# ТЕСТ ДИЗАЙН

Тест дизайн — один из первоначальных этапов тестирования программного обеспечения, этап планирования и проектирования тестов. Тест дизайн представляет собой продумывание и написание тестовых случаев (test case), в соответствии с требованиями проекта, критериями качества будущего продукта и финальными целями тестирования.

Тест дизайн

### **Цели тест дизайна**

1. Обеспечить покрытие функционала приложения тестами:
   1. Тесты должны покрывать весь функционал
   2. Тестов должно быть минимально достаточно

## **Тест дизайн задачи**

1. Проанализировать требования к продукту
2. Оценить риски возможные при использовании продукта
3. Написать достаточное минимальное количество тестов
4. Разграничить тесты на приемочные, критические, расширенные

Какие скиллы необходимо прокачивать тест дизайнеру для того чтобы разрабатывать тесты оптимальными, быстро и с минимальными ошибками. В большинстве небольших компаний разработка таких тестов доверяется непосредственно тестировщику, для этого нужно иметь знания и быть готовым к такой задаче.

### **Тест дизайн скиллы профессионала:**

1. Декомпозиция приложения — разбивание тестируемой системы на компоненты.
2. Навыки и способности поиска информации для приложения.
3. Расстановка приоритетов
4. Грамотная речь и верный вектор мыслительного процесса
5. Знание техник тест дизайна
6. Отточенное мастерство применения техник тест дизайна

## **Техники тест дизайна**

Техники тест дизайна это рекомендации, советы и правила по которым стоит разрабатывать тест для проведения тестирования приложения. Это не образцы тестов, а только рекомендации к применению. В частности различные инженеры могут работая под одним и тем же проектом создать различный набор тестов. Правильным будет считаться тот набор тестов, который за меньшее количество проверок обеспечит более полное покрытие тестами.

\* **Анализ Граничных Значений (Boundary Value Analysis — BVA)**. Если взять пример выше, в качестве значений для позитивного тестирования выберем минимальную и максимальную границы (1 и 10), и значения больше и меньше границ (0 и 11). Анализ Граничный значений может быть применен к полям, записям, файлам, или к любого рода сущностям имеющим ограничения.

## Техника анализа граничных значений

Это техника проверки поведения продукта на крайних (граничных) значениях входных данных. Граничное тестирование также может включать тесты, проверяющие поведение системы на входных данных, выходящих за допустимый диапазон значений. При этом система должна определённым (заранее оговоренным) способом обрабатывать такие ситуации. Например, с помощью исключительной ситуации или сообщения об ошибке.

!!!Граничные значения очень важны и их обязательно следует применять при написании тестов, т.к. именно в этом месте чаще всего и обнаруживаются ошибки.

Техника анализа граничных значений

На каждой границе диапазона следует проверить по три значения:

1. граничное значение;
2. значение перед границей;
3. значение после границы.

Цель этой техники — найти ошибки, связанные с граничными значениями.

Алгоритм использования техники граничных значений:

* выделить классы эквивалентности;

Как и в предыдущей технике, этот шаг является очень важным и от того, насколько правильным будет разбиение на классы эквивалентности, зависит эффективность тестов граничных значений.

* определить граничные значения этих классов;
* нужно понять, к какому классу будет относиться каждая граница;
* нужно провести тесты по проверке значения до границы, на границе и сразу после границы.

Количество тестов для проверки граничных значений будет равен количеству границ, умноженному на 3. Рекомендуется проверять значения вплотную к границе. К примеру, есть диапазон целых чисел, граница находится в числе 100. Таким образом, будем проводить тесты с числом 99 (до границы), 100 (сама граница), 101 (после границы).

**пример**

\* **Эквивалентное Разделение (Equivalence Partitioning — EP).** Как пример, у вас есть диапазон допустимых значений от 1 до 10, вы должны выбрать одно верное значение внутри интервала, скажем, 5, и одно неверное значение вне интервала — 0.

**Класс эквивалентности (equivalence class)** — одно или несколько значений ввода, к которым программное обеспечение применяет одинаковую логику.

## **Техника анализа классов эквивалентности**

это техника, при которой мы разделяем функционал (часто диапазон возможных вводимых значений) на группы эквивалентных по своему влиянию на систему значений. Такое разделение помогает убедиться в правильном функционировании целой системы — одного класса эквивалентности, проверив только один элемент этой группы. Эта техника заключается в разбиении всего набора тестов на классы эквивалентности с последующим сокращением числа тестов.

Техника анализа классов эквивалентности

Техника рекомендует проведение тестов для всех классов эквивалентности, хотя бы по одному тесту для каждого класса. Техника анализа классов эквивалентности стремится не только сокращать количество тестов, но и сохранять приемлемое тестовое покрытие.

Признаки эквивалентности тестов:

* направлены на поиск одной и той же ошибки;
* если один из тестов обнаруживает ошибку, другие скорее всего, тоже её обнаружат;
* если один из тестов не обнаруживает ошибку, другие, скорее всего, тоже её не обнаружат;
* тесты используют схожие наборы входных данных;
* для выполнения тестов мы совершаем одни и те же операции;
* тесты генерируют одинаковые выходные данные или приводят приложение в одно и то же состояние;
* все тесты приводят к срабатыванию одного и того же блока обработки ошибок;
* ни один из тестов не приводит к срабатыванию блока обработки ошибок.

### **Техника анализа классов эквивалентности алгоритм использования:**

* Определить классы эквивалентности.

Это главный шаг техники, т.к. во многом от него зависит эффективность её применения.

1. Выбрать одного представителя от каждого класса эквивалентности.

На этом этапе следует выбрать один тест из эквивалентного набора тестов.

1. Выполнение тестов.

На этом шаге следует выполнить тесты от каждого класса эквивалентности.

Если есть время, можно протестировать еще несколько представителей от каждого класса эквивалентности. Следует иметь ввиду, при правильном определение классов эквивалентности дополнительные тесты скорее всего будут избыточными и дадут такой же результат.

Техника анализа классов эквивалентности классический пример:

**Пример**

Есть поле ввода с диапазоном допустимых значений от 1 до 100.

Сами понимаете, что на 95 тестов на допустимые значения и на несметное количество тестов на недопустимые значения уйдет очень много времени. И здесь нам помогут классы эквивалентности.

Исходя из того, с одной стороны, все допустимые значения могут влиять на поле ввода одинаково, следовательно все числа от 1 до 100 можно смело считать эквивалентными. С другой стороны, все недопустимые значения должны одинаково влиять на поле ввода (в идеале не должно быть возможности ввода этих значений в поле). Таким образом, есть уже несколько классов эквивалентности:

1. допустимые значения (от 1 до 100);

недопустимые значения:

1. от — ∞ до 0;
2. от 101 до + ∞;
3. специальные символы (# @ + — / \_ : ; “ ‘ и т.д.);
4. буквы.

Используя классы эквивалентности можно протестировать поле ввода минимум из 5 тестов.

На практике классы эквивалентности обязательны при тестировании всевозможных форм и полей ввода.

**Плюсы и минусы техники анализа эквивалентных классов**

К плюсам можно отнести отсеивание огромного количества значений ввода, использование которых просто бессмысленно.

К минусам можно отнести неправильное использование техники, из-за которого есть риск упустить баги.

Подытожим — Техника анализа классов эквивалентности одна из нескольких часто применяемых техник при планировании и разработке тестов. Значительно сокращает количество тестов необходимых для проверки функционала и время, с другой стороны в не опытных руках может стать инструментом который не только не поможет найти дефекты, но и сложит ошибочное представление о покрытие приложения тестами.

\* **Попарное тестирование (Pairwise Testing)** — это техника формирования наборов тестовых данных. Сформулировать суть можно, например, вот так: формирование таких наборов данных, в которых каждое тестируемое значение каждого из проверяемых параметров хотя бы единожды сочетается с каждым тестируемым значением всех остальных проверяемых параметров.

Сформулировать суть попарного тестирования можно следующим образом: формирование таких наборов данных, в которых каждое тестируемое значение каждого из проверяемых параметров хотя бы единожды сочетается с каждым тестируемым значением всех остальных проверяемых параметров.

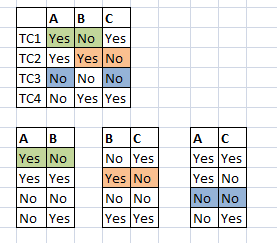
Главные цели Pairwise Testing:

* убрать избыточные проверки;
* обеспечить хорошее тестовое покрытие;
* выявить наибольшее количество багов на минимальном наборе тестов.

Рассмотрим более детально суть попарного тестирования на примерах.

**Пример 1**

Представим, что у нас есть параметры A, B и C принимающие значения Yes или No. Максимальное количество комбинаций значений этих параметров – 8. Но при использовании попарного тестирования достаточно четырех комбинаций, так как учитываются все возможные пары параметров (пара A и B, пара B и C, пара A и C):



### **2. Инструменты**

Составление нужных комбинаций данных – задача часто не самая простая, но, к счастью, для ее решения [существует множество инструментов](http://www.pairwise.org/tools.asp), разного уровня качества.

В данном материале будет рассмотрен инструмент **PICT** (**Pairwise Independent Combinatorial Testing** – инструмент для попарного тестирования от Microsoft).

PICT позволяет генерировать компактный набор значений тестовых параметров, который представляет собой все тестовые сценарии для всестороннего комбинаторного покрытия параметров.

Рассмотрим работу с программой. Запускается PICT из командной строки.

### **3. Где и когда применяется Pairwise тестирование?**

Метод эффективен лишь на поздних этапах разработки, либо дополненный основными функциональными тестами. Например, если проводить конфигурационное тестирование, то прежде чем использовать попарное тестирование следует убедиться, что основной сценарий функционирует на всех операционных системах с параметрами по умолчанию (провести **Smoke testing** (**Дымовое тестирование**) или **Build Verification Test** (**Тестирование сборки**)). Это значительно облегчит локализацию будущих багов, ведь при попарном тестировании в одном тесте фигурирует множество параметров со значениями не по умолчанию, каждый из которых может стать причиной сбоя и его локализация в этом случае весьма затруднительна. А в случае провала тестирования сборки следует отказаться от использования метода попарного тестирования, так как многие тесты будут провальными, а исключение даже одного теста влечет за собой потерю, как правило, нескольких пар, и смысл использования метода теряется.

Поэтому метод следует использовать лишь на стабильном функционале, когда текущие тесты уже теряют свою эффективность.

[Источник](http://software-testing.ru/library/testing/test-analysis/1304-pairing)

Таким образом, Pairwise Testing – специальный метод оптимизации составления тест-кейсов.

Основная суть техники Pairwise Testing – не проверить все сочетания всех значений, но проверить все пары значений.

\* **Причина / Следствие (Cause/Effect — CE).** Это, как правило, ввод комбинаций условий (причин), для получения ответа от системы (Следствие). Например, вы проверяете возможность добавлять клиента, используя определенную экранную форму. Для этого вам необходимо будет ввести несколько полей, таких как «Имя», «Адрес», «Номер Телефона» а затем, нажать кнопку «Добавить» — это «Причина». После нажатия кнопки «Добавить», система добавляет клиента в базу данных и показывает его номер на экране — это «Следствие».

\* **Предугадывание ошибки (Error Guessing — EG).** Это когда тестировщик использует свои знания системы и способность к интерпретации спецификации на предмет того, чтобы «предугадать» при каких входных условиях система может выдать ошибку. Например, спецификация говорит: «пользователь должен ввести код». Тестировщик будет думать: «Что, если я не введу код?», «Что, если я введу неправильный код? », и так далее. Это и есть предугадывание ошибки.

\* **Исчерпывающее тестирование (Exhaustive Testing — ET)** — это крайний случай. В пределах этой техники вы должны проверить все возможные комбинации входных значений, и в принципе, это должно найти все проблемы. На практике применение этого метода не представляется возможным, из-за огромного количества входных значений.