Курс «Тестирование ПО»
Тема 4
«Разработка тестов»

Введение

Мы рассмотрели общие вопросы организации процесса тестирования и вопросы планирования тестовых испытаний. Следующие за планированием является стадия разработки тестов. От того, насколько качественно будет выполнена эта стадия зависит процесс выполнения тестов и документирования результатов их выполнения. В этой теме мы будем учиться разрабатывать тесты. Итак, приступим.

Какие бывают тесты. Обсуждение.

Перед вами обыкновенная ручка. Давайте подумаем, как её можно протестировать?

Какие бывают тесты (1/12)

Тесты на основе требований (requirements based tests)

- Извлекается и вставляется ли в ручку стержень?
- Присутствует ли держатель, позволяющий цеплять ручку за край кармана?
- Переключается ли ручка из рабочего в нерабочее положение?

Какие бывают тесты (2/12)

Функциональные тесты (functional test)

- Вставить в ручку стержень.
- Переключить в рабочее положение.
- Написать несколько слов.
- Переключить в нерабочее положение.
- Извлечь стержень.

Какие бывают тесты (3/12)

Сравнительные («параллельные») тесты (parallel testing)

- Что мы можем сказать об этой ручке в сравнении с другими ручками, которые выпускает наша фирма?
- Что мы можем сказать об этой ручке в сравнении с ручками, которые выпускают конкуренты?
- В чём преимущества именно этой модели ручек?

Какие бывают тесты (4/12)

Сценарные тесты (scenario tests)

Как ручку может использовать:

- Секретарь.
- Преподаватель.
- Студент.
- Школьник.
- Прораб.
- Сантехник.
- Милиционер.
- Моряк.
- …

Какие бывают тесты (5/12)

Тесты ошибочных ситуаций (fault injection tests)

- Что произойдёт, если препятствовать выходу стержня в рабочее положение?
- Какое усилие и где надо приложить к ручке, чтобы её сломать?
- Если стержень застрял, легко ли его извлечь?
- Что произойдёт, если писать по стеклу, асфальту?

Какие бывают тесты (6/12)

Тесты интерфейса (interface tests, GUI tests)

- Измерения: высота, ширина, длина, вес.
- Цвет.
- Читаемость логотипа фирмы-производителя.

Какие бывают тесты (7/12)

Тесты удобства использования (usability tests)

- Есть ли у нас какие-либо замечания по юзабилити ручек от пользователей?
- Есть ли у нас представители целевых групп, чтобы привлечь их к тестированию?
- Как много времени у пользователя занимает переключение ручки из нерабочего положения в рабочее и обратно?
- Как быстро пользователь понимает, как пользоваться ручкой?
- Как быстро пользователь привыкает к этой ручке?
- Легко ли понять, какие стержни подходят к ручке?
- Легко ли заменить стержень?
- Может ли ручкой пользоваться левша?

Какие бывают тесты (8/12)

Тесты упаковки и документации (packaging/documentation tests)

- Вложена ли в упаковку копия текста о гарантийных обязательствах?
- Ясно ли видно на упаковке, что внутри?
- Легко ли открыть упаковку?
- Насколько материалы упаковки вредны для окружающей среды?
- Есть ли какие-то особые требования к упаковке?
- На сайте, в каталоге, на упаковке написано и нарисовано одно и то же?
- Текст на упаковке и в гарантийном обязательстве – на одном и том же языке?
- На упаковке и в документации нет грамматических ошибок, опечаток и т.д.?

Какие бывают тесты (9/12)

Стрессовые тесты (stress tests)

- При какой температуре расплавится пластиковая часть ручки?
- При какой температуре потечёт стержень?
- При какой температуре ручка перестаёт писать?
- Какое воздействие необходимо применить к ручке, чтобы сломать её?
- Пишет ли ручка под водой? А по мокрой бумаге?
- Если ручку уронить в песок – что произойдёт?
- А если уронить со стола?
- А если из окна офиса?

Какие бывают тесты (10/12)

Тесты производительности (performance tests)

- Сколько текста можно написать ручкой в единицу времени?
- Как быстро ручку можно привести в рабочее положение?
- Как много раз ручку можно переключить из нерабочего в рабочее положение, прежде чем её начнёт заедать?

Какие бывают тесты (11/12)

Конфигурационные тесты (configuration tests)

- Какие стержни подходят к нашей ручке?
- На каких поверхностях она может писать?

Какие бывают тесты (12/12)

Законодательные тесты (regulation tests)

- Подлежит ли этот продукт какому-то виду лицензирования?
- Необходима ли какая-то особая сопроводительная документация?
- Ясно ли из документации ручки видно, в какой стране она произведена?
- Существуют ли какие-то законодательные особенности, препятствующие распространению нашего продукта?

Задача о треугольнике (1/5)

Эту задачу предложил в 1979 году Гленфорд Майерс в своей книге «Искусство тестирования программ» («The Art Of Software Testing»). С тех пор она известна как «задача о треугольнике» и является своего классическим вопросом на множестве собеседований на должность тестировщика.

Программа производит чтение с перфокарты трёх целых чисел, которые интерпретируются как длины сторон треугольника. Далее программа печатает сообщение о том, является ли треугольник неравносторонним, равнобедренным или равносторонним.

Напишите на листе бумаги (или в файле) набор тестов, которые, как вам кажется, будут адекватно проверять эту программу. На это у вас десять минут.

Задача о треугольнике (2/5)

Были изучены различные версии данной программы и составлен список общих ошибок. Оцените ваш набор тестов, попытавшись с его помощью ответить на приведенные ниже вопросы. За каждый ответ «да» присуждается одно очко.

1. Составили ли вы тест, который представляет правильный неравносторонний треугольник? (Ответ «да» на этот вопрос не засчитывается, если вы имеете в виду тесты, проверяющие значения длин сторон вида «1, 2, 3» или «2, 5, 10», так как не существует треугольников, имеющих такие стороны.)
2. Составили ли вы тест, который представляет правильный равносторонний треугольник?
3. Составили ли вы тест, который представляет правильный равнобедренный треугольник? (Тесты со значениями сторон «2, 2, 4» принимать в расчёт не следует, т.к., опять же, не бывает таких треугольников.)

Задача о треугольнике (3/5)

4. Составили ли вы по крайней мере три теста, которые представляют правильные равнобедренные треугольники, полученные как перестановки двух равных сторон треугольника (например, «3, 3, 4», «3, 4, 3», «4, 3, 3»)?
5. Составили ли вы тест, в котором длина одной из сторон треугольника принимает нулевое значение?
6. Составили ли вы тест, в котором длина одной из сторон треугольника принимает отрицательное значение?
7. Составили ли вы тест, включающий три положительных целых числа, сумма двух из которых равна третьему? (Другими словами, если программа выдала сообщение о том, что числа «1, 2, 3» представляют собой стороны неравностороннего треугольника, то такая программа содержит ошибку.)
8. Составили ли вы по крайней мере три теста с заданными значениями всех трёх перестановок, в которых длина одной стороны равна сумме длин двух других сторон (например, «1, 2, 3», «1, 3, 2», «3, 1, 2»)?

Задача о треугольнике (4/5)

9. Составили ли вы тест из трёх целых положительных чисел, таких, что сумма двух из них меньше третьего числа (т. е. «1, 2, 4» или «12, 15, 30»)?
10. Составили ли вы по крайней мере три теста из категории 9, в которых вами испытаны все три перестановки (например, «1, 2, 4», «1, 4, 2», «4, 1, 2»)?
11. Составили ли вы тест, в котором все стороны треугольника имеют длину, равную нулю (т. е. «0, 0, 0»)?
12. Составили ли вы по крайней мере один тест, содержащий нецелые значения?
13. Составили ли вы хотя бы один тест, содержащий неправильное число значений (например, два, а не три целых числа)?
14. Специфицировали ли вы в каждом тесте не только входные значения, но и выходные данные программы?

Задача о треугольнике (5/5)

Конечно, нет гарантий, что с помощью набора тестов, который удовлетворяет вышеперечисленным условиям, будут найдены все возможные ошибки. Но поскольку вопросы 1+13 представляют ошибки, имевшие место в различных версиях данной программы, адекватный тест для неё должен их обнаруживать.

Отметим, что опытные профессиональные программисты набирают в среднем только 7-8 очков из 14 возможных.

Выполненное упражнение показывает нам, что тестирование даже тривиальных программ, подобных приведенной, – непростая задача.

И коль скоро это так, представьте трудности тестирования программ размером, например, в 1’000’000 и более операторов. А если это программа по управлению реактором АЭС?

Классы эквивалентности (1/14)

На только что рассмотренных примерах с ручкой и треугольником вы могли убедиться, что набор тестов у вас получается неидеальным. А как его сделать лучше? Как убедиться, что тестов достаточно, и в то же время их не слишком много?

Для этого есть прекрасная техника – продумывание классов эквивалентности и граничных условий.

Классы эквивалентности (2/14)

Если мы ожидаем одинакового результата от выполнения двух и более тестов, эти тесты эквивалентны. Такие множества тестов называются классами эквивалентности.

Признаки эквивалентности (несколько тесто эквивалентны, если):

- Она направлены на поиск одной и той же ошибки.
- Если один из тестов обнаруживает ошибку, другие её тоже, скорее всего, обнаружат.
- Если один из тестов НЕ обнаруживает ошибку, другие её тоже, скорее всего, НЕ обнаружат.
- Тесты используют одни и те же 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 («error handling block»).
- Ни один из тестов не приводит к срабатыванию блока обработки ошибок («error handling block»).

Классы эквивалентности (3/14)

Граничные условия (или просто – границы) – это те места, в которых один класс эквивалентности переходит в другой.

Например, одна группа тестов вызывает сообщение «вы ввели слишком маленькое число», а другая вызывает сообщение «вы ввели слишком большое число». Граница будет лежать где-то в районе чисел «самых больших из слишком маленьких» и «самых маленьких из слишком больших».

Граничные условия очень важны, и их обязательно следует проверять в тестах, т.к. именно в этом месте чаще всего и обнаруживаются ошибки.

Рассмотрим на примере.

Классы эквивалентности (4/14)

Необходимо проверить, как работает поле, в которое можно ввести целое число в диапазоне от 1 до 99.

Классы эквивалентности здесь:

- Любое целое в диапазоне от 1 до 99. Как правило, это будет середина числового отрезка. Позитивный тест.
- Любое число меньше 1. Негативный тест.
- Любое число больше 99. Негативный тест.
- Дробь и «не число» (буквы, спецсимволы). Негативный тест.

Тесты, которые мы выполним:

- Ввести 1, 99, 50
- Ввести 0
- Ввести 100
- Ввести 50.5, букву, спецсимвол: ~`!”@’#$;%:^&?*()[]{},.\/+=-_

Классы эквивалентности (5/14)

Рассмотрим ещё один пример.

- Программа предназначена для сложения двух целых чисел.
- Каждое из слагаемых – не более чем двузначное целое число.
- Программа запрашивает у пользователя два числа, после чего выводит результат.

Ваши соображения?

Классы эквивалентности (6/14)

Допустим, мы написали первый тест:

Первое слагаемое:
3
Второе слагаемое:
7
Сумма:
10

Классы эквивалентности (7/14)

Нужно ли проверять такие комбинации?

2 + 7	4 + 7	5 + 7
3 + 8	3 + 6	3 + 5
3 + 3	7 + 3	7 + 7
2 + 2	6 + 4	2 + 5
(-3) + 7	(-3) + (-7)	3 + (-7)
0 + 7	3 + 0	0 + 0
99 + 7	3 + (-99)	(-99) + 0
99 + 99	99 + (-99)	(-99) + 99
100 + 3	100 + 100	100 + (-100)

Классы эквивалентности (8/14)

Выделим классы эквивалентности.

	Классы корректных данных (позитивные тесты)	Классы некорректных данных (негативные тесты)	Граничные и специальные значения	Примечания
Первое слагаемое	от -99 до 99	> 99 < -99	99, -99 100, -100	
Второе слагаемое	от -99 до 99	> 99 < -99	99, -99 100, -100	

Классы эквивалентности (9/14)

Выделим классы эквивалентности.

	Классы корректных данных (позитивные тесты)	Классы некорректных данных (негативные тесты)	Граничные и специальные значения	Примечания
Первое слагаемое	от -99 до 99	> 99 < -99	99, -99 100, -100	
Второе слагаемое	от -99 до 99	> 99 < -99	99, -99 100, -100	
Сумма	от -198 до 198	> 198 < -198	(-99, -99) (99, 99)	Как попасть в некорректные?

Классы эквивалентности (10/14)

Выделим классы эквивалентности.

	Классы корректных данных (позитивные тесты)	Классы некорректных данных (негативные тесты)	Граничные и специальные значения	Примечания
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)	Как попасть в некорректные?

Классы эквивалентности (11/14)

Порядок действий для выделения классов эквивалентности:

- В таблице перечисляются все переменные (входные и выходные).
- Для каждой переменной определяется разбиение на классы
- Строятся все возможные комбинации классов.
- В качестве представителей классов берутся тесты на граничные, приграничные или специальные значения.

Классы эквивалентности (12/14)

Что можно отразить на числовую прямую?

- Числа.
- Символы.
- Количество (например, количество разрешённых установок ПО, количество записей в БД, количество строк, количество символов).
- Размерность чисел (двоичную и десятичную).
- Длину строки, в том числе:
  - Длину конкатенации (результата объединения строк).
  - Длину пути в файловой системе или веб-ссылки.
  - Длину имени файла.
  - Длину текста в файле.
  - Длину фрагментов текста (слова, предложения, абзаца).

Классы эквивалентности (13/14)

Что можно отразить на числовую прямую? (продолжение)

- Размер файла (особенно кратный чему-либо).
- Объём памяти (особенно кратный чему-либо).
- Размер экрана, разрешение экрана.
- Размер окна.
- Количество цветов. Цветовую гамму.
- Версии операционной системы.
- Версии библиотек.
- Время (в т.ч. между событиями).
- Скорость передачи данных.
- Объём передаваемых данных.
- Интенсивность передачи данных.
- Переключение между разными алгоритмами (количество, время, скорость).

Классы эквивалентности (14/14)

Ещё один пример.

«Чтобы открыть файл, пользователь должен кликнуть по кнопке Открыть, выбрать файл и кликнуть по кнопке OK». Давайте абстрагируемся от пользовательского интерфейса и подумаем о файле. Какие случаи нам надо будет проверить?

- «Корректный» файл
- Заблокированный файл
- Очень большой файл
- Несуществующий файл
- «Файл по сети»
- Уже открытый файл (нашим приложением и другим приложением)
- Файл неверного формата (по расширению и реальному содержимому)
- Пустой файл
- Повреждённый файл

Выводы:

- Классы эквивалентности не всегда очевидны.
- Как правило, негативных тестов получается больше, чем позитивных.
- Принадлежность теста к позитивным или негативным зависит от требований.

Выводы

Рекомендации:

- Начинайте с простых очевидных тестов. Используйте простые и очевидные значения для передачи в программу. Если она завалится даже на таких значениях, это будет очевидным показателем того, что существуют проблемы.
- Затем переходите к более сложным тестам. Если программа хорошо справляется с очевидными задачами, поставьте перед ней неочевидную.
- Помните о граничных условиях. На граничных значениях можно построить много хороших тестов.
- Если остаётся время, занимайтесь исследовательским тестированием.
- Учитесь на своём и чужом опыте.

Последовательность выполнения (и разработки) тестов:

- Просты позитивные.
- Простые негативные.
- Сложные позитивные.
- Сложные негативные.

Документирование тестов (1/3)

Тесты документируются в виде т.н. «тестовых случаев» («test cases»).

Что же такое тест-кейс?

IEEE Std 610-1990:

«A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.»

(«Набор тестовых входных данных, условий выполнения и ожидаемых результатов, разработанных с конкретной целью, такой как проверка некоторого пути выполнения программы или проверка соответствия некоторому требованию.»)

IEEE Std 829-1983:

«Documentation specifying inputs, predicted results, and a set of execution conditions for a test item.»

(«Документ, определяющий набор входных данных, ожидаемых результатов и условий выполнения теста.»)

Далее в нашем курсе мы будем рассматривать слова «тест-кейс» и «тест» как синонимы (в случаях, если не оговорено обратное).

Документирование тестов (2/3)

Документируя тест, мы должны указать:

- Идентификатор теста (id).
- Связанное с тестом требование (related requirement).
- Краткое заглавие теста (title).
- Модуль и подмодуль приложения, к которым относится тест (module, submodule).
- Приоритет теста (priority: smoke, critical, extended; A, B, C, D).
- Исходные данные, необходимые для теста (initial data).
- Шаги для выполнения теста (steps).
- Ожидаемые результаты (expected results).
- Отвести поле для пометки, прошёл тест или нет (passed or failed).
- Указать автора теста (author), время последнего выполнения теста (last time run), последний полученный актуальный результат (actual result), связанный с тестом баг (если есть) (related bug).

В общем случае некоторые поля могут опускаться, и шаблон для оформления теста будет выглядеть вот так (см. рисунок)

Документирование тестов (3/3)

Прио-ритет

Связанное с тестом требование

Заглавие (суть) теста

Ожидаемый результат по каждому шагу

UG_U 1.12	A	R97	Гале рея	Загрузка файла	**Галерея, загрузка файла, имя со спецсимволами** Приготовления: создать непустой файл с именем #$%^&.jpg 1. Нажать кнопку «Загрузить картинку» 2. Нажать кнопку «Выбрать» 3. Выбрать из списка приготовленный файл 4. Нажать кнопку «ОК» 5. Нажать кнопку «Добавить в галерею»	1. Появляется окно загрузки картинки 2. Появляется диалоговое окно браузера выбора файла для загрузки 3. Имя выбранного файла появляется в поле «Файл» 4. Диалоговое окно файла закрывается, в поле «Файл» появляется полное имя файла 5. Выбранный файл появляется в списке файлов галереи

Идентификатор

Исходные данные, необходимые для выполнения теста

Модуль и подмодуль

Шаги

Зачем нужны тест-кейсы

- «Планирование, и только потом – выполнение!» Тест-кейсы дают нам структурированный системный подход, что снижает вероятность пропуска ошибки.
- Тест-кейсы – хороший способ хранения части проектной информации.
- Написание тест-кейсов – один из способов протестировать проектную документацию ещё до выхода первого билда.
- Наличие тест-кейсов значительно ускоряет регрессионное тестирование.
- Тест-кейсы – прекрасный способ быстро ввести в курс дела новичка или сотрудника, только что подключившегося к проекту.
- Имея тест-кейсы, мы можем в любой момент «вспомнить», что мы делали месяц, полгода, год назад.
- Мы можем обмениваться тест-кейсами (и «чек-листами») между проектами.
- Тест-кейсы позволяют легко отслеживать прогресс (X% тестов выполнено, Y% тестов прошло (завалилось), Z% требований покрыто тестами).

Хороший тест-кейс – каков он?

Тест-кейсы могут быть:

- Специфичными или общими.
- Простыми или сложными.
- Независимыми или связанными друг с другом.
- Позитивными или негативными.

Специфичность или общность?

Оба тест-кейса делают одну и ту же проверку. **Какой из них лучше?**

1.В поле A ввести 10 2.В поле B ввести 15 3.Нажать кнопку «Сложить» 4.Проверить значение в поле C	4. Значение в поле C равно 25

1.Проверить, что программа суммирует два числа корректно	4. Суммирует корректно

Оказывается, оба варианта – плохие. **Почему?**

Специфичность или общность?

- Когда все детали прописаны до мелочей, при повторных выполнениях теста всегда будут выполняться строго одни и те же действия, что снижает вероятность обнаружить ошибку.
- Слишком общий тест-кейс сложно выполнять по многим объективным и субъективным причинам, а потому он вполне может остаться невыполненным.
- Однако интеграционные тесты, как правило, бывают более общими, чем иные. Это связано со спецификой интеграционного тестирования.
- Если в тесте прописано много мелких деталей, возрастает время его создания и поддержки.
- Однако недостаток деталей может усложнить работу новичка.

Специфичность или общность?

Сложение A и B 1. В поле A ввести корректное целое число 2. В поле B ввести корректное целое число 3. Нажать кнопку «Сложить» 4. Проверить значение поля C 5. Повторить шаги 1-4 для значений: 0, максимального и минимального допустимого значений, 1.5, символов и спецсимволов	4. Значение поля C равно сумме A и B

- Здесь мы не привязаны к конкретным значениям.
- Мы знаем, как проверить результат.
- Мы сокращаем время написания и поддержки теста ссылкой на шаги 1-4.
- Мы перечислили значения, представляющие для нас особый интерес.

Простота или сложность?

Где в ниже перечисленном простые тест-кейсы, а где – сложные?

Набор 1:

1. Откройте файл «1.txt». Файл открыт.
2. Введите слово «Дом». Появляется слово «Дом.
3. Сохраните файл. Кнопка «Сохранить» становится неактивной.

Набор 2:

1. В документе размером более 100 Мб создайте таблицу 100x100, в ячейку 50x50 вставьте картинку размером 30 Мб, применив к ней функцию «Авторасположение». Проверьте результат.

Простые тесты оперируют за раз одним объектом.

Простота или сложность?

Каковы преимущества простых тест-кейсов?

- Их легко выполнять.
- Они понятны новичкам.
- Они упрощают диагностику ошибки.
- Они делают наличие ошибки очевидным.

Каковы преимущества сложных тест-кейсов?

- Больше шансов что-то сломать.
- Пользователи, как правило, используют сложные сценарии.
- Программисты сами редко проверяют такие варианты.

Следует постепенно повышать сложность тестов.

Независимость или связанность?

Каковы преимущества независимого самостоятельного тест-кейса?

- Его легко и просто выполнить.
- Такие тесты могут работать даже после краха приложения на других тестах.
- Такие тесты можно группировать любым образом и выполнять в любом порядке.

Каковы преимущества наборов тесно связанных тестов?

- Они имитируют работу реальных пользователей.
- Они удобны для интеграционного тестирования.
- Они удобны для разбиения на части тестов с большим количеством шагов.
- Следующий в наборе тест использует данные и состояние приложения, подготовленные предыдущим.

Промышленным стандартом являются независимые тесты.

Использование сценариев не запрещено, но не следует делать их слишком длинными.

Позитивность или негативность?

Позитивные тесты проверяют, что приложение делает то, на что оно рассчитано (т.е. такие тесты используют корректные данные и условия выполнения).

Негативные тесты проверяют работу приложения в нестандартных условиях (при получении некорректных данных или команд или в при работе в некорректных условиях).

Обе разновидности тестов важны и нужны, однако следует помнить последовательность их разработки и выполнения:

1. Простые позитивные.
2. Простые негативные.
3. Сложные позитивные.
4. Сложные негативные.

Язык написания тестов

Одним из важных правил оформления теста является язык написания.

Рекомендации:

- Используйте активный залог: («open», «paste», «click»). В русском языке используйте безличную форму: «открыть» (вместо «откройте»).
- Описывайте поведение системы: «появляется окно…», «приложение закрывается».
- Используйте простой технический стиль.
- ОБЯЗАТЕЛЬНО указывайте ТОЧНЫЕ названия всех элементов приложения.
- Не объясняйте базовые понятия работы с ОС.

Заключение: хороший тест-кейс

Хороший тест-кейс удовлетворяет следующим критериям:

- Обладает высокой вероятностью обнаружения ошибки.
- Исследует соответствующую («ту, которую надо») область приложения.
- Выполняет какие-то интересные действия.
- Не выполняет ненужных действий.
- Является не слишком простым, но и не слишком сложным.
- Не является избыточным по отношению к другим тестам.
- Делает обнаруженную ошибку очевидной.
- Позволяет легко диагностировать ошибку.

Тестовые сценарии

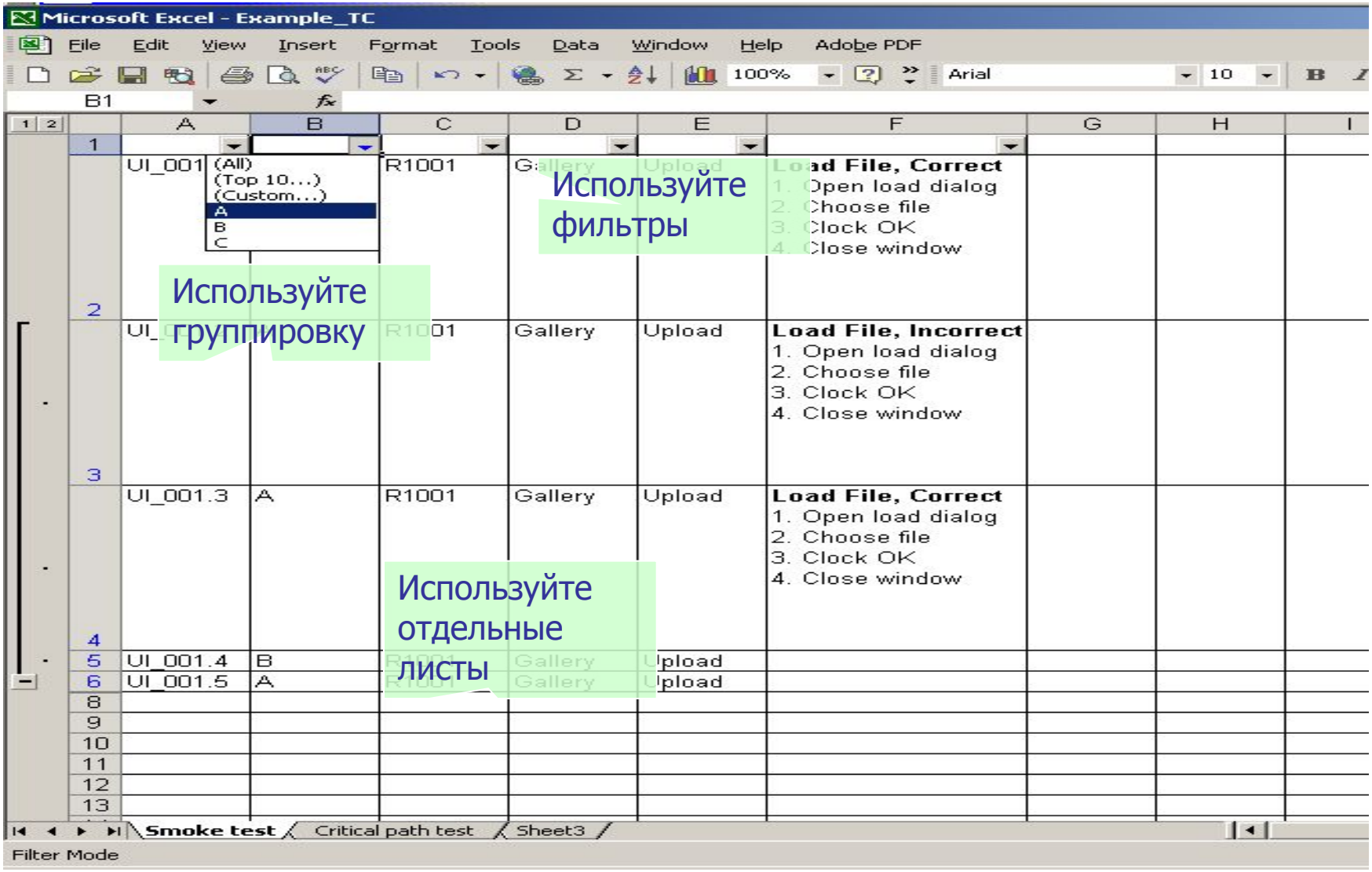
Тестовый сценарий – набор тестов (тест-кейсов), собранных в последовательность для достижения некоторой цели.

Хороший тестовый сценарий всегда следует некоторой логике, например: типичному использованию приложения, удобству тестирования, распределению функций по модулям и т.д.

Тестовые сценарии: рекомендации

- Пишите сценарий для отдельной части приложения.
- Пишите отдельно сценарии для Smoke и Critical Path тестов.
- Постепенно повышайте сложность тестов.
- Организуйте сценарий логично.

Тестовые сценарии: рекомендации



Тестовые сценарии: рекомендации

- Используйте один тест для ОДНОЙ проверки.
- Помните, что заголовки тестов отражают их суть. Правильно формулируйте и оформляйте заголовки.
- Помните о необходимых приготовлениях к тесту. Описывайте их.
- Не повторяйте в нескольких тестах одни и те же шаги.
- Старайтесь избегать похожих тестов (таких, в которых набор шагов и ожидаемых результатов визуально кажется одинаковым).

Техники ускорения написания тестов

- Copy-paste.
- Если по ходу разработки тестов возникают вопросы, пишите их прямо в документ с тестами, помечая красным цветом.
- Используйте т.н. «косметику» (жирный, подчёркнутый, наклонный шрифт, разные цвета т. д.) Это значительно повышает читаемость документа.
- По-максимуму используйте возможности ПО, в котором вы разрабатываете тесты (группировки, фильтры, ссылки и т.д.)
- Если вы пишете тесты в файле, обязательно прописывайте в самом файле историю его изменения.

Шаги разработки тестов

1. **Начинайте как можно раньше, ещё до выхода первого билда.**

Какая информация у нас есть в это время?

А какой нет?

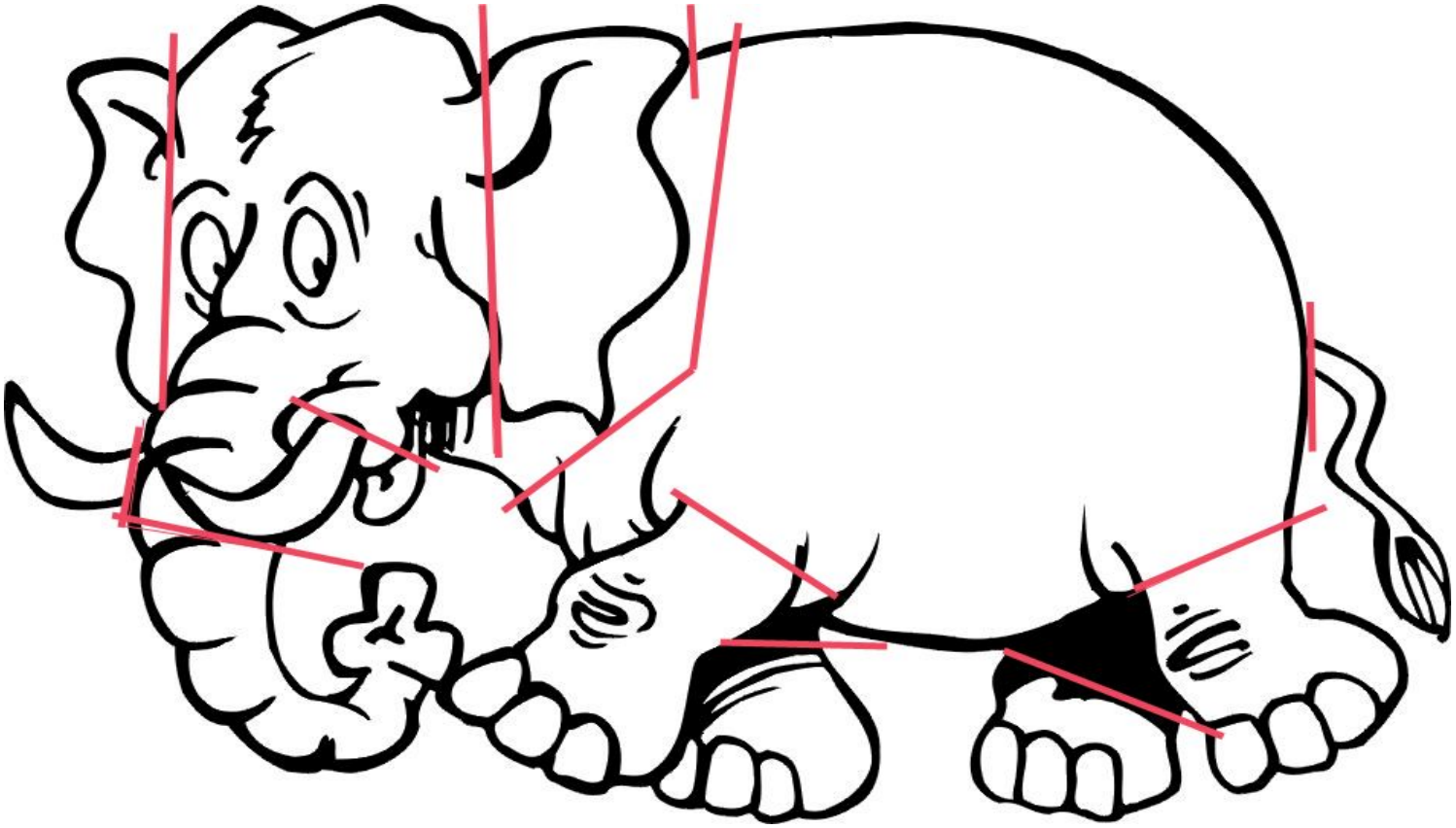
У нас нет работающего приложения.

Но у нас есть документация и представители заказчика.

Смело задавайте вопросы. Помните, что на этой стадии развития проекта исправить ошибку легко и просто, а потом это будет сложно и дорого.

Шаги разработки тестов

2. Разбивайте приложение на отдельные части/модули.



Шаги разработки тестов

3. Для каждой области/модуля пишите чек-лист.

- Так проще проверить, всё ли нужное предусмотрено, нет ли чего лишнего.
- Удобно реорганизовывать наборы тестов.
- Легко увидеть, где можно использовать copy-paste.
- Так можно разделять интеллектуальную и рутинную работу.

Внимание!

- Просто скопированное требование – ЭТО НЕ ТЕСТ!
- Если пишете в Excel/Word – начинайте каждый новый тест в новой строке таблицы.
- Если что-то непонятно – СРАЗУ ЖЕ записывайте вопрос.

Шаги разработки тестов

4. **Пишите вопросы, уточняйте детали, добавляйте «косметику», используйте copy-paste.**

Открытие PDF-файла 1. Нажать кнопку "Открыть". 2. Выбрать файл из диалогового окна. 3. Нажать кнопку "OK" 4. Проверить отображение открытого файла.	1. Появляется диалог выбора файлов. **Какой каталог должен открываться по умолчанию?** 2. Выбранный файл помечается выделением. 3. Диалог выбора файлов закрывается. 4. Файл отображается в рабочей области приложения.
Открытие DJV-файла 1. Нажать кнопку "Открыть". 2. Выбрать файл из диалогового окна. 3. Нажать кнопку "OK" 4. Проверить отображение открытого файла.	1. Появляется диалог выбора файлов. **Какой каталог должен открываться по умолчанию?** 2. Выбранный файл помечается выделением. 3. Диалог выбора файлов закрывается. 4. Файл отображается в рабочей области приложения.

Шаги разработки тестов

5. Получите рецензию коллег-тестировщиков, разработчиков, заказчиков.

Так вы можете получить ответы на вопросы:

- Пропущено ли что-то?
- Есть ли избыточные тесты?
- Легко ли ваши тесты понять?
- Этого ли ожидает заказчик?
- Есть ли в тестах ошибки?

Вы получаете ещё одну точку зрения на ситуацию.

Коллеги могут заметить те ошибки, которые вы пропустили.

У коллег может оказаться та информация, которой не было у вас.

У разработчиков может быть своё мнение по поводу той или иной функциональности.

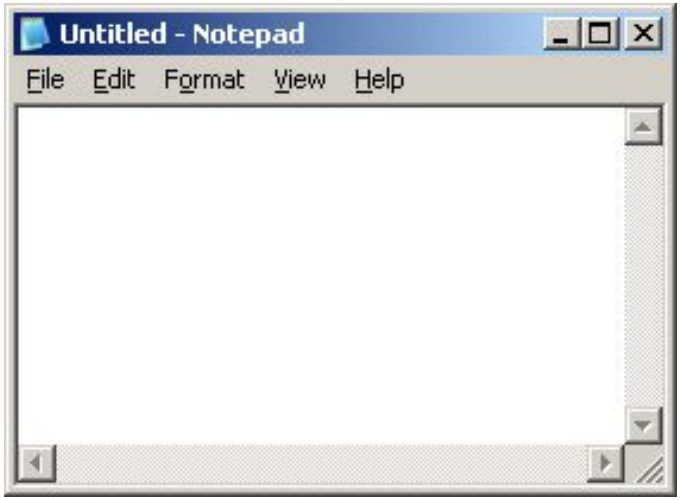
Рецензирование (перепросмотр) хорошо стимулирует повышение качества разработки тестов.

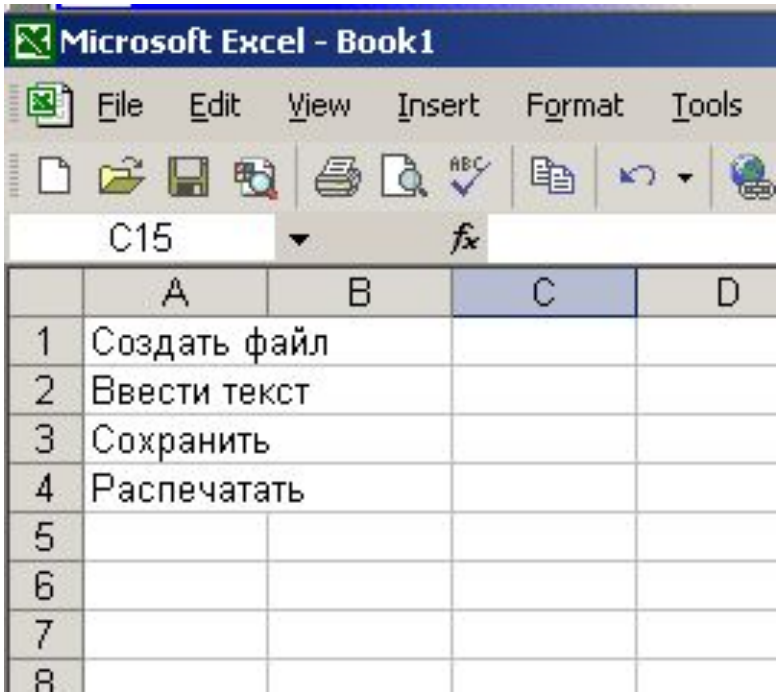
Шаги разработки тестов

6. **Обновляйте тесты, как только обнаружили ошибку или изменилась функциональность.**

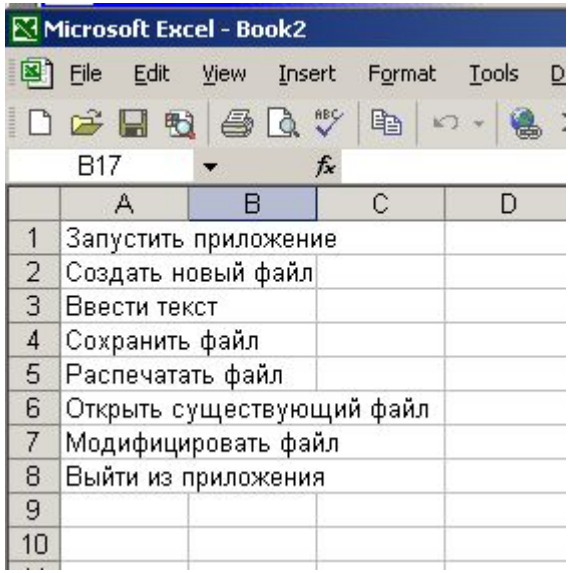
   – Мелкие изменения вносите сразу же, как в этом возникла необходимость.
   – Большие изменения можно вносить в те моменты, когда нагрузка на команду тестировщиков снижается, или когда просто появилось свободное время.

Пример разработки тестов

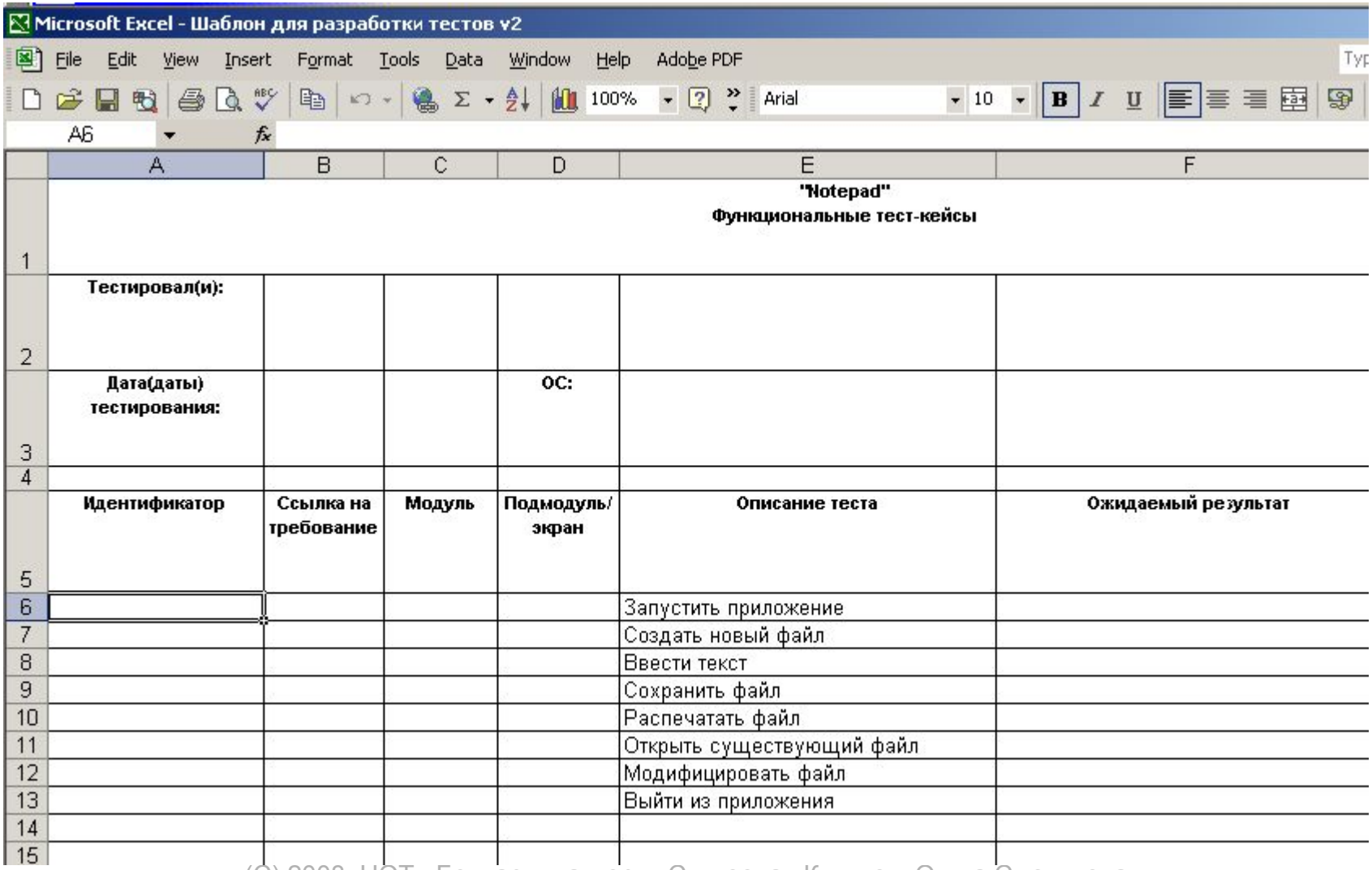




Пример разработки тестов



Пример разработки тестов

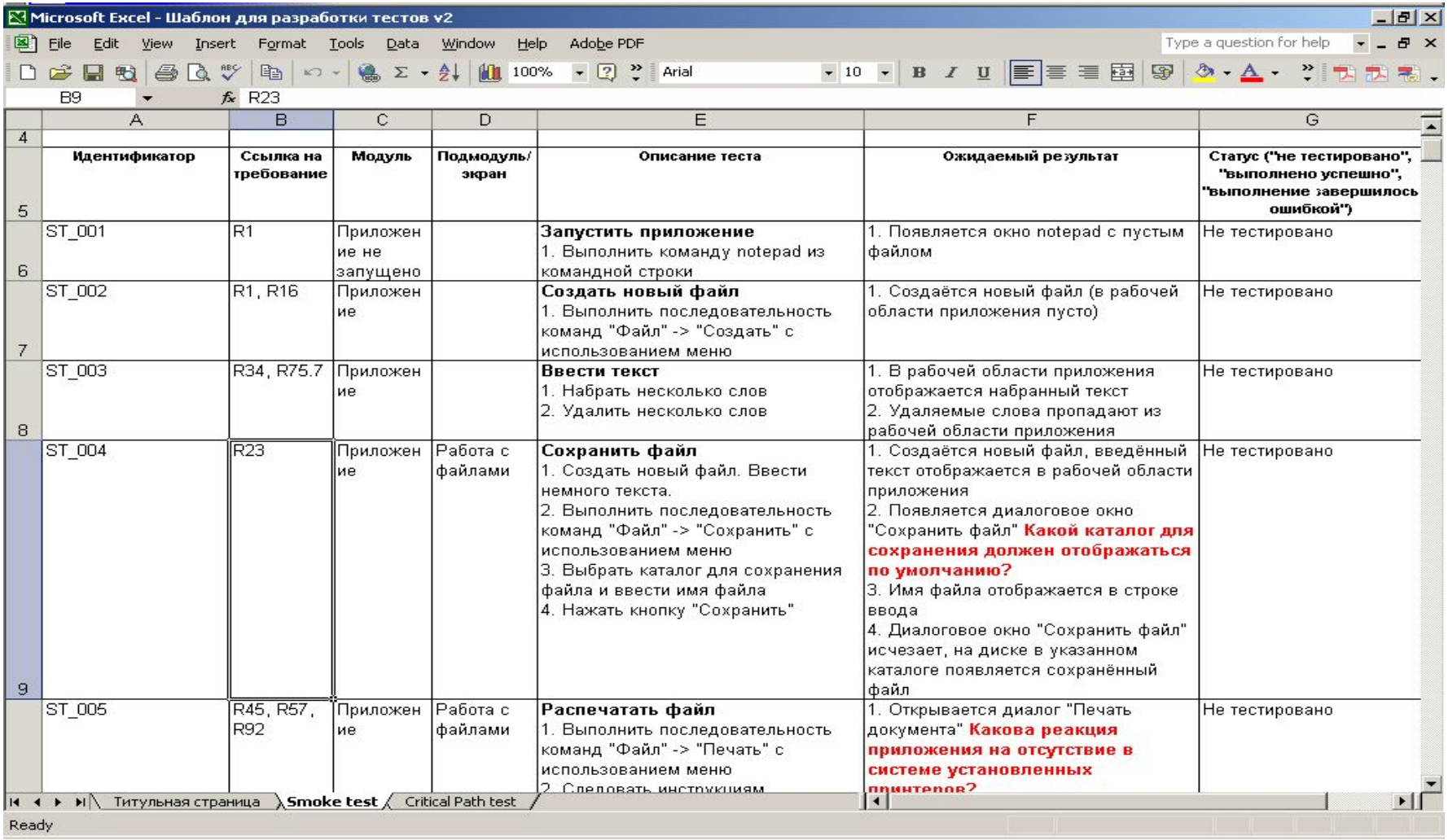

Microsoft Excel - Шаблон для разработки тестов v2

File Edit View Insert Format Tools Data Window Help Adobe PDF

A6

	A	B	C	D	E	F
1					**"Notepad"** **Функциональные тест-кейсы**	
2	**Тестировал(и):**					
3	**Дата(даты) тестирования:**			**ОС:**		
4						
5	**Идентификатор**	**Ссылка на требование**	**Модуль**	**Подмодуль/ экран**	**Описание теста**	**Ожидаемый результат**
6					Запустить приложение	
7					Создать новый файл	
8					Ввести текст	
9					Сохранить файл	
10					Распечатать файл	
11					Открыть существующий файл	
12					Модифицировать файл	
13					Выйти из приложения	
14						
15						

Пример разработки тестов


Microsoft Excel - Шаблон для разработки тестов v2

File Edit View Insert Format Tools Data Window Help Adobe PDF

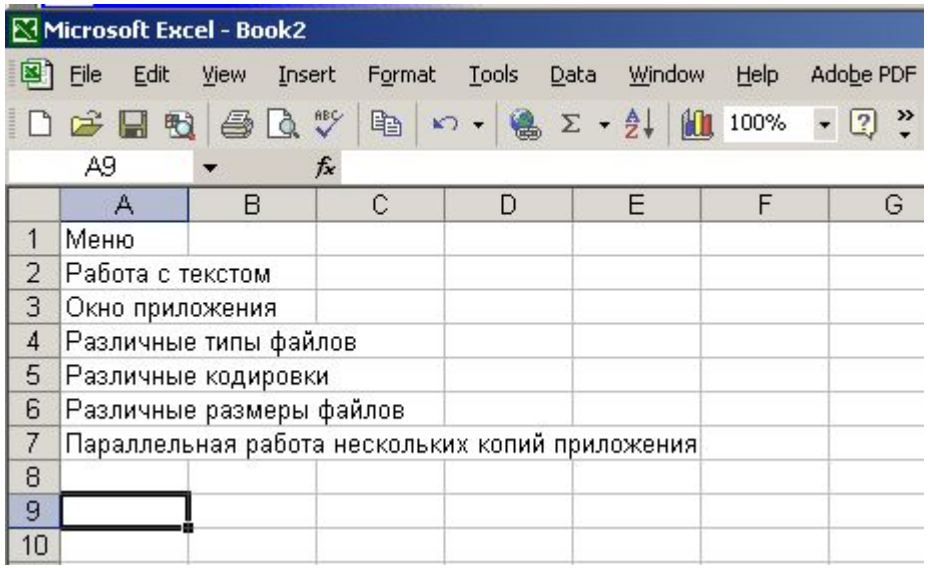
B9 R23

	A	B	C	D	E	F	G
4							
5	**Идентификатор**	**Ссылка на требование**	**Модуль**	**Подмодуль/ экран**	**Описание теста**	**Ожидаемый результат**	**Статус ("не тестировано", "выполнено успешно", "выполнение завершилось ошибкой")**
6	ST_001	R1	Приложение не запущено		**Запустить приложение** 1. Выполнить команду notepad из командной строки	1. Появляется окно notepad с пустым файлом	Не тестировано
7	ST_002	R1, R16	Приложение		**Создать новый файл** 1. Выполнить последовательность команд "Файл" -> "Создать" с использованием меню	1. Создаётся новый файл (в рабочей области приложения пусто)	Не тестировано
8	ST_003	R34, R75.7	Приложение		**Ввести текст** 1. Набрать несколько слов 2. Удалить несколько слов	1. В рабочей области приложения отображается набранный текст 2. Удаляемые слова пропадают из рабочей области приложения	Не тестировано
9	ST_004	R23	Приложение	Работа с файлами	**Сохранить файл** 1. Создать новый файл. Ввести немного текста. 2. Выполнить последовательность команд "Файл" -> "Сохранить" с использованием меню 3. Выбрать каталог для сохранения файла и ввести имя файла 4. Нажать кнопку "Сохранить"	1. Создаётся новый файл, введённый текст отображается в рабочей области приложения 2. Появляется диалоговое окно "Сохранить файл" **Какой каталог для сохранения должен отображаться по умолчанию?** 3. Имя файла отображается в строке ввода 4. Диалоговое окно "Сохранить файл" исчезает, на диске в указанном каталоге появляется сохранённый файл	Не тестировано
	ST_005	R45, R57, R92	Приложение	Работа с файлами	**Распечатать файл** 1. Выполнить последовательность команд "Файл" -> "Печать" с использованием меню 2. Следовать инструкциям	1. Открывается диалог "Печать документа" **Какова реакция приложения на отсутствие в системе установленных принтеров?**	Не тестировано

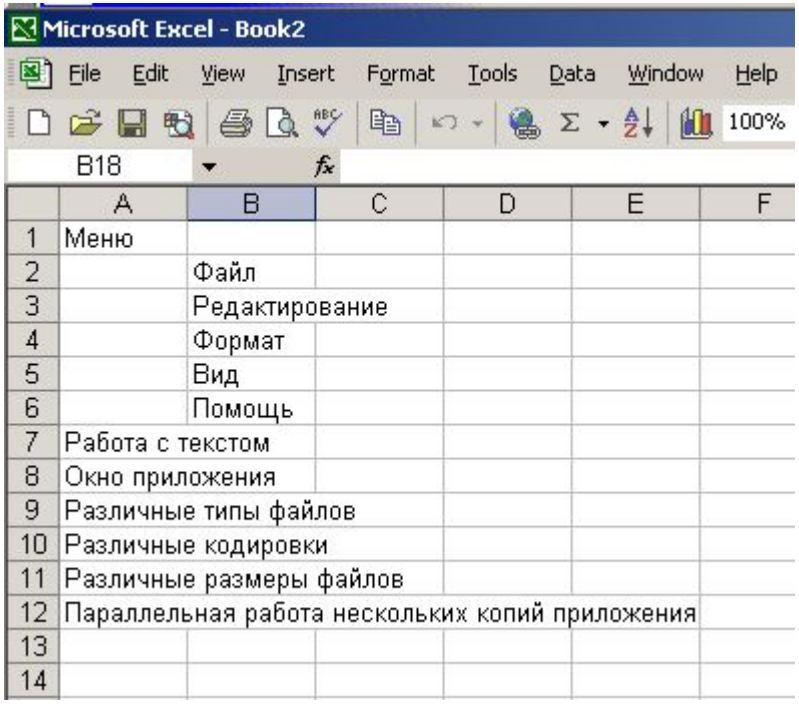
Титульная страница | Smoke test | Critical Path test

Ready

Пример разработки тестов

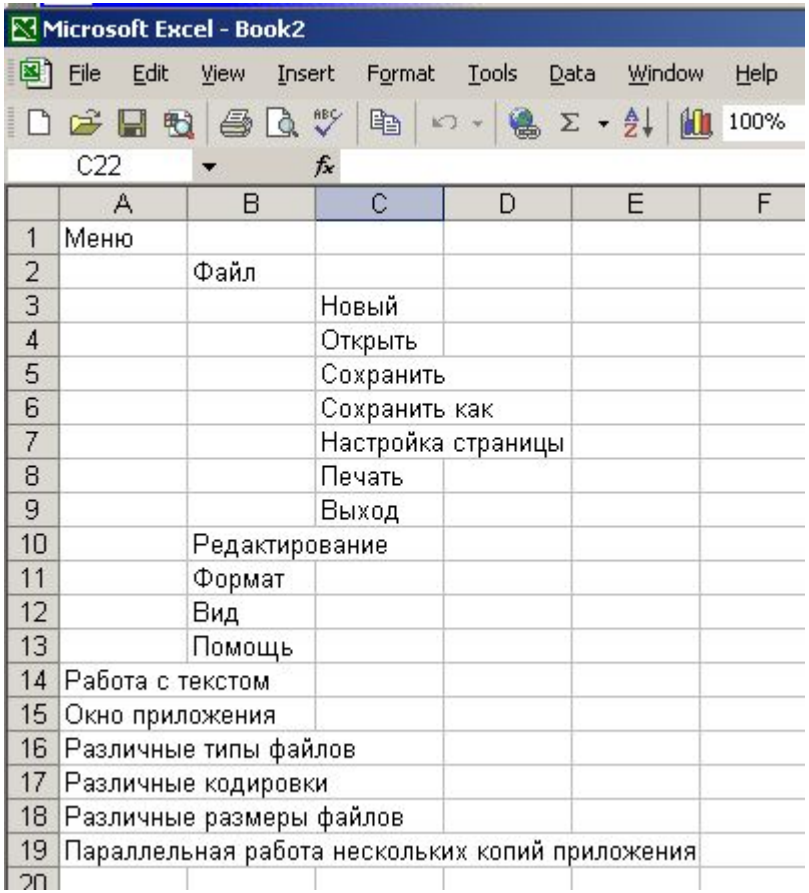


Пример разработки тестов



	A	B	C	D	E	F
1	Меню					
2		Файл				
3		Редактирование				
4		Формат				
5		Вид				
6		Помощь				
7	Работа с текстом					
8	Окно приложения					
9	Различные типы файлов					
10	Различные кодировки					
11	Различные размеры файлов					
12	Параллельная работа нескольких копий приложения					
13						
14						

Пример разработки тестов



	A	B	C	D	E	F
1	Меню					
2		Файл				
3			Новый			
4			Открыть			
5			Сохранить			
6			Сохранить как			
7			Настройка страницы			
8			Печать			
9			Выход			
10		Редактирование				
11		Формат				
12		Вид				
13		Помощь				
14	Работа с текстом					
15	Окно приложения					
16	Различные типы файлов					
17	Различные кодировки					
18	Различные размеры файлов					
19	Параллельная работа нескольких копий приложения					
20						

Практика

Сейчас вам предстоит поработать самостоятельно. Вы будете выполнять практическое задание. Вам надо будет написать тест кейсы для Smoke Test и сформировать чек-лист для Critical Path Test.

Чтобы приблизить наши условия работы к реальным, мы воспроизведём ситуацию прототипирования приложения, когда тест-кейсы приходится иногда писать, ещё не имея требований.

Ваша задача – написать тест-кейсы для стандартного калькулятора, включённого в поставку ОС Windows.