Техники тест-дизайна

Test Design.

Тест дизайн – это этап процесса тестирования ПО, на котором проектируются и создаются тестовые случаи (тест кейсы), в соответствии с определёнными ранее критериями качества и целями тестирования.

Test design [**ISTQB Glossary of terms**]: The process of transforming general testing objectives into tangible test conditions and test cases.

Test Design. План работы.

- Анализ имеющихся проектных артефактов: документация (**спецификации, требования, планы**), модели, исполняемый код и т.д.
- Написание спецификации по тест дизайну (**Test Design Specification**)

проектирование и создание тестовых случаев (**Test Cases**)

Test Design. Roles.

Тест аналитик - определяет **"ЧТО тестировать?"**
Тест дизайнер - определяет **"КАК тестировать?"**

Попросту говоря, задача тест аналитиков и дизайнеров сводится к тому, чтобы используя различные стратегии и техники тест дизайна, создать набор тестовых случаев, обеспечивающий оптимальное тестовое покрытие тестируемого приложения. Однако, на большинстве проектов эти роли не выделяется, а доверяется обычным тестировщикам, что не всегда положительно сказывается на качестве тестов, тестировании и, как из этого следует, на качестве программного обеспечения (конечного продукта)

Тест аналитик.

- Исследует продукт;
- Составляет логическую карту продукта;
- Разбивает программный продукт на основные части;
- Расставляет приоритеты для тестирования.

Тест аналитик. Исследование программного продукта.

- Понимание цели создания продукта;
- Какими способами цель должна достигаться;
- Какие и основные и вспомогательные возможности предоставляет продукт пользователям;
- Оценка, правильно ли понял разработчик заказчика.

Тест аналитик. Логическая карта продукта.

Интеллект - карта - техника представления любого процесса, события, мысли или идеи в систематизированной визуальной форме.

Тест аналитик. Логическая карта продукта.

Интеллект - карта. Для чего?

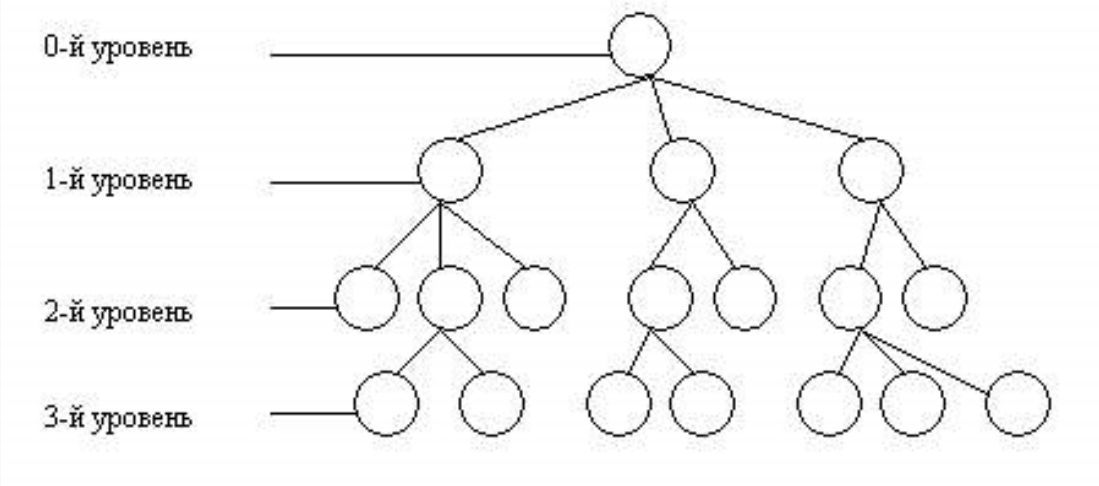
- Цельный взгляд на весь проект дает возможность отследить логические связи внутри продукта.
- При детализации карты достигается разделение функций до наименьших, атомарных составляющих (собственно, выполняем декомпозицию продукта).

Интеллект - карты. Инструменты.

- coggle.it
- xmind.net
- mindomo.com

Что такое декомпозиция?

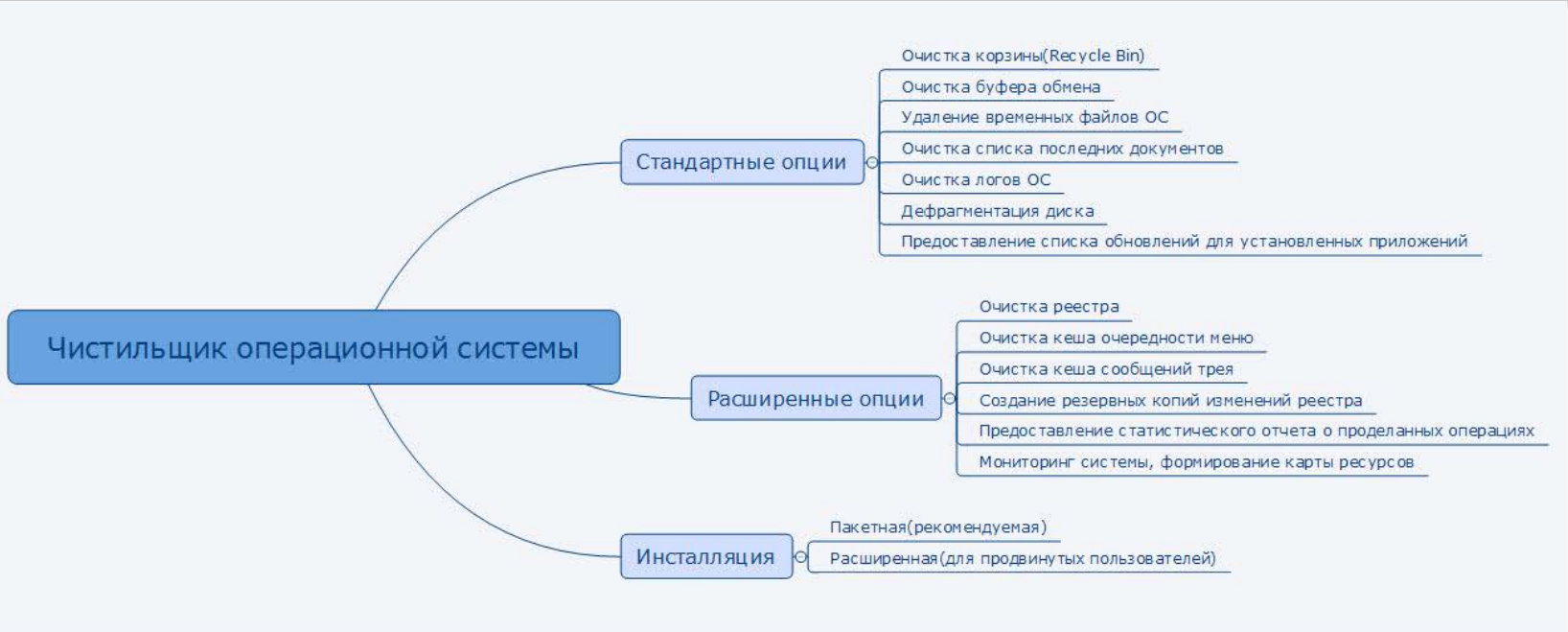
- Каждое расчленение образует свой уровень;



Что такое декомпозиция?

- Система расчленяется только по одному, постоянному для всех уровней признаку (Они должны отвечать на один и тот же вопрос, по отношению к своему родителю);
- Вычленяемые подсистемы должны взаимно исключать друг друга, а в сумме - характеризовать систему.
- На каждом уровне рекомендуется использовать не более 7 подсистем.

Что такое декомпозиция? Пример.



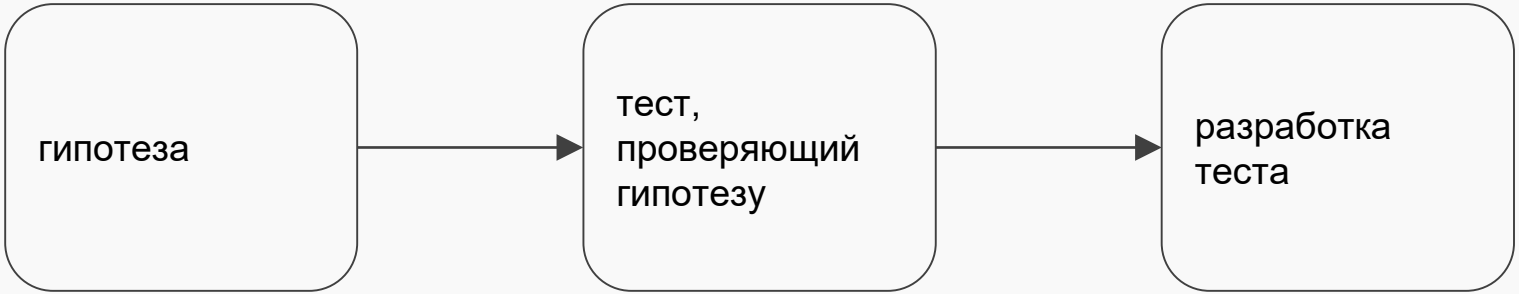
Приоритезация.

- Требования клиента;
- Степень риска;
- Сложность системы;
- Временные ограничения.

Тест дизайнер.

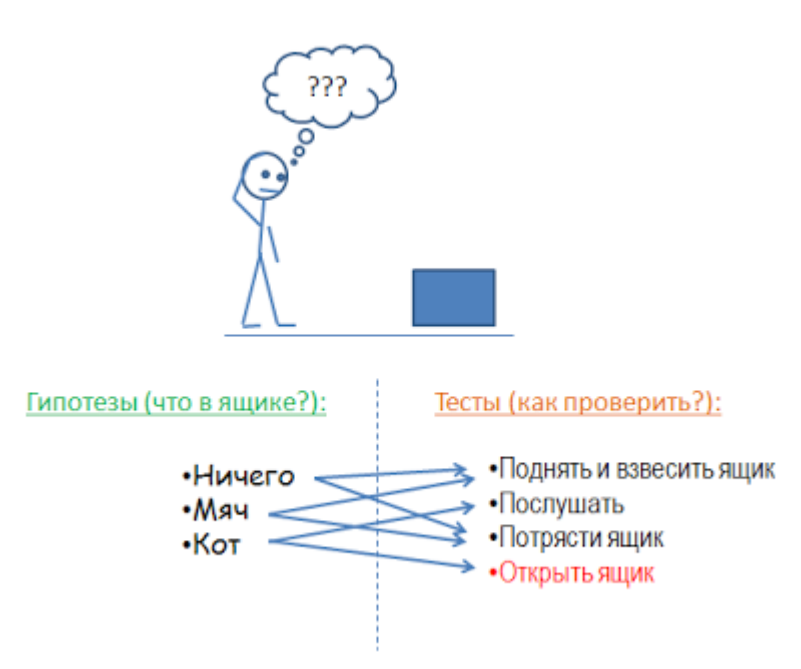
Человек, который должен выстроить процесс тестирования всех важнейших частей продукта, используя минимально возможное количество проверок.

Test Design.



Каждый тест либо подтверждает, либо опровергает нашу гипотезу.

Test Design.


???
Гипотезы (что в ящике?):
Тесты (как проверить?):
•Ничего
•Мяч
•Кот
•Поднять и взвесить ящик
•Послушать
•Потрясти ящик
•Открыть ящик

Test Design.

Можно провести аналогию с тестированием программного продукта. Гипотезы, которые мы можем проверять в этом случае:

- Программа работает неправильно;
- Программа работает правильно;
- Программа удобна;
- Программа работает быстро;

Test Design.

И возможные тесты по проверке будут такими:

- Проверить работу одной функции;
- Проверить работу другой функции программы;
- Проверить работу интерфейса;
- Измерить время отклика программы на действия пользователей;

Test Design. Цели.

Главные цели:

1. Придумать тесты, которые обнаружат наиболее серьезные ошибки продукта. Да, мы можем придумывать тесты, которые находят несерьезные ошибки, но тогда тестирование будет неэффективным.

2. Минимизировать количество тестов, необходимых для нахождения большинства серьезных ошибок. Мы может придумать столько тестов, сколько не в состоянии будем выполнить. Поэтому перед разработчиками тестов всегда стоит задача – сохранить эффективность тестов (то есть их способность обнаруживать серьезные ошибки) без увеличения их числа.

Test Design. Необходимые навыки.

Умение разделять систему на составляющие (делать декомпозицию). То есть, нужно уметь видеть систему как целое, но и уметь раскладывать ее на составные части. Это очень полезный навык для проведения функционального тестирования, где проверяется каждая фича продукта.

Умение собирать и анализировать требования к продукту. Если нет формально описанных требований – нужно уметь их собирать у разработчиков, у аналитиков, у пользователей.

Умение расставлять приоритеты. Тест дизайнер должен уметь отличать более важное от менее важного, и расставлять приоритеты тестирования.

Умение формулировать свои мысли (письменно и устно). Это умение важно для тестировщика в принципе. Тест дизайнеру оно здорово поможет при создании тест кейсов.

Знание техник тест дизайна.

Умение применять их на практике.

Test Design Technics.

Многие тестируют и пишут тестовые случаи (test cases), но не многие пользуются специальными техниками тест дизайна. Постепенно, набираясь опыта они осознают, что постоянно делают одну и ту же работу, поддающуюся конкретным правилам. И тогда они находят, что все эти правила уже описаны.

Test Design Technics. Главные.

1. Эквивалентное разделение (Equivalence Partitioning - EP).
2. Анализ граничных значений (Boundary Value Analysis - BVA).
3. Предугадывание ошибки (Error Guessing - EG).
4. Исчерпывающее тестирование (Exhaustive Testing - ET).
5. Причина/Следствие (Cause/Effect - CE)

Test Design Technics. Главные.

6. Таблица принятия решений (Decision table).

7. Тестирование состояний и переходов (State - transition testing).

8. метод парного тестирования (Pairwise testing).

Test Design Technics. Эквивалентное разделение.

Тестовые данные разбиваются на определенные классы
допустимых значений. В рамках каждого класса выполнение теста с любым значением тестовых данных
приводит к эквивалентному результату

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

Test Design Technics. Эквивалентное разделение. Алгоритм.

1. Определить классы эквивалентности.

2. Выбрать одного представителя от каждого класса.

3. Выполнить тесты.

Test Design Technics. Эквивалентное разделение.

Представим, что мы тестируем модуль для рекрутера, который проверяет возможность принятия на работу кандидата, в зависимости от его возраста. Установлены следующие условия отбора:

- 0 - 16 лет - не нанимать;
- 16 - 18 лет - можно нанять только на неполный рабочий день;
- 18 - 55 лет - можно нанять на полный рабочий день;
- 55 - 99 лет - не нанимать.

Test Design Technics. Эквивалентное разделение.

- Класс эквивалентности NO: 0 - 16 лет;
- Класс эквивалентности PART: 16 - 18 лет;
- Класс эквивалентности FULL: 18 - 55 лет;
- Класс эквивалентности NO: 55 - 99 лет.

Test Design Technics. Эквивалентное разделение.

Таким образом у нас остается 4 позитивных тест - кейса из возможных 100 (0 - 99):

- 12 - не нанимать;
- 17 - нанять на неполный рабочий день;
- 50 - нанять на полный рабочий день;
- 89 - не нанимать.

Test Design Technics. Анализ граничных значений.

В отличии от эквивалентного разбиения, которое ориентировано на покрытие тестами, эта техника основана на рисках - программа может сломаться в области граничных значений.

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.

Test Design Technics. Анализ граничных значений.

Эта техника основана на том факте, что одним из самых слабых мест в приложении является область граничных значений.

Test Design Technics. Анализ граничных значений.

Вернемся к примеру, рассмотренному в технике “Классов эквивалентности”:

- 0 - 16 лет - не нанимать;
- 16 - 18 лет - можно нанять только на неполный рабочий день;
- 18 - 55 лет - можно нанять на полный рабочий день;
- 55 - 99 лет - не нанимать.

Test Design Technics. Анализ граничных значений.

Для правильного применения техники, необходимо записать условие по - другому:

- 0 - 15 лет - не нанимать;
- 16 - 17 лет - нанимать на неполный рабочий день;
- 18 - 54 года - нанимать на полный рабочий день;
- 55 - 99 лет - не нанимать.

Test Design Technics. Анализ граничных значений.

Таким образом набор значений, по которым будут составлены тесты будет выглядеть так:

{-1, 0, 1}, {15, 16, 17}, {17, 18, 19}, {54, 55, 56}, {98, 99, 100}.

Test Design Technics. Предугадывание ошибки.

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". Тест аналитик, будет думать: "Что, если я не введу код?", "Что, если я введу неправильный код? ", и так далее. Это и есть предугадывание ошибки

Test Design Technics. Исчерпывающее тестирование.

Крайний случай.
Тестировщик должен 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

Test Design Technics.
Причина/Следствие.

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".

Test Design Technics. Таблица принятия решений.

Это удобный инструмент для фиксирования требований и описания функциональности приложения. Таблицей удобно описывать бизнес - логику приложения и они могут служить отличной основой для тест - кейсов.

Test Design Technics. Таблица принятия решений.

Таблицы принятия решений описывают логику приложения основываясь на условиях системы, характеризующих ее состояния. Каждая таблица должна описывать одно состояние системы.

Таблица принятия решений. Пример.

	Тест 1	Тест 2	Тест 3	Тест 4
Условия				
Состоит в браке	Да	Да	Нет	Нет
Хороший студент	Да	Нет	Да	Нет
Действия				
Дать скидку($)	60	50	40	30

Таблица принятия решений. Пример.

Предоставление скидки в зависимости от комбинации условий будет нашим действием в приложении.

После этого нам следует создать по одному тест - кейсу для каждого из предполагаемых действий.

Test Design Technics. Таблица состояний и переходов.

Система переходит в то или иное состояние в зависимости от того, какие действия над ней выполняются.

Таблица состояний и переходов. Пример.

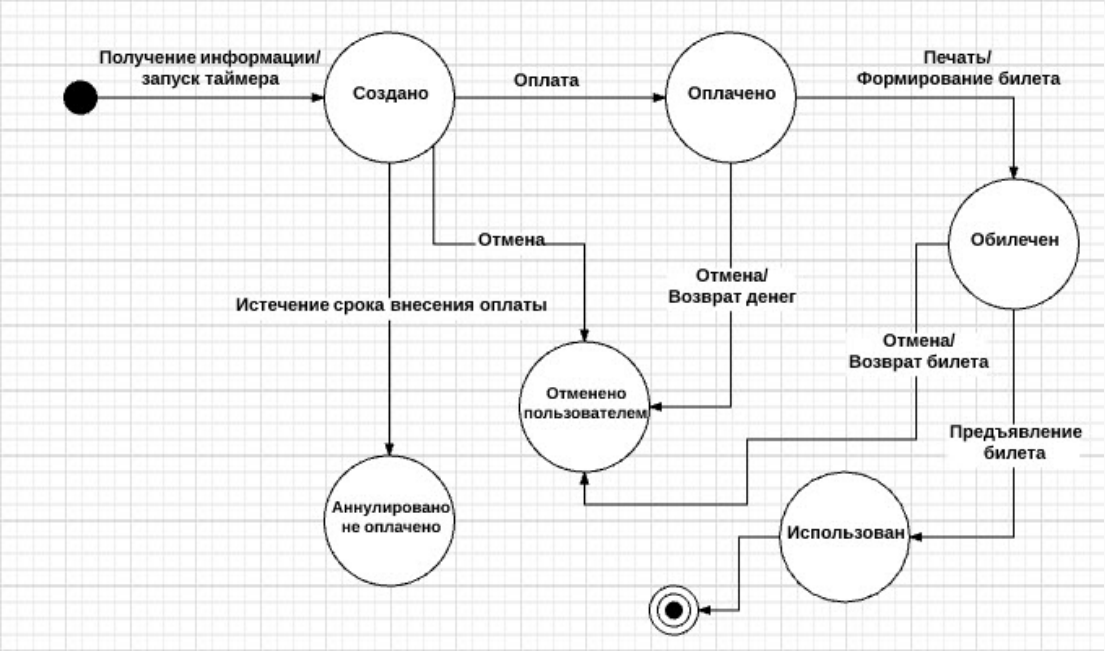


Таблица состояний и переходов.

- Состояние (представлено в виде круга);
- Переход (Представлен в виде стрелки);
- Событие (Представлено ярлыком над стрелкой);
- Действие (Представлено после “/”);
- Точка входа (Представлена черным кругом);
- Точка выхода (Представлена мишенью).

Test Design Technics. Метод парного тестирования.

Основан на следующей идее: подавляющее большинство багов, выявляются тестами, проверяющими либо один параметр, либо сочетание двух параметров.

Ошибки которые появились сочетанием трех и более параметров, как правило, значительно менее критичны.

Test Design Technics. Другие

Функциональное тестирование (это проверка всех функций продукта, одна за одной)
Тестирование на основе сценариев использования (это проверка продукта по наиболее частым и важным сценариям использования – use cases)
Тестирование на основе рисков (это проверка важных проблем, которые могут возникнуть)
Unit-тестирование
Регрессионное тестирование
Тестирование безопасности

Совсем другие :)

Партизанское тестирование – попытка найти ошибки в некоторой области программы, причем тесты выполняются быстрые и вредоносные.
Есть еду своей собаки – когда продукт используется внутри самой компании, в повседневной работе.
Тестирование глупой обезьяны – беспорядочное автоматическое тестирование, когда с клавиатуры вводятся случайные числа и мышь кликает по случайным местам экрана.
Тестирование мыльной оперы – когда проверяются слегка усложненные и расширенные сценарии реального использования. Главное в одном сценарии проверить как можно больше всего, как в мыльной опере, в которой в одной серии может произойти столько событий, сколько умещается во всей жизни

ПРИДУМАЙ САМ :)

Техники тестирования. Эквивалентное разбиение

Рассмотрим **пример**

- Программа складывает два целых числа
- Каждое из слагаемых – не более чем целое двузначное число
- Программа запрашивает у пользователя два числа и выводит результат

Классы эквивалентности

	Классы корректных данных	Классы некорректных данных	Граничные и специальные значения
Первое слагаемое	от -99 до -10 от -9 до -1 0 от 1 до 9 от 10 до 99	> 99 < -99	0, 1, -1, 9, -9 10, -10 99, -99 100, -100
Второе слагаемое	-”-”-	-”-”-	-”-”-
Сумма	от -198 до -100 от -99 до -1 0 от 1 до 99 от 100 до 198	> 198 < -198	(-99, -99) (-49, -51) (99, 99) (49, 51)

Техники тестирования. Эквивалентное разбиение

Какие еще сущности можно разбивать на классы эквивалентности

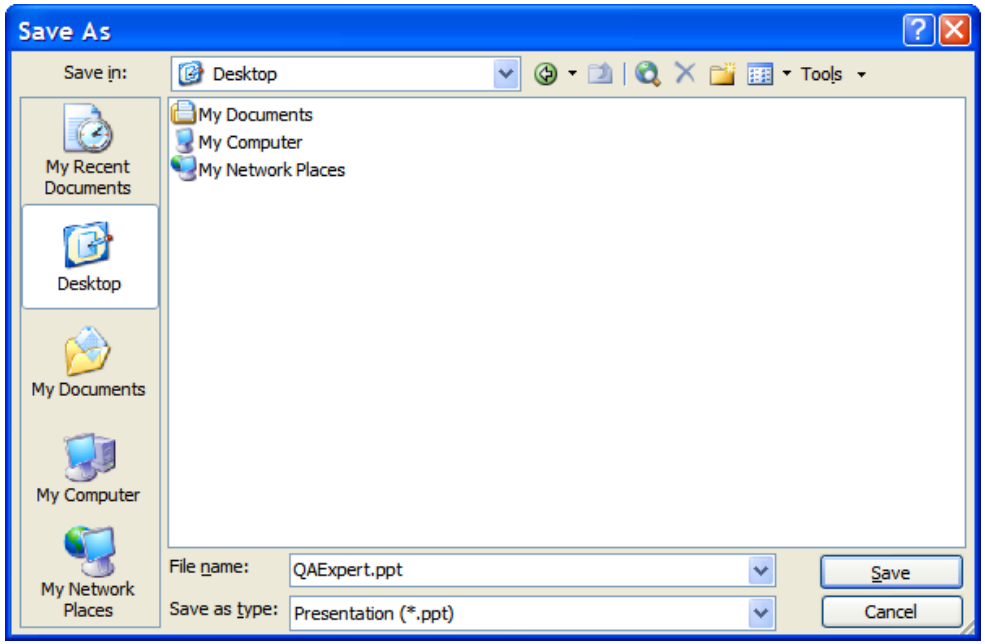
- числа
- символы
- количество (записей в БД, строк)
- длина строки
- размер файла
- объем памяти
- разрешение экрана
- версии операционной системы, библиотек
- объем передаваемых данных

Предугадывание ошибки

Описание тестируемого функционала:

- ❑ Поле для ввода названия папки
- ❑ Кнопка «Сохранить»
- ❑ Название папки не должно превышать 64 символа
- ❑ Ваши предложения?

Диалог сохранения файла



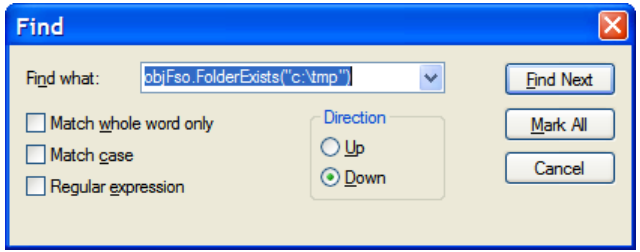
«Фиксируем шаги»

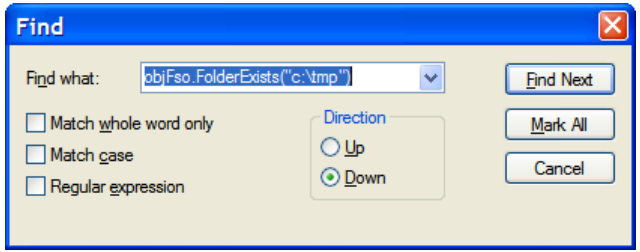
- Сначала выделяем наиболее рискованные (и важные) области – собственно сохранение , выбор нужного места, сохранение с длинным именем, с национальными символами, перезапись и т.п.
- Потом выясняем какие сценарии использования (use case)
- Выясняем классы эквивалентности
- Пишем тест-кейсы (позитивные, негативные, исследовательские)

Парное тестирование

Рассмотрим пример

- Окно поиска в текстовом редакторе





- **Подсчитаем количество тестов**
- 5 переменных:
  - Find what (FW) – строка
  - Match whole words only (MW) – Boolean
  - Match case (MC) – Boolean
  - Regular expression (RE) – Boolean
  - Direction (D) – перечисляемый тип (Up, Down)
- Тестовые значения
  - FW = {‘lower’; ‘UPPER’; ‘MiXeD’}
  - MW, MC, RE = {Yes; No}
  - D = {Up; Down}
- Итого: 3 x 2 x 2 x 2 x 2 = 48 тестов

Способы снижения количества тестов

- **Выбор комбинаций**
- Для данного случая методы выбора на основе рисков и на основе сценариев малопригодны
- Оптимальнее использовать механический перебор по некоторой системе:
  - Полный перебор
  - Все пары (каждый с каждым)
  - Все значения хотя бы по разу

Способы снижения количества тестов

Полный перебор (все Nки)

	FW	MW	MC	RE	D
1	L	Y	Y	Y	Up
2	U	Y	Y	Y	Up
3	M	Y	Y	Y	Up
4	L	Y	Y	Y	Up
5	L	N	Y	Y	Up
…	…	…	…	…	…
47	M	N	N	N	Up
48	M	N	N	N	D

Способы снижения количества тестов

Все значения хотя бы по разу

	FW	MW	MC	RE	D
1	L	Y	N	Y	Up
2	U	N	Y	N	D
3	M	Y	Y	N	Up

3 теста, а не 48

Способы снижения количества тестов

Все пары (каждый с каждым)

	FW	MW	MC	RE	D
1	L	Y	N	Y	Up
2	L	N	Y	N	D
3	U	Y	Y	N	Up
4	U	N	N	Y	D
5	M	N	N	N	Up
6	M	Y	Y	Y	D

- Этот метод является «золотой серединой»
- Метод «всех пар» хорошо работает для независимых переменных
- Зачастую случайное тестирование хорошо приближается к методу «всех пар»