**3)-Тест дизайн**

Определяем как именно должно быть протестировано то, что было определено в рамках анализа. Другими словами это процесс перевода анализа тестирования в конкретные тесты (чек-листы, тест-кейсы и т.д.).

Чек лист - некий общий список того что мы будем тестировать

Тест кейс - детальная проработка каждого пункта чек листа где мы прописываем шаги последовательности тестирования(нажать кнопку, перейти туда-то, ввести то-то… конкретные шаги).

Если система маленькая возможно обойтись только чек листом, но если система большая и на проекте работает много тестировщиков то нужны чек листы, где конкретно описано как и что надо проверить и что для этого надо. Особенно когда есть текучка кадров и на проект приходит новый (возможно мало опытный)специалист.

Этот этап отвечает на вопрос - Как будем тестировать?

- На основе Test scenario создаём **Test case** - основной документ(артефакт) тестировщика (пошаговая инструкция проверки определённого test scenario)

- Обьеденяем test case в Test suits (тестовые наборы). Группируем для удобства:

test cases- для одной страницы.

тесты для определённого функционала

Тесты по платформам (Chrome, Firefox)

По значению: тестирование удобства пользования, тестирование безопасности…

- Расставляем приоритеты

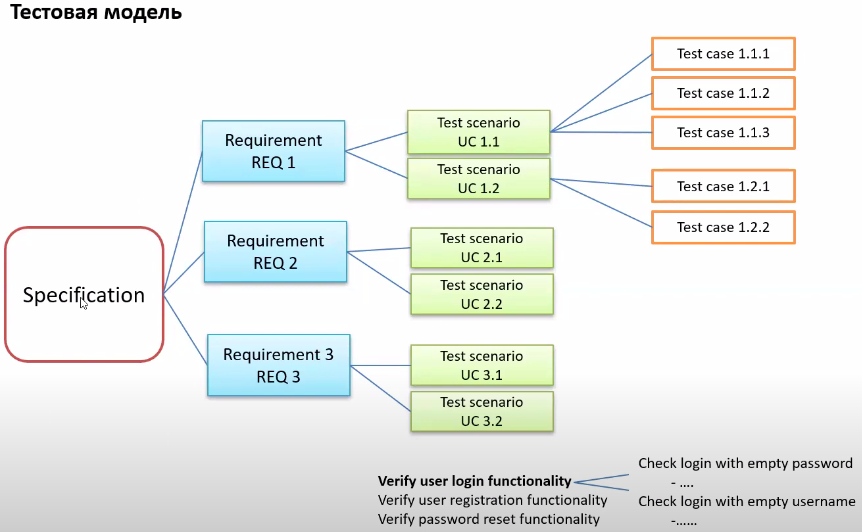
- Подготавливаем тестовые задания

- Формируем требования и настраиваем тестовую среду, инструменты. Например, для мобильного тестирования нужны разные устройства с разной версией.

- Traceability matrix

**Тестовая модель**

Это наш Анализ и дизайн (2-а этапа)



Specification - требования к системе, возможно написанный заказчиком в word документе, типа я хочу чтобы у меня была система, которая делает то-то…Далее аналитики разбивают требования по пунктам, и нумеруются, чтобы облегчить одну большую задачу на более мелкие и чтобы при необходимости легко можно было найти требуемую спецификацию.(на рисунке это голубым Requriment1,2…). Далее каждое такое requerment разбивают на Тест сценарии (их ещё могут называть - use case) их тоже может быть несколько (на ресунке это салатовым). На каждый тестовый сценарий пишутся test case-ы. Важно вести нумерацию для общего понимание что этот тестовый сценарий имеет столько-то тест кейсов и они имеют вот такой-то номер (смотри рисунок), это помогает покрыть все требования тестами и ничего не пропустить.

Например:

Есть тест сценарио: **Check the login functionality**, затем я пишу тест кейсы к нему

1)- Check system behavior when valid email id and valid password is entered.

2)- Check system behavior when invalid email id and valid password is entered.

3)- Check system behavior when valid email id and invalid password is entered.

4)- Check system behavior when valid invalid email id and invalid password is entered.

5)- Check system behavior when email id and password are left blank and Sign in entered.

6)- Check forgot your password is working as expected

7)- Check system behavior when valid/invalid phone number and password is entered.

8)- Check system behavior when “Keep me signed in” is checked.

**У тест кейса должно/могут быть самые простые/базовые поля:**

**1**-Уникальный номер

**2**-Описание того, что мы делаем

**3**-Предварительные условия (в каком состоянии должна находится программа перед тем как начать тест)

**4**-Шаги, последовательность действий,чтобы выполнить тест кейс

**5**- что я в результате получу. Actual result.

**6**- Статус-либо pass или fail

Надо стараться/желательно избегать:

1)-Зависимость от других тестов (что то типа: сначала выполни тест 2.5 а затем вот это) полностью избежать этого нельзя но надо постараться чтобы тесты были автономны.

2)-Нечёткая формулировка ожидаемого результата

3)-Отсутствие необходимой информации и в тоже время наличие излишней информации, надо придерживаться золотой середины)))

**Техники тест дизайна**(как писать тест кейсы).

Для эффективного написания тест кейсов надо следовать определённым техникам, приёмам и помнить что нельзя всё протестировать на 100 % !!!

**1) Equivalance partitioning** - тестирование на основе классов эквивалентности. Деление входных данных на группы эквивалентности по своему влиянию на систему. Включаем позитивные, негативные значения.

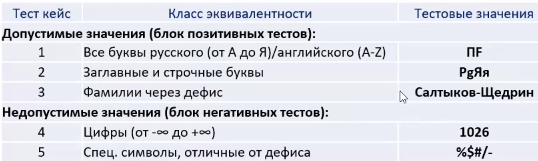
т.е. мы разбиваем все возможные значения на группы, которые называются классы эквивалентности.

Пример: надо протестировать поле **Фамилия**, допускается ввод руских, английских букв и один спец. символ дефис.

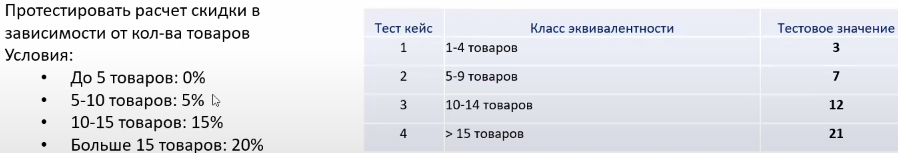
Невозможно ввести фамилии которые содержали бы все буквы руские, затем фамилии содержащие все буквы английские, затем фамилии с дефисом(Салтыков-Щедрин), это очень много тест кейсов.

Мы разбиваем по группам / классов эквивалентности:

Достаточно взять из каждой группы по 1 значению из каждой группы. В таблице ниже в колонке тестовые значения указано какое значение соответственно мы будем вводить.



ещё пример



**2)Boundary value testing** - техника анализа граничных значений. Проверка поведения системы на граничных входных данных. Подвид: equivalence partitioning для цифровых значений.

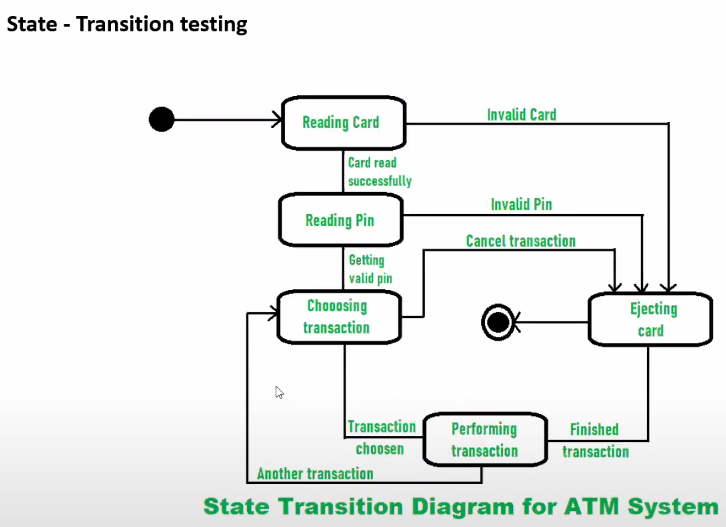
Другими словами ошибки часто встречаются на границах, в первом условии кол-во товаров от 0 до 5, соответственно протестируем нижнюю границу, т.е. 0, граничными значениями будут:-1,0,1. Во втором условии от 5, значит протестируем 4,5,6,... Такой подход намного эффективней в данном случае, нежели бы мы взяли классы эквивалентности. А как понять где что лутше? Смысл, если можно разбить на чёткие границы, то тогда лучше использовать технику анализа граничных значений, так как в этих местах ошибки найболее вероятны.

Пример:



**3)State-Transition testing** - тестирование на основе состояний переходов. Применяется для фиксирования требований и описания дизайна приложения.

Ниже диаграмма для банкомата которая описывает функционал устройств когда у них есть статусы и между ними переходы.

Когда поместили карту в банкомат, происходит чтение карты, дальше возможны варианты что карта инвалид, тогда банкомат просто реджектит карту(выплёвывает), всё это конечная точка. Если карта успешно прочитана то мы переходим в следующее состояние-чтение пин кода. Опять же 2 выхода с него есть инвалидный, тогда опять карту реджектит и валидный тогда переходим в следующее состояние выбора транзакции на этом этапе можно зделать кансел, банкомат отдаст карту, либо выбираем какую то транзакцию, тем самым переходим на стадию выполнения транзакции, откуда у нас опять 2 варианта либо завершить транзакцию и забрать карту, либо выбрать ещё одну транзакцию и перейти в стадию выбора транзакции.

Для тестирования такого рода диаграмм достаточно написать тесты которые покроют переход из состояния в состояние хотя бы один раз. Например:

1 тест кейс:

1.1-вставить карту

1.2-карта принята

1.3-ввести пин код

1.4-выбрать транзакцию.

1.5-совершить транзакцию

1.5-забрать карту

2 тест кейс:

2.1-вставить карту

2.2-карта неисправна, забрать карту

Т.е. надо по разу протестить каждый возможный вариант/кейс.

**4)-Decision Table tasting** - техника основанная на методе чёрного ящика, которая применяется для систем со сложной логикой.

**5)-Pair-wise testing** - метод пар

It operates on the principle that a majority of defects are caused by interactions between two parameters.

Example:

Let's consider a simple example of an online registration form to illustrate the Pairwise Testing Method. The registration form has the following input fields:

Country (3 options: USA, Canada, UK) ← 1

Language (2 options: English, French) ← 2

Age Group (3 options: 18-25, 26-40, 41-60) ← 3

Subscription Type (2 options: Free, Premium) ← 4

To test all possible combinations of input values, we would need 3 \* 2 \* 3 \* 2 = 36 test cases. However, using pairwise testing, we can reduce the number of test cases while still testing all pairs of input values. Statistics say that the most common places for bugs are a combination of two parameters. To find the main errors, it is enough to break the parameters into unique pairs and test them -> от этого и название pair wise (дели на пары грамотно)

Pairwise testing generates test cases in such a way that each pair of input values is covered at least once. For our example, the generated test cases could be as follows:

Country: USA, Language: English, Age Group: 18-25, Subscription Type: Free

Country: USA, Language: French, Age Group: 26-40, Subscription Type: Premium

Country: USA, Language: English, Age Group: 41-60, Subscription Type: Premium

Country: Canada, Language: French, Age Group: 18-25, Subscription Type: Premium

Country: Canada, Language: English, Age Group: 26-40, Subscription Type: Free

Country: Canada, Language: French, Age Group: 41-60, Subscription Type: Free

Country: UK, Language: English, Age Group: 18-25, Subscription Type: Premium

Country: UK, Language: French, Age Group: 26-40, Subscription Type: Free

Country: UK, Language: English, Age Group: 41-60, Subscription Type: Free

With these nine test cases, we cover all pairs of input values for each field:

All three countries are paired with both languages.

All three countries are paired with all age groups.

All three countries are paired with both subscription types.

Both languages are paired with all age groups.

Both languages are paired with both subscription types.

All age groups are paired with both subscription types.

Наверно до меня дошло:сначала берём первый инпут(здесь - Country) и обеспечиваем ему уникальные элементы с вторым инпутом (здесь - Language), затем с трейтим (Age Group) с четвёртым (Subscription Type), в общем со всеми

This example demonstrates how pairwise testing significantly reduces the number of test cases while maintaining adequate test coverage of the interactions between input parameters.