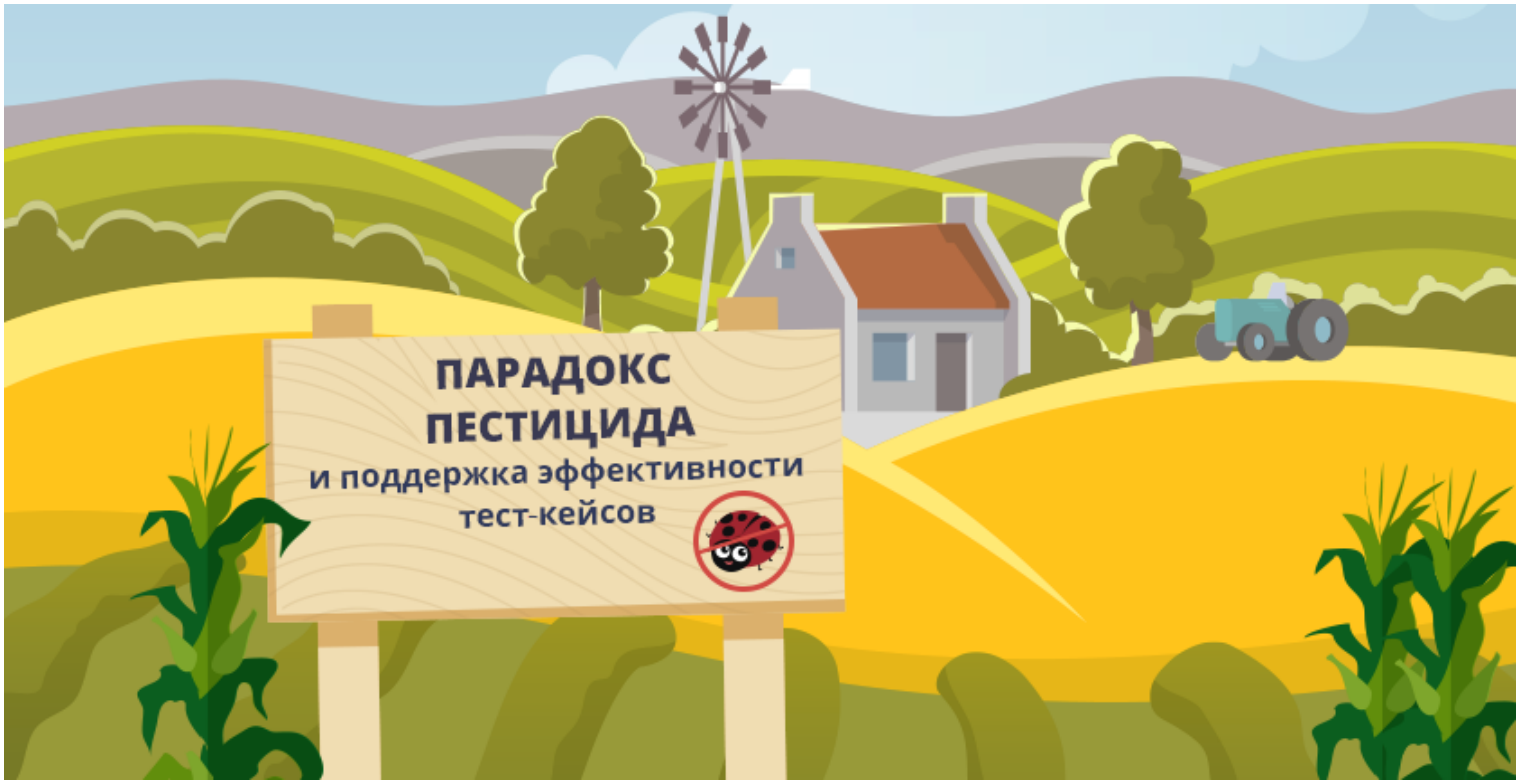
ПАРАДОКС ПЕСТИЦИДА И ПОДДЕРЖКА ЭФФЕКТИВНОСТИ ТЕСТ-КЕЙСОВ

14.03.2019



Что такое парадокс пестицида?

На большинстве проектов случаются такие ситуации, когда чем больше мы тестируем продукт, тем больший иммунитет вырабатывается у багов, которые мы пытаемся найти, используя наши тестовые наборы. Когда одни и те же тесты повторяются снова и снова, то в конце концов они перестают находить новые баги. В такой ситуации, даже детально описанный тест-кейсами, функционал может быть не протестирован достаточно тщательно и к пользователям могут попасть серьезные баги.

Так же, как и у насекомых, у багов в тестируемом ПО может развиться сопротивление к одному и тому же пестициду, т.е. сформироваться иммунитет к уже существующим тест-кейсам. Такой феномен и называется **эффектом пестицида**. Этот термин около 30 лет назад придумал американский

инженер Борис Бейзер. Суть данного эффекта состоит в том, что многократное применение одинаковых методов со временем становится неэффективным, так как выжившие вредители приобретают иммунитет.

Почему со временем тест-кейсы перестают быть актуальными?

Конечно же, самыми простыми причинами могут быть ситуации, когда на проекте вносятся изменения, добавляется новый функционал или же оптимизируется старый. Обновление тест-кейсов в таком случае необходимо и без учета эффекта пестицида. Но даже в случае, когда проект сильно не изменяется, существует надобность обновлять набор тестов для поиска все новых и новых багов, которые, без сомнения, есть в любом продукте.

Существует несколько причин, почему один и тот же набор тест-кейсов перестает приносить хороший улов багов.

Во-первых, просто невозможно изначально предусмотреть все доступные сценарии для тестирования. Даже если тестируемое ПО достаточно простое, написать исчерпывающее количество тестов, которые найдут все баги, увы, нереально. Не говоря уже о сложных системах с огромным набором входящих данных, зависимостей и сценариев развития. Поэтому любые новые идеи о том, как можно протестировать продукт должны быть проверены и добавлены в арсенал тестировщика.

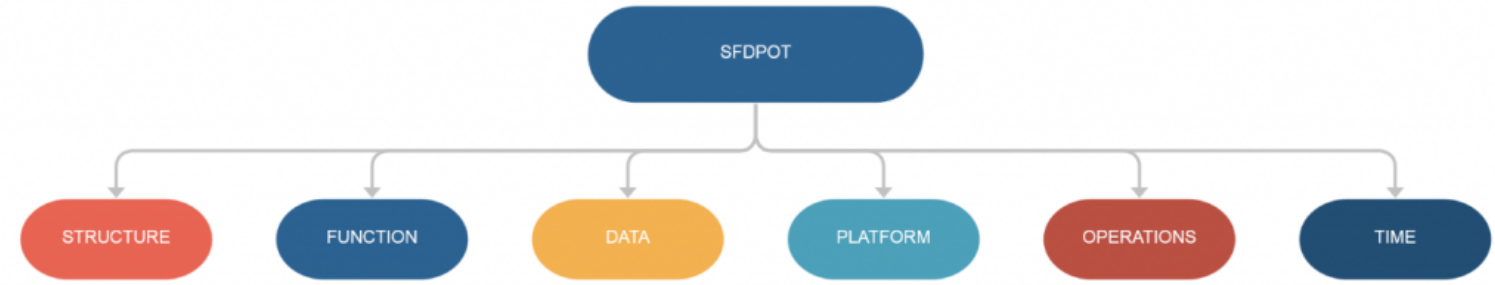
Во-вторых, со временем тестировщик привыкает к проекту, ему становятся знакомыми все проблемные места тестируемого ПО и есть риск, что он может пропускать дефекты или даже посчитать увиденный дефект за оговоренную ранее особенность функционала. Когда тестировщик работает на проекте полгода или больше, ему необходимо раз за разом выполнять регрессионное тестирование и проходить один и тот же набор тестов, то сохранить свежий взгляд на продукт – нелегкая задача.

Как поддерживать тест-кейсы?

Очень важно при тестировании использовать набор разных подходов и техник, чтобы посмотреть на ПО под разными углами, с точки зрения структуры проекта, его функций, обрабатываемых им данных. Важно также понимать, как пользователи будут взаимодействовать с ПО, какие возможные проблемы у них могут возникнуть. Ведь можно написать сотни тест-кейсов и все равно пропустить важный баг. Существует множество техник, которые способствуют тестировщику в тщательном исследовании ПО с разных сторон.

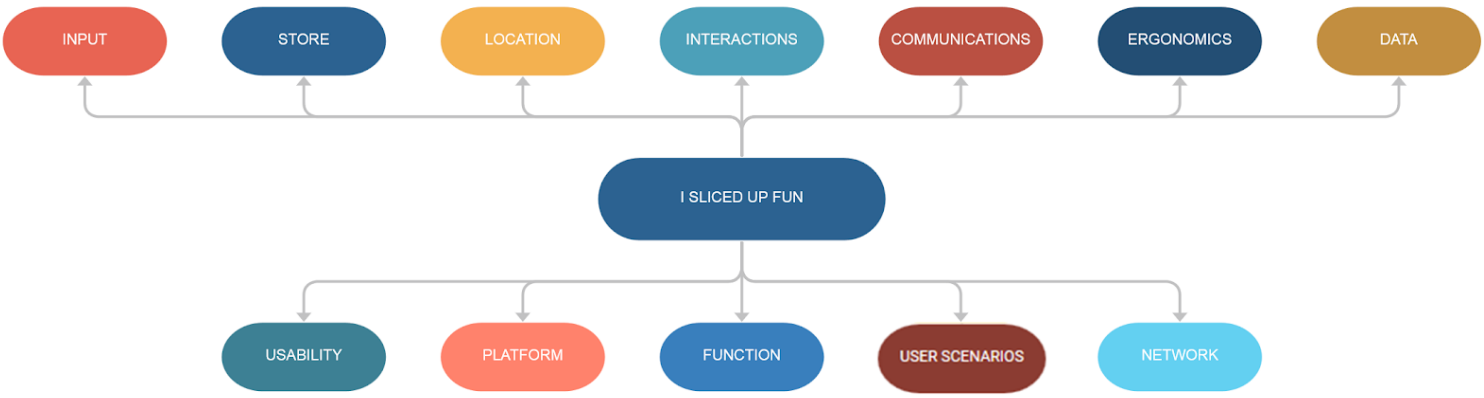
Например, есть ряд мнемоник, которые помогают тестировщику расширить количество и эффективность проектируемых тестов. Одна из популярных мнемоник – **SFDPOT** (San Francisco Depot), придуманная Джеймсом Бахом. Каждая буква в SFDPOT представляет аспект тестируемого продукта, который нужно проверить:

- **S**tructure – определяет из чего состоит ПО.
- **F**unction – что делает ПО, его функции.
- **D**ata – какие обрабатывает данные.
- **P**latform – на каких платформах поддерживается ПО.
- **O**perations – как ПО будет использоваться.
- **T**ime – как ПО поведет себя в зависимости от времени.



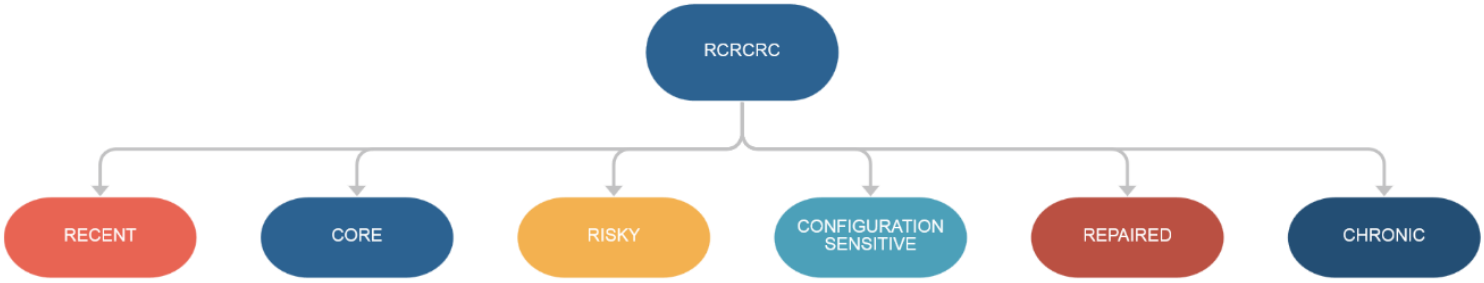
Еще одна мнемоника – **I SLICED UP FUN**, которая придумана Джонатаном Колом для тестирования мобильных приложений, напоминает о том, что необходимо проверить:

- **I**nput – возможные способы передачи информации приложению.
- **S**tore – требования магазинов приложений.
- **L**ocation – местоположение пользователя.
- **I**nteractions/interruptions – прерывания работы.
- **C**ommunication – связь.
- **E**rgonomics – эргономика.
- **D**ata – обрабатываемые данные.
- **U**sability – удобство использования.
- **P**latform – поддерживаемые платформы.
- **F**unction – функционал.
- **U**ser scenarios – сценарии использования.
- **N**etwork – сеть.

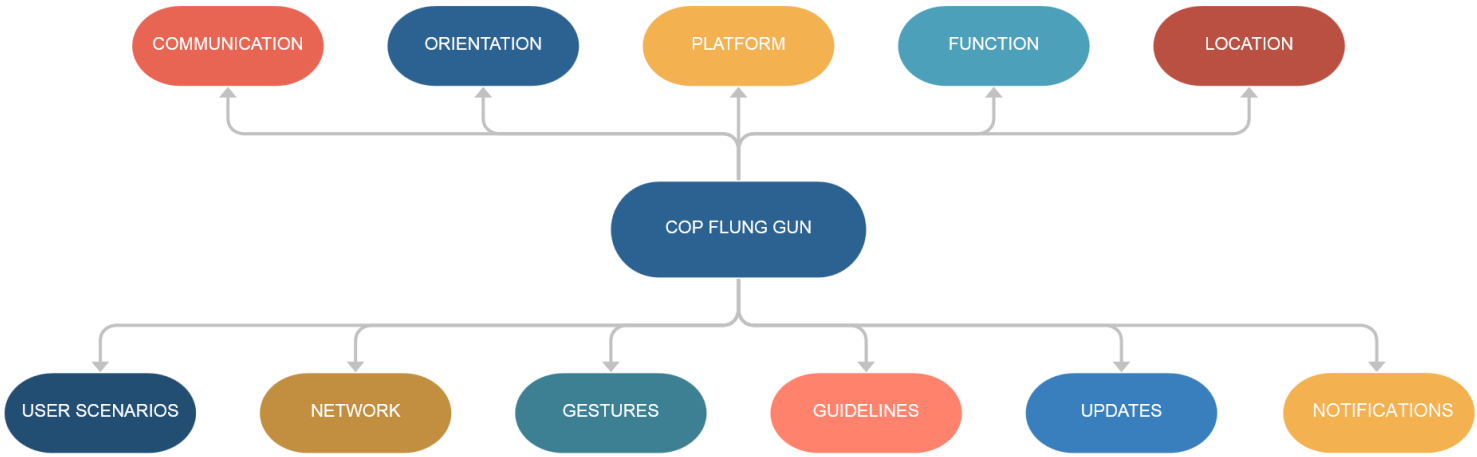


Существует множество других мнемоник, например:

RCRCRC (Recent – Core – Risky – Configuration sensitive – Repaired – Chronic) – для помощи при регрессионном тестировании.



COP FLUNG GUN (Communication – Orientation – Platform – Function – Location – User Scenarios – Network – Gestures – Guidelines – Updates – Notifications) – для тестирования мобильных приложений.



Благодаря взгляду на продукт с разных сторон, можно придумать много хороших тестов. Использование мнемоник помогает уменьшить риск того, что тестировщик забудет обратить внимание на важную часть ПО, дает возможность убрать хаотическое генерирование идей для тестов и выработать более структурированный подход к тестированию и/или проектированию тестов. Выполнив список проверок по мнемонике, тестировщик будет знать, что главные стороны продукта протестированы и ничего не упущено. Конечно же, мнемоники – это не исчерпывающий тест-план, но удобный помощник при генерации идей.

Для избежания ситуации, когда существующий тестовый набор не соотносится с текущим состоянием ПО и перестает быть актуальным, его необходимо пополнять проверками всякий раз, когда ПО изменяется и пополняется новым функционалом.

Следующие советы помогут поддерживать актуальность тест-кейсов:

1. **Редактируйте уже существующие тест-кейсы при изменении ПО.**
   При первом проектировании кейса учитывайте, что, возможно, в будущем вам придется его править. Поэтому старайтесь писать легко исправляемые тест-кейсы. Будущий вы или ваш коллега скажет вам большое спасибо за это.

2. **Перемещайте устаревшие и ненужные тест-кейсы из актуального тестового набора.**
   Если вы видите, что функционал, который проверяется тест-кейсом убрали – переносите этот тест в архив, где он будет спокойно отдыхать и ждать возможного возвращения своего функционала. Удалять тест-кейсы совсем не рекомендуется, так как всегда существует вероятность того, что функционал захотят вернуть по просьбе пользователей, по желанию менеджера или по любым другим причинам.

3. **Добавляйте новые тест-кейсы, необходимые для проверки нового функционала.**
   Каждый раз, когда команде тестирования поручается проверить новый функционал, стоит написать хотя бы базовый набор тестов, который позже можно будет расширить. Конечно, бывают ситуации, когда задачу закончить надо на вчера и продукт должен быть протестирован как можно скорее, но и в таком случае следует записать, какие сценарии вы собираетесь протестировать, хотя бы в виде названий тест-кейсов без заполнения всех его атрибутов. Это поможет вам более вдумчиво подойти к тестированию сейчас, а позже, когда пожар на проекте утихнет, вы сможете закончить тест-кейсы.

4. **Добавляйте тест-кейсы для проверки сценариев связанных с багами, которые были найдены командой тестирования или конечными пользователями.**
   Это позволит дополнить набор интересными кейсами, о которых не подумали ранее при проектировании тест-кейсов.

Пополняя свой набор тест-кейсов, тестировщик сможет находить новые интересные и важные баги, посмотреть на ПО с разных сторон и быть уверенным, что продукт будет выпущен действительно качественным. Как следствие, тестировщик не сможет заскучать на проекте, так как ему не придется прогонять все одни и те же тесты, которые будут приносить все меньше и меньше пользы.