полных имен подходящих пользователей и
   - номерами сертификатов.

   Этот список используется компонентом 2, который ответствен за рассылку е-мейлов.

б. Компонент 2 доставляет пользователю в качестве е-мейла информацию о подарочном сертификате. Пользователь может использовать сертификат (компонент 3), только ес ли он знает правильный номер своего сертификата.

в. Компонент 1 генерирует код сертификата, который ис пользуется компонентом 3.

Итак, в нашем случае при интеграционном тестировании у нас есть для проверки 3 связи. Приведем примеры соответствующих тестов на интеграцию.

а. Здесь можно проверить, совместим ли формат файла, созданного компонентом 1, с программой рассылки компонента 2. Например, последняя принимает следующий формат файла:

полное имя пользователя, е-мейл, номер сертификата.

Значения отделены друг от друга запятой *(comma-delimited)*. Информация о каждом новом пользователе — на новой строчке. Сам файл — простой текстовый файл, который можно открыть программой *Notepad.*

Образец файла:

Ferdinando Magellano, f.magellano@trinidad.pt, QWERT98362
James Cook, james.cook@endeavour.co.uk, ASDFG54209 Иван Крузенштерн, ikruzenstern@nadejda.ru, LKJHG61123

Допустим, программист ошибочно заложил в коде, что значения файла разделяются **не запятой** (форматом, принимаемым программой рассылки), **а точкой с запятой:**

Ferdinando Magellano; f.magellano@trinidad.pt; QWERT98362
James Cook; james.cook@endeavour.co.uk; ASDFG54209 Иван Крузенштерн; ikruzenstern@nadejda.ru; LKJHG61123

Когда мы проводим интеграционный тест, мы обнаруживаем, что программа рассылки не принимает файл неподходящего формата, и соответственно никакие е-мейлы до пользователей не дойдут, если этот баг не будет устранен.

б. В данном случае у нас может быть ситуация, когда файл имеет верный номер сертификата, но из-за бага в программе рас сылки пользователь получает е-мейл с "неправильным" номером сертификата.

Это может произойти из-за того, что программа рассылки может быть ошибочно сконфигурирована, чтобы "брать" только 9 первых символов из третьей колонки (колонки с номерами сертификатов), т.е. QWERT98362 будет преподнесена пользователю в укороченном виде (truncated): QWERT9836.

Интеграционный тест по использованию номера сертификата, полученного по е-мейлу, может выявить этот баг.

в. Здесь может быть ситуация, когда номер сертификата, сгене рированный компонентом 1, не принимается компонентом 3.

***Пример** такой ситуации*

Компонент 1 сохранил номер сертификата в базе данных в зашифрованном виде, т.е. в целях безопасности использовался алгоритм, который превратил "LKJHG61123", например, в '"&"(*&86%(987$!$#". Из-за бага в компоненте 3 последний не дешифровал номер сертификата,*

взятый из БД, а просто попытался сравнить эту абракадабру из БД и номер сертификата, введенный пользователем, что привело к тому, что номера не сошлись и легитимная скидка не была предоставлена.

Должен ли был быть номер сертификата зашифрован или нет, для нас сейчас значения не имеет. Значение имеет тот факт, что баг был обнаружен во время **интеграционного тестирования.**

СИСТЕМНОЕ ТЕСТИРОВАНИЕ

Это тестирование системы (функциональности) **от начала до конца** *(end-to-end),* т.е. каждый сценарий будет затрагивать всю цепочку: компонент 1 —> компонент 2 —> компонент 3.

Я рекомендую ставить простой тест-кейс с системным тестом в самое начало тест-комплекта. Так можно сразу увидеть, если что-то явно не в порядке. Это своего рода тест приемки непосредственно для вещи, тестируемой данным тест-комплектом.

Хорошая идея *вдогонку*

Е-мейл состоит из следующих частей:

е-мейла алиаса;
собаки;
домена почтового сервера;
точки;
глобального домена.

В вашем рабочем е-мейле алиасом будет, как правило, ваши имя (или инициал) и фамилия: ***rsavin.***

Собака остается собакой, хотя по-аглицки она называется "at" (читается как "эт"):

@ Доменом почтового сервера будет домен

компании:

testshop

Точка остается точкой, хотя по-аглицки она называется "dot" (читается как "дот"):

.

Глобальный домен — это зона домена компании, например "com" или "ги":

rs,

т.е. получаем: rsavin@testshop.rs

При тестировании интернет-проектов приходится создавать много счетов пользователей. Загвоздка в том, что е-мейл пользователя, который очень часто является его именем, может быть использован только один раз, т.е. мой рабочий е-мейл rsavin@testshop.rs может быть использован для создания только одного счета.

Что делать? Открывать бесчисленные счета на хотмейлах и яху? Ответ неверный.

Самая хорошая идея: *поговорите с администратором почтового сервера вашей компании, чтобы он модифицировал настройки сервера так, чтобы к вам приходили все е-мейлы следующего формата:*

rsavin+sometext@testshop. rs,

т. е. после моего алиаса стоит ***знак плюс*** *и между знаком плюс и собакой* ***находятся любые легитимные знаки.***

Например, для тестирования компонента 1 я регистрируюсь с е-мейлом:

rsavin+component1_test@testshop.rs

Таким образом, вы можете создавать тысячи эккаунтов пользователей своего сайта, не регистрируя тысяч новых е-мейл-эккаунтов.

Рекомендую. Очень удобно.

7. По степени автоматизированное™ тестирования

- **ручное** тестирование *(manual testing);*
- **автоматизированное** тестирование *(automated testing);*
- **смешанное / полуавтоматизированное** тестирование *(semi automated testing).*

О каждом из трех "друзей" будет еще **сказано очень много и** в **подробностях.** Пока же давайте поговорим концептуально.

РУЧНОЕ ТЕСТИРОВАНИЕ

Это исполнение тест-кейсов без помощи каких-либо программ, автоматизирующих вашу работу. Например, для того чтобы создать эккаунт нового пользователя, мы идем на наш *www.main.testshop.rs,* открываем страницу регистрации, заполняем формы и т.д.

АВТОМАТИЗИРОВАННОЕ ТЕСТИРОВАНИЕ

Это отдельная дисциплина искусства тестирования. Значительная часть эффективности работы отдела тестирования зависит от того, какие задачи отданы для автоматизации и как эта автоматизация была осуществлена. Автоматизация может как принести огромное облегчение всем тестировщикам, так и завалить работу всего отдела и отложить релиз, премию, отпуск и другие сладкие вещи.

Оговорка

Термин "тул" (tool (англ.) — инструмент) используется для обозначения компьютерной программы, как правило, вспомогательного свойства.

Автоматизировать можно сотни вещей. Вот наиболее часто встречающиеся виды автоматизации:

а. Тулы для помощи в черноящичном и сероящичном тестировании.

Например,

- тул, который автоматически создает для нас эккаунт пользователя;
- тул, совершающий запросы к базе данных и генерирующий файлы формата, утвержденного системой *VISA,* используя извлеченные данные;
- тул, генерирующий транзакции покупки в нашем магазине, и т.д. и т.п.

Вариантам нет конца и края. Такие тулы пишутся программистами компании или самими тестировщиками.

***Пример** тула, создающего эккаунты пользователя*

Если набрать в браузере www.main.testshop.rs/tools/register.py (это все, естественно, гипотетически, так как такого сайта в природе не существует), то мы увидим не 10 обязательных полей, которые нужно заполнить, а одно текстовое поле и кнопку "создать тест-эккаунт". Вы просто вводите уникальный е-мейл нового пользователя, например rsavin-testuser1000@testshop.rs, и нажимаете на кнопку. Тул делает за вас все остальное. Пароль для всех эккаунтов будет, например "898989".

Хорошая идея:

*используется автоматизация для создания новых эккаунтов или нет, очень удобно, когда в компании существует **конвенция для одного пароля** при создании тест-эккаунтов, например "898989".*

Дело в том, что иногда нет времени/возможности создать эккаунт с определенными транзакциями, настройками и т.д., и если такой эккаунт уже существует, то, зная пароль, вы сможете им воспользоваться.

При этом помните о деловой этике, и если этот эккаунт создан не вами, то по возможности вежливо спросите у "хозяина" эккаунта разрешение.

б. Программы для регрессивного тестирования

Это специальное ПО, созданное для буквального воспроизведения действий тестировщика.

Пример

Согласно тест-кейсу вы должны

- *войти в систему,*
- *выбрать товар,*
- *положить его в корзину,*
- *заплатить и*
- *удостовериться, что баланс на кредитной карте уменьшился на сумму покупки.*

Чтобы исполнить этот тест-кейс, вы должны запустить браузер, ввести имя пользователя и пароль, нажать на кнопку "Вход"... и, в конце концов, сравнить фактический и ожидаемый результаты.

Теперь представьте себе, что некая программа делает те же самые действия, что и вы, т.е. сама запускает браузер, печатает, где положено, имя пользователя и пароль, нажимает на кнопку "Вход"... и, в конце концов, сравнивает ожидаемый и фактический результат и сообщает вам о нем (через сообщение на экране, запись в файле, е-мейл и т.д.).

Такое ПО, как правило, поддерживает режим "Запись / Воспроизведение", т.е. когда мы нажимаем на кнопку "Запись" и начинаем кликать мышками и клацать клавишами клавиатуры, ПО записывает наши действия и, когда мы закончили, генерирует код. Этот код мы можем запустить с этим же ПО, и оно воспроизведет все наши клики и клацы, т.е. буквально будет водить курсором мышки, набирать текст и т.д.

Такое ПО, как правило, имеет собственный язык программирования, т.е. можно не записывать свои действия, а непосредственно написать код, что и делается теми, кто профессионально работает с таким ПО.

Наиболее популярная и мощная программа для автоматизации регрессивного тестирования веб-проектов — это *Silk Test,* выпускаемый компанией *Segue.*

У нас будет отдельная беседа о хороших и плохих вещах, связанных с автоматизацией регрессивного тестирования.

в. Программы для тестирования скорости и надежности

О таком ПО мы уже говорили. И так как *stress/load/performance testing* — это песня не нашего черно-сероящичного репертуара, петь, т.е. говорить, о них больше не будем.

г. Прочие программы

Это, например, "Проверяльщики линков" *(link checkers).*

СМЕШАННОЕ/ПОЛУАВТОМАТИЗИРОВАННОЕ ТЕСТИРОВАНИЕ

Здесь ручной подход сочетается с автоматизированным. Например, с помощью тула я создаю новый эккаунт и потом вручную генерирую транзакцию покупки.

8. По степени подготовки к тестированию

- тестирование по тест-кейсам *(documented testing);*
- интуитивное тестирование *(ad hoc testing).*

Здесь все просто. Есть тестирование по тест-кейсам, а есть тестирование *ad hoc (лат. —* для этой цели, читается как "эд-хок"), т.е. мы просто интуитивно роемся в ПО, пытаясь найти баги. Интуитивное тестирование, как правило, применятся:

- тестировщиком в качестве теста приемки и/или теста сдачи (если тест-кейсы для них не формализованы в документации);
- тестировщиком в качестве успокаивающего для сердца в довесок к документированным тестированию новых функциональностей и регрессивному тестированию;
- тестировщиком, который только что пришел в компанию, где код уже написан и нужно срочно все протестировать;
- когда бухгалтерия и менеджмент протягивают тестировщикам руку помощи перед релизом;
- в других случаях, когда нет тест-кейсов.

Нужно отметить, что эд **хок-тестирование часто дает поразительные результаты:** бывает, исполняешь только что пришедшие в голову сценарии, которые и не снились при подготовке к тестированию, и находишь дородные, розовощекие и ухмыляющиеся баги.

Краткое подведение итогов

1. Мы классифицировали основные виды тестирования в интернет-компаниях.
2. Мы узнали о трех основных подходах к тестированию: "Черный ящик", "Белый ящик" и "Серый ящик". Водораздел между ними лежит в плоскостях степени знания о внутренностях системы и ориентированности на надежды и чаяния конечного пользователя.
3. Мы узнали, что паттерн поведения пользователя составляют сценарии и данные для них (хотя мы стали все это вместе называть сценариями).

4. Мы узнали об основных источниках знания о потенциальных паттернах поведения пользователей.
5. Мы узнали концепцию тестировочного покрытия.
6. Мы узнали, что количественное и качественное тестирование обеспечивается путем слияния в оргазме черноящичных и белоящичных методик тестирования.
7. Мы узнали, что мало быть хорошим человеком. Надо еще понимать, какой ожидаемый вывод является тем самым ожидаемым результатом, который приведет нас к реальному тестированию.
8. Мы поняли разницу между тестированием **интерфейса** пользователя и тестированием **с помощью интерфейса** пользователя.
9. Мы удивились, узнав, что код, прекрасно работающий функционально, может привести к сбою в работе веб-сайта (проблемы перформанса).
10. Мы прочувствовали, что несовместимость — это проблема не только человеческих отношений, но и отношений нашего сайта с "железом" и ПО пользователя.
11. Мы запомнили, что, как правило, позитивные тесты исполняются в первую очередь.
12. Мы прошли шаг за шагом от компонентного до системного тестирования.
13. Мы разобрались в видах автоматизации.
14. Мы отметили, что интуитивное (эд хок) тестирование иногда приносит превосходные результаты.

Задание для самопроверки

Приведите, пожалуйста, классификацию видов тестирования с определением каждого из них.