



Лекция 16 Наборы тест-кейсов

Терминология и общие сведения

	Набор тест-кейсов (test case suite ³⁰³ , test suite, test set) — совокупность тест-кейсов, выбранных с некоторой общей целью или по некоторому общему признаку. Иногда в такой совокупности результаты завершения одного тест-кейса становятся входным состоянием приложения для следующего тест-кейса.
	Внимание! Из-за особенностей перевода очень часто вместо «набор тест-кейсов» говорят «тестовый сценарий». Формально это можно считать ошибкой, но это явление приобрело настолько широкое распространение, что стало вариантом нормы.

Как мы только что убедились на примере множества отдельных тест-кейсов, крайне неудобно (более того, это ошибка!) каждый раз писать в каждом тест-кейсе одни и те же приготовления и повторять одни и те же начальные шаги.

Намного удобнее объединить несколько тест-кейсов в набор или последовательность. И здесь мы приходим к классификации наборов тест-кейсов.

В общем случае наборы тест-кейсов можно разделить на свободные (порядок выполнения тест-кейсов не важен) и последовательные (порядок выполнения тест-кейсов важен).


Преимущества свободных наборов:

- Тест-кейсы можно выполнять в любом удобном порядке, а также создавать «наборы внутри наборов».
- Если какой-то тест-кейс завершился ошибкой, это не повлияет на возможность выполнения других тест-кейсов.

Преимущества последовательных наборов:

- Каждый следующий в наборе тест-кейс в качестве входного состояния приложения получает результат работы предыдущего тест-кейса, что позволяет сильно сократить количество шагов в отдельных тест-кейсах.
- Длинные последовательности действий куда лучше имитируют работу реальных пользователей, чем отдельные «точечные» воздействия на приложение.

Пользовательские сценарии (сценарии использования)

	В данном случае речь НЕ идёт о use cases (вариантах использования), представляющих собой форму требований ⁽³⁶⁾ . Пользовательские сценарии как техника тестирования куда менее формализованы, хотя и могут строиться на основе вариантов использования.
-------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

К отдельному подвиду последовательных наборов тест-кейсов (или даже неоформленных идей тест-кейсов, таких, как пункты чек-листа) можно отнести пользовательские сценарии³⁰⁴ (или сценарии использования), представляющие собой цепочки действий, выполняемых пользователем в определённой ситуации для достижения определённой цели.

³⁰³ **Test case suite (test suite, test set).** A set of several test cases for a component or system under test, where the post condition of one test is often used as the precondition for the next one. [ISTQB Glossary]

³⁰⁴ A scenario is a hypothetical story, used to help a person think through a complex problem or system. [Cem Kaner, «An Introduction to Scenario Testing», <http://www.kaner.com/pdfs/ScenarioIntroVer4.pdf>]

Поясним это сначала на примере, не относящемся к «Конвертеру файлов». Допустим, пользователь хочет распечатать табличку на дверь кабинета с текстом «Идёт работа, не стучать!» Для этого ему нужно:

- 1) Запустить текстовый редактор.
- 2) Создать новый документ (если редактор не делает это самостоятельно).
- 3) Набрать в документе текст.
- 4) Отформатировать текст должным образом.
- 5) Отправить документ на печать.
- 6) Сохранить документ (спорно, но допустим).
- 7) Закрыть текстовый редактор.

Вот мы и получили пользовательский сценарий, пункты которого могут стать основой для шагов тест-кейса или целого набора отдельных тест-кейсов.

Сценарии могут быть достаточно длинными и сложными, могут содержать внутри себя циклы и условные ветвления, но при всём этом они обладают рядом весьма интересных преимуществ:

- Сценарии показывают реальные и понятные примеры использования продукта (в отличие от обширных чек-листов, где смысл отдельных пунктов может теряться).
- Сценарии понятны конечным пользователям и хорошо подходят для обсуждения и совместного улучшения.
- Сценарии и их части легче оценивать с точки зрения важности, чем отдельные пункты (особенно низкоуровневых) требований.
- Сценарии отлично показывают недоработки в требованиях (если становится непонятно, что делать в том или ином пункте сценария, — с требованиями явно что-то не то).
- В предельном случае (нехватка времени и прочие форс-мажоры) сценарии можно даже не прописывать подробно, а просто именовать — и само наименование уже подскажет опытному специалисту, что делать.

Последний пункт проиллюстрируем на примере. Классифицируем потенциальных пользователей нашего приложения (напомним, что в нашем случае «пользователь» — это администратор, настраивающий работу приложения) по степени квалификации и склонности к экспериментам, а затем дадим каждому «виду пользователя» запоминающееся имя.

Таблица 2.4.а — Классификация пользователей

	Низкая квалификация	Высокая квалификация
Не склонен к экспериментам	«Осторожный»	«Консервативный»
Склонен к экспериментам	«Отчаянный»	«Изощённый»

Согласитесь, уже на этой стадии не составляет труда представить себе отличия в логике работы с приложением, например, «консервативного» и «отчаянного» пользователей.

Но мы пойдём дальше и озаглавим для них сами сценарии, например, в ситуациях, когда такой пользователь позитивно и негативно относится к идее внедрения нашего приложения:

Таблица 2.4.b — Сценарии поведения на основе классификации пользователей

	«Осторожный»	«Консервативный»	«Отчаянный»	«Изощённый»
Позитивно	«А можно вот так?»	«Начнём с инструкции!»	«Гляньте, что я придумал!»	«Я всё оптимизирую!»
Негативно	«Я ничего не понимаю.»	«У вас вот здесь несоответствие...»	«Всё равно поломаю!»	«А я говорил!»

Проявив даже самую малость воображения, можно представить, что и как будет происходить в каждой из получившихся восьми ситуаций. Причём на создание пары таких таблиц уходит всего несколько минут, а эффект от их использования на порядки превосходит бездумное «кликание по кнопкам в надежде найти баг».



Куда более полное и техничное объяснение того, что такое сценарное тестирование, как его применять и выполнять должным образом, можно прочесть в статье Сэма Канера «An Introduction to Scenario Testing»³⁰⁵.

Подробная классификация наборов тест-кейсов



Представленный здесь материал сложно отнести к категории «для начинающих» (и вы можете сразу перейти к пункту «Принципы построения наборов тест-кейсов»^[145]). Но если вам любопытно, взгляните на подробную классификацию, которая представлена ниже.

Подробная классификация наборов тест-кейсов может быть выражена следующей таблицей.

Таблица 2.4.c — Подробная классификация наборов тест-кейсов

		По изолированности тест-кейсов друг от друга	
		Изолированные	Обобщённые
По образованию тест-кейсами строгой последовательности	Свободные	Изолированные свободные	Обобщённые свободные
	Последовательные	Изолированные последовательные	Обобщённые последовательные

- Набор изолированных свободных тест-кейсов (рисунок 2.4.g): действия из раздела «приготовления» нужно повторить перед каждым тест-кейсом, а сами тест-кейсы можно выполнять в любом порядке.
- Набор обобщённых свободных тест-кейсов (рисунок 2.4.h): действия из раздела «приготовления» нужно выполнить один раз (а потом просто выполнять тест-кейсы), а сами тест-кейсы можно выполнять в любом порядке.
- Набор изолированных последовательных тест-кейсов (рисунок 2.4.i): действия из раздела «приготовления» нужно повторить перед каждым тест-кейсом, а сами тест-кейсы нужно выполнять в строго определённом порядке.
- Набор обобщённых последовательных тест-кейсов (рисунок 2.4.j): действия из раздела «приготовления» нужно выполнить один раз (а потом просто выполнять тест-кейсы), а сами тест-кейсы нужно выполнять в строго определённом порядке.

³⁰⁵ «An Introduction to Scenario Testing», Cem Kaner [<http://www.kaner.com/pdfs/ScenarioIntroVer4.pdf>]

Главное преимущество изолированности: каждый тест-кейс выполняется в «чистой среде», на него не влияют результаты работы предыдущих тест-кейсов.

Главное преимущество обобщённости: приготовления не нужно повторять (экономия времени).

Главное преимущество последовательности: осязаемое сокращение шагов в каждом тест-кейсе, т.к. результат выполнения предыдущего тест-кейса является начальной ситуацией для следующего.

Главное преимущество свободы: возможность выполнять тест-кейсы в любом порядке, а также то, что при провале некоего тест-кейса (приложение не пришло в ожидаемое состояние) остальные тест-кейсы по-прежнему можно выполнять.

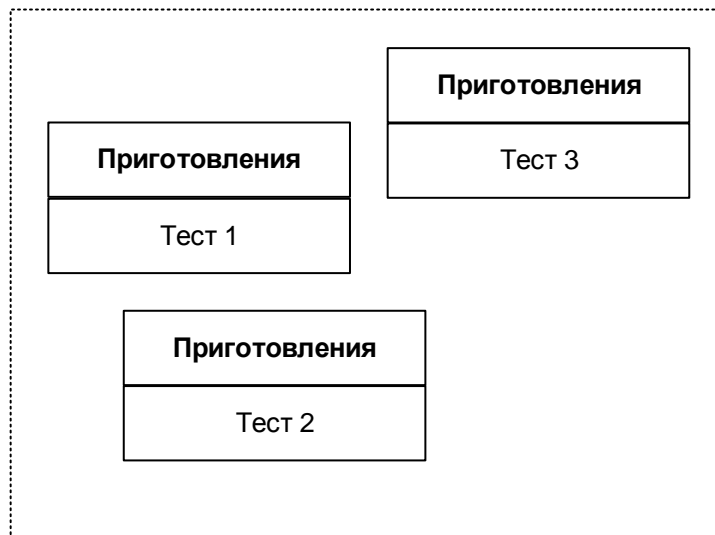


Рисунок 2.4.g — Набор изолированных свободных тест-кейсов

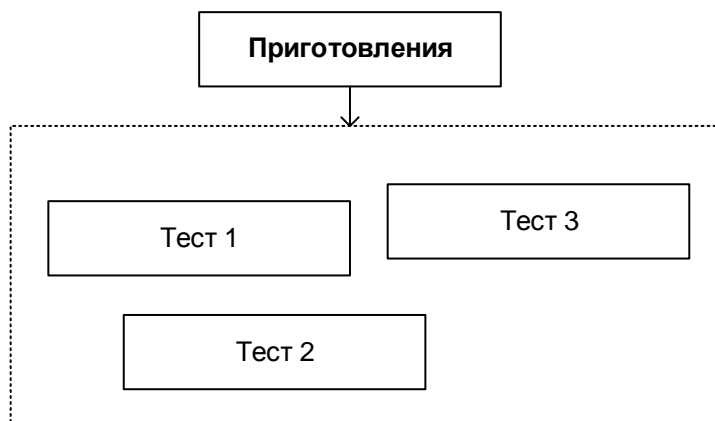


Рисунок 2.4.h — Набор обобщённых свободных тест-кейсов

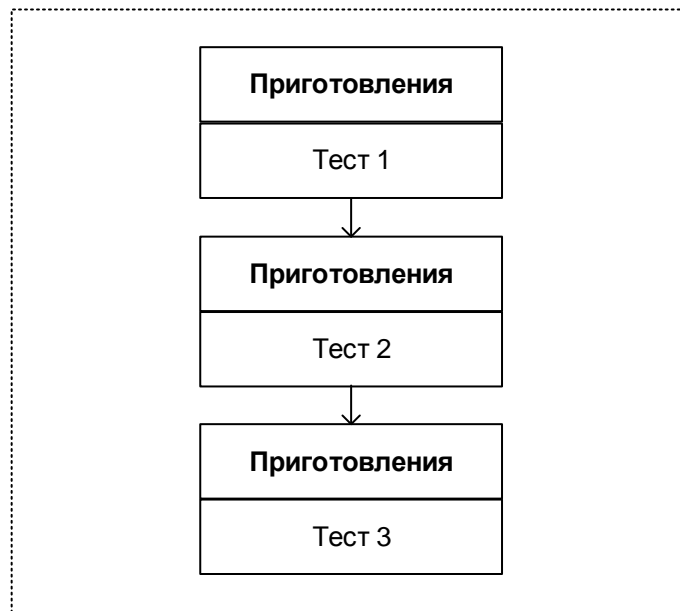


Рисунок 2.4.i — Набор изолированных последовательных тест-кейсов

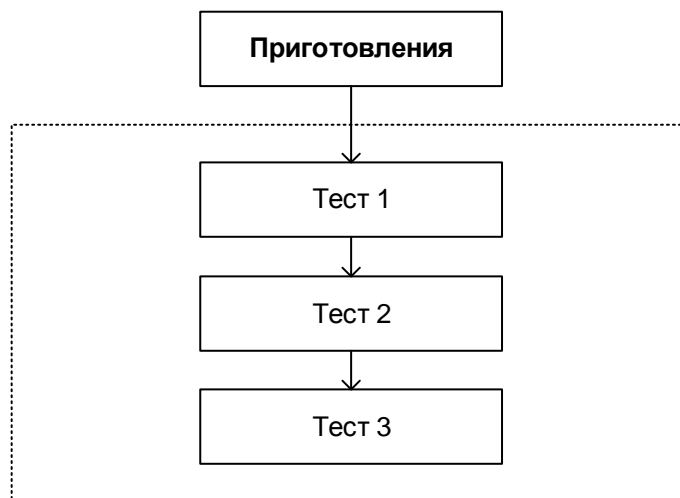


Рисунок 2.4.j — Набор обобщённых последовательных тест-кейсов

Принципы построения наборов тест-кейсов

Теперь — о самом главном: как формировать наборы тест-кейсов. Правильный ответ звучит очень кратко: логично. И это не шутка. Единственная задача наборов — повысить эффективность тестирования за счёт ускорения и упрощения выполнения тест-кейсов, увеличения глубины исследования некоей области приложения или функциональности, следования типичным пользовательским сценариям⁽¹⁴¹⁾ или удобной последовательности выполнения тест-кейсов и т.д.

Набор тест-кейсов всегда создаётся с какой-то целью, на основе какой-то логики, и по этим же принципам в набор включаются тесты, обладающие подходящими свойствами.

Если же говорить о наиболее типичных подходах к составлению наборов тест-кейсов, то можно обозначить следующее:

- На основе чек-листов. Посмотрите внимательно на примеры чек-листов⁽¹¹¹⁾, которые мы разработали в соответствующем разделе⁽¹¹⁰⁾: каждый пункт чек-листа может превратиться в несколько тест-кейсов — и вот мы получаем готовый набор.

- На основе разбиения приложения на модули и подмодули⁽¹²⁰⁾. Для каждого модуля (или его отдельных подмодулей) можно составить свой набор тест-кейсов.
- По принципу проверки самых важных, менее важных и всех остальных функций приложения (именно по этому принципу мы составляли примеры чек-листов⁽¹¹¹⁾).
- По принципу группировки тест-кейсов для проверки некоего уровня требований или типа требований⁽³⁶⁾, группы требований или отдельного требования.
- По принципу частоты обнаружения тест-кейсами дефектов в приложении (например, мы видим, что некоторые тест-кейсы раз за разом завершаются неудачей, значит, мы можем объединить их в набор, условно названный «проблемные места в приложении»).
- По архитектурному принципу (см. «многоуровневая архитектура»¹⁴⁹ самостоятельно): наборы для проверки пользовательского интерфейса и всего уровня представления, для проверки уровня бизнес-логики, для проверки уровня данных.
- По области внутренней работы приложения, например: «тест-кейсы, затрагивающие работу с базой данных», «тест-кейсы, затрагивающие работу с файловой системой», «тест-кейсы, затрагивающие работу с сетью», и т.д.
- По видам тестирования (см. главу «Подробная классификация тестирования»⁽⁶⁴⁾).

Не нужно заучивать этот список. Это всего лишь примеры — грубо говоря, «первое, что приходит в голову». Важен принцип: если вы видите, что выполнение некоторых тест-кейсов в виде набора принесёт вам пользу, создавайте такой набор.

Примечание: без хороших инструментальных средств управления тест-кейсами работать с наборами тест-кейсов крайне тяжело, т.к. приходится самостоятельно следить за приготовлениями, «недостающими шагами», изолированностью или обобщённостью, свободностью или последовательностью и т.д.

2.4.7. Логика создания эффективных проверок

Теперь, когда мы рассмотрели принципы построения чек-листов⁽¹¹⁰⁾ и оформления тест-кейсов⁽¹¹⁹⁾, свойства качественных тест-кейсов⁽¹³¹⁾, а также принципы объединения тест-кейсов в наборы⁽¹⁴⁵⁾, настало время перейти к самой сложной, «философской» части, в которой мы будем рассуждать уже не о том, что и как писать, а о том, как думать.

Ранее уже было сказано: если у тест-кейса не указаны входные данные, условия выполнения и ожидаемые результаты, и/или не ясна цель тест-кейса — это плохой тест-кейс. И здесь во главе угла стоит **цель**. Если мы чётко понимаем, что и зачем мы делаем, мы или быстро находим всю остальную недостающую информацию, или столь же быстро формулируем правильные вопросы и адресуем их правильным людям.

Вся суть работы тестировщика в конечном итоге направлена на повышение качества (процессов, продуктов и т.д.) Но что такое качество? Да, существует сухое официальное определение³⁰⁶, но даже там сказано про «user/customer needs and expectations» (потребности и ожидания пользователя/заказчика).

И здесь проявляется главная мысль: **качество — это некая ценность для конечного пользователя** (заказчика). Человек в любом случае платит за использование продукта — деньгами, своим временем, какими-то усилиями (даже если вы не получаете эту «оплату», человек вполне обоснованно считает, что что-то уже на вас потратил, и он прав). Но получает ли он то, на что рассчитывал (предположим, что его ожидания — здравы и реалистичны)?

Если мы подходим к тестированию формально, мы рискуем получить продукт, который по документам (метрикам и т.д.) выглядит идеально, но в реальности никому не нужен.

Поскольку практически любой современный программный продукт представляет собой непростую систему, среди широкого множества её свойств и функций объективно есть самые важные, менее важные и совсем незначительные по важности для пользователей.

Если усилия тестировщиков будут сконцентрированы на первой и второй категории (самом важном и чуть менее важном), наши шансы создать приложение, удовлетворяющее заказчика, резко увеличиваются.

Есть простая логика:

- Тесты ищут ошибки.
- Но все ошибки найти невозможно.
- Значит, наша задача — найти максимум ВАЖНЫХ ошибок за имеющееся время.

Под важными ошибками здесь мы понимаем такие, которые приводят к нарушению важных для пользователя функций или свойств продукта. Функции и свойства разделены не случайно — безопасность, производительность, удобство и т.д. не относятся к функциям, но играют ничуть не менее важную роль в формировании удовлетворённости заказчика и конечных пользователей.

Ситуация усугубляется следующими фактами:

- в силу множества экономических и технических причин мы не можем выполнить «все тесты, что придут нам в голову» (да ещё и многократно) — приходится тщательно выбирать, что и как мы будем тестировать, принимая во внимание только что упомянутую мысль: качество — это некая ценность для конечного пользователя (заказчика);

³⁰⁶ **Quality.** The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations. [ISTQB Glossary]

- никогда в реальной жизни (как бы мы ни старались) мы не получим «совершенного и идеального набора требований» — там всегда будет некоторое количество недоработок, и это тоже надо принимать во внимание.

Однако существует достаточно простой алгоритм, позволяющий нам создавать эффективные проверки даже в таких условиях. Приступая к продумыванию чек-листа, тест-кейса или набора тест-кейсов, задайте себе следующие вопросы и получите чёткие ответы:

- Что перед вами? Если вы не понимаете, что вам предстоит тестировать, вы не уйдёте дальше бездумных формальных проверок.
- Кому и зачем оно нужно (и насколько это важно)? Ответ на этот вопрос позволит вам быстро придумать несколько характерных сценариев использования^[141] того, что вы собираетесь тестировать.
- Как оно обычно используется? Это уже детализация сценариев и источник идей для позитивного тестирования^[77] (их удобно оформить в виде чек-листа).
- Как оно может сломаться, т.е. начать работать неверно? Это также детализация сценариев использования, но уже в контексте негативного тестирования^[77] (их тоже удобно оформить в виде чек-листа).

К этому алгоритму можно добавить ещё небольшой перечень универсальных рекомендаций, которые позволят вам проводить тестирование лучше:

- Начинайте как можно раньше — уже с момента появления первых требований можно заниматься их тестированием и улучшением, можно писать чек-листы и тест-кейсы, можно уточнять план тестирования, готовить тестовое окружение и т.д.
- Если вам предстоит тестировать что-то большое и сложное, разбивайте его на модули и подмодули, функциональность подвергайте функциональной декомпозиции³⁰⁷ — т.е. добейтесь такого уровня детализации, при котором вы можете без труда удержать в голове всю информацию об объекте тестирования.
- Обязательно пишите чек-листы. Если вам кажется, что вы сможете запомнить все идеи и потом легко их воспроизвести, вы ошибаетесь. Исключений не бывает.
- По мере создания чек-листов, тест-кейсов и т.д. прямо в текст вписывайте возникающие вопросы. Когда вопросов накопится достаточно, соберите их отдельно, уточните формулировки и обратитесь к тому, кто может дать ответы.
- Если используемое вами инструментальное средство позволяет использовать косметическое оформление текста — используйте (так текст будет лучше читаться), но старайтесь следовать общепринятым традициям и не раскрашивать каждое второе слово в свой цвет, шрифт, размер и т.д.
- Используйте технику беглого просмотра^[46] для получения отзыва от коллег и улучшения созданного вами документа.
- Планируйте время на улучшение тест-кейсов (исправление ошибок, доработку по факту изменения требований и т.д.).
- Начинайте проработку (и выполнение) тест-кейсов с простых позитивных проверок наиболее важной функциональности. Затем постепенно повышайте сложность проверок, помня не только о позитивных^[77], но и о негативных^[77] проверках.

³⁰⁷ «Functional decomposition», Wikipedia [http://en.wikipedia.org/wiki/Functional_decomposition]

- Помните, что в основе тестирования лежит цель. Если вы не можете быстро и просто сформулировать цель созданного вами тест-кейса, вы создали плохой тест-кейс.
- Избегайте избыточных, дублирующих друг друга тест-кейсов. Минимизировать их количество вам помогут техники классов эквивалентности⁽⁸⁹⁾, граничных условий⁽⁹⁰⁾, доменного тестирования⁽⁹⁰⁾.
- Если показательность⁽¹³⁵⁾ тест-кейса можно увеличить, при этом не сильно изменив его сложность и не отклонившись от исходной цели, сделайте это.
- Помните, что очень многие тест-кейсы требуют отдельной подготовки, которую нужно описать в соответствующем поле тест-кейса.
- Несколько позитивных тест-кейсов⁽⁷⁷⁾ можно безбоязненно объединять, но объединение негативных тест-кейсов⁽⁷⁷⁾ почти всегда запрещено.
- Подумайте, как можно оптимизировать созданный вами тест-кейс (набор тест-кейсов и т.д.) так, чтобы снизить трудозатраты на его выполнение.
- Перед тем как отправлять финальную версию созданного вами документа, ещё раз перечитайте написанное (в доброй половине случаев найдёте опечатку или иную недоработку).



Задание 2.4.е: дополните этот список идеями, которые вы почерпнули из других книг, статей и т.д.

Пример реализации логики создания эффективных проверок

Ранее мы составили подробный чек-лист⁽¹¹⁰⁾ для тестирования нашего «Конвертера файлов»⁽⁵⁵⁾. Давайте посмотрим на него критически и подумаем: что можно сократить, чем мы в итоге пожертвуем и какой выигрыш получим.

Прежде чем мы начнём оптимизировать чек-лист, важно отметить, что решение о том, что важно и что неважно, стоит принимать на основе ранжирования требований по важности, а также согласовывать с заказчиком.

Что для пользователя САМОЕ важное? Ради чего создавалось приложение? Чтобы конвертировать файлы. Принимая во внимание тот факт, что настройку приложения будет выполнять квалифицированный технический специалист, мы можем даже «отодвинуть на второй план» реакцию приложения на ошибки стадии запуска и завершения.

И на первое место выходит:

- Обработка файлов, разные форматы, кодировки и размеры:

Таблица 2.4.d — Форматы, кодировки и размеры файлов

		Форматы входных файлов		
		ТХТ	HTML	MD
Кодировки входных файлов	WIN1251	100 КБ	50 МБ	10 МБ
	CP866	10 МБ	100 КБ	50 МБ
	KOI8R	50 МБ	10 МБ	100 КБ
	Любая	0 байт		
	Любая	50 МБ + 1 Б	50 МБ + 1 Б	50 МБ + 1 Б
	-	Любой недопустимый формат		
	Любая	Повреждения в допустимом формате		

Можем ли мы как-то ускорить выполнение этих проверок (ведь их много)? Можем. И у нас даже есть два взаимодополняющих инструмента:

- дальнейшая классификация по важности;
- автоматизация тестирования.

Сначала поделим таблицу на две — чуть более важное и чуть менее важное. В «чуть более важное» попадает:

Таблица 2.4.е — Форматы, кодировки и размеры файлов

		Форматы входных файлов		
		ТХТ	HTML	MD
Кодировки входных файлов	WIN1251	100 КБ	50 МБ	10 МБ
	CP866	10 МБ	100 КБ	50 МБ
	KOI8R	50 МБ	10 МБ	100 КБ

Подготовим 18 файлов — 9 исходных + 9 сконвертированных (в любом текстовом редакторе с функцией конвертации кодировок), чтобы в дальнейшем сравнивать работу нашего приложения с эталоном.

Для «чуть менее важного» осталось:

- Файл размером 0 байт (объективно для него не важна такая характеристика, как «кодировка»). *Подготовим один файл размером 0 байт.*
- Файл размером 50 МБ + 1 Б (для него тоже не важна кодировка). *Подготовим один файл размером 52'428'801 байт.*
- Любой недопустимый формат:
 - По расширению (файл с расширением, отличным от .txt, .html, .md). *Берём любой произвольный файл, например картинку (размер от 1 до 50 МБ, расширение .jpg).*
 - По внутреннему содержанию (например, .jpg переименовали в .txt). *Копии файла из предыдущего пункта даём расширение .txt.*
- Повреждения в допустимом формате. *Вычёркиваем. Вообще. Даже крайне сложные дорогие редакторы далеко не всегда способны восстановить повреждённые файлы своих форматов, наше же приложение — это просто миниатюрная утилита конвертации кодировок, и не стоит ждать от неё возможностей профессионального инструментального средства восстановления данных.*

Что мы получили в итоге? Нам нужно подготовить следующие 22 файла (поскольку у файлов всё равно есть имена, усилим этот набор тестовых данных, представив в именах файлов символы латиницы, кириллицы и спецсимволы).

Таблица 2.4.f — Итоговый набор файлов для тестирования приложения

№	Имя	Кодировка	Размер
1	«Мелкий» файл в WIN1251.txt	WIN1251	100 КБ
2	«Средний» файл в CP866.txt	CP866	10 МБ
3	«Крупный» файл в KOI8R.txt	KOI8R	50 МБ
4	«Крупный» файл в win-1251.html	WIN1251	50 МБ
5	«Мелкий» файл в cp-866.html	CP866	100 КБ
6	«Средний» файл в koi8-r.html	KOI8R	10 МБ
7	«Средний» файл в WIN_1251.md	WIN1251	10 МБ
8	«Крупный» файл в CP_866.md	CP866	50 МБ
9	«Мелкий» файл в KOI8_R.md	KOI8R	100 КБ
10	«Мелкий» эталон WIN1251.txt	UTF8	100 КБ
11	«Средний» эталон CP866.txt	UTF8	10 МБ
12	«Крупный» эталон KOI8R.txt	UTF8	50 МБ
13	«Крупный» эталон в win-1251.html	UTF8	50 МБ
14	«Мелкий» эталон в cp-866.html	UTF8	100 КБ
15	«Средний» эталон в koi8-r.html	UTF8	10 МБ
16	«Средний» эталон в WIN_1251.md	UTF8	10 МБ
17	«Крупный» эталон в CP_866.md	UTF8	50 МБ
18	«Мелкий» эталон в KOI8_R.md	UTF8	100 КБ
19	Пустой файл.md	-	0 Б
20	Слишком большой файл.txt	-	52'428'801 Б
21	Картинка.jpg	-	~ 1 МБ
22	Картинка в виде TXT.txt	-	~ 1 МБ

И только что мы упомянули автоматизацию как способ ускорения выполнения тест-кейсов. В данном случае мы можем обойтись самыми тривиальными командными файлами. В приложении «Командные файлы для Windows и Linux, автоматизирующие выполнение дымового тестирования»⁽²⁷⁹⁾ приведены скрипты, полностью автоматизирующие выполнение всего уровня дымового тестирования⁽⁷⁴⁾ над представленным выше набором из 22 файлов.



Задание 2.4.f: доработайте представленные в приложении⁽²⁷⁹⁾ командные файлы так, чтобы их выполнение заодно проверяло работу тестируемого приложения с пробелами, кириллическими символами и спецсимволами в путях к входному каталогу, выходному каталогу и файлу журнала. Оптимизируйте получившиеся командные файлы так, чтобы избежать многократного дублирования кода.

Если снова вернуться к чек-листу, то оказывается, что мы уже подготовили проверки для всего уровня дымового тестирования⁽⁷⁴⁾ и части уровня тестирования критического пути⁽⁷⁵⁾.

Продолжим оптимизацию. Большинство проверок не представляет особой сложности, и мы разберёмся с ними по ходу дела, но есть в чек-листе пункт, вызывающий особую тревогу: производительность.

Тестирование и оптимизация производительности⁽⁸⁶⁾ — это отдельный вид тестирования со своими достаточно непростыми правилами и подходами, к тому же разделяющийся на несколько подвидов. Нужно ли оно нам в нашем приложении? Заказчик в АК-1.1 определил минимальную производительность приложения как способность обрабатывать входные данные со скоростью не менее 5 МБ/сек. Грубые эксперименты на указанном в АК-1.1 оборудовании показывают, что даже куда более сложные операции (например, архивирование видеофайла с максимальной степенью сжатия) выполняются быстрее (пусть и ненамного). Вывод? Вычёркиваем. Вероятность встретить здесь проблему ничтожно мала, а соответству-

ющее тестирование требует ощутимых затрат сил и времени, а также наличия соответствующих специалистов.

Вернёмся к чек-листу:

- Конфигурирование и запуск:
 - ~~С верными параметрами:~~
 - Значения SOURCE_DIR, DESTINATION_DIR, LOG_FILE_NAME указаны и содержат пробелы и кириллические символы (повторить для форматов путей в Windows и *nix файловых системах, обратить внимание на имена логических дисков и разделители имён каталогов ("/" и "\")). **(Уже учтено при автоматизации проверки работы приложения с 22 файлами.)**
 - Значение LOG_FILE_NAME не указано. **(Объединить с проверкой ведения самого файла журнала.)**
 - Без параметров.
 - С недостаточным количеством параметров.
 - С неверными параметрами:
 - Недопустимый путь SOURCE_DIR.
 - Недопустимый путь DESTINATION_DIR.
 - Недопустимое имя LOG_FILE_NAME.
 - DESTINATION_DIR находится внутри SOURCE_DIR.
 - Значения DESTINATION_DIR и SOURCE_DIR совпадают.
- Обработка файлов:
 - ~~Разные форматы, кодировки и размеры.~~ **(Уже сделано.)**
 - Недоступные входные файлы:
 - Нет прав доступа.
 - Файл открыт и заблокирован.
 - Файл с атрибутом «только для чтения».
- ~~Остановка:~~
 - ~~Закрытием окна консоли.~~ **(Вычёркиваем. Не настолько важная проверка, а если и будут проблемы — технология PHP не позволит их решить.)**
- Журнал работы приложения:
 - Автоматическое создание (при отсутствии журнала), имя журнала указано явно.
 - Продолжение (дополнение журнала) при повторных запусках, имя журнала не указано.
- ~~Производительность:~~
 - ~~Элементарный тест с грубой оценкой.~~ **(Ранее решили, что наше приложение явно уложится в весьма демократичные требования заказчика.)**

Перепишем компактно то, что у нас осталось от уровня тестирования критического пути^[75]. Внимание! Это — НЕ тест-кейс! Это всего лишь ещё одна форма записи чек-листа, более удобная на данном этапе.

Таблица 2.4.g — Чек-лист для уровня критического пути

Суть проверки	Ожидаемая реакция
Запуск без параметров.	Отображение инструкции к использованию.
Запуск с недостаточным количеством параметров.	Отображение инструкции к использованию и указание имён недостающих параметров.
Запуск с неверными значениями параметров: <ul style="list-style-type: none"> ○ Недопустимый путь SOURCE_DIR. ○ Недопустимый путь DESTINATION_DIR. ○ Недопустимое имя LOG_FILE_NAME. ○ DESTINATION_DIR находится внутри SOURCE_DIR. ○ Значения DESTINATION_DIR и SOURCE_DIR совпадают. 	Отображение инструкции к использованию и указание имени неверного параметра, значения неверного параметра и пояснения сути проблемы.
Недоступные входные файлы: <ul style="list-style-type: none"> ○ Нет прав доступа. ○ Файл открыт и заблокирован. ○ Файл с атрибутом «только для чтения». 	Отображение сообщения в консоль и файл журнала, дальнейшее игнорирование недоступных файлов.
Журнал работы приложения: <ul style="list-style-type: none"> ○ Автоматическое создание (при отсутствии журнала), имя журнала указано явно. ○ Продолжение (дополнение журнала) при повторных запусках, имя журнала не указано. 	Создание или продолжение ведения файла журнала по указанному или вычисленному пути.

Наконец, у нас остался уровень расширенного тестирования⁽⁷⁶⁾. И сейчас мы сделаем то, чего по всем классическим книгам учат не делать, — мы откажемся от всего этого набора проверок целиком.

- ~~Конфигурирование и запуск:~~
 - ~~Значения SOURCE_DIR, DESTINATION_DIR, LOG_FILE_NAME:~~
 - * ~~В разных стилях (Windows-пути + *nix-пути) — одно в одном стиле, другое — в другом.~~
 - * ~~С использованием UNC-имён.~~
 - * ~~LOG_FILE_NAME внутри SOURCE_DIR.~~
 - * ~~LOG_FILE_NAME внутри DESTINATION_DIR.~~
 - ~~Размер LOG_FILE_NAME на момент запуска:~~
 - * ~~2–4 ГБ.~~
 - * ~~4+ ГБ.~~
 - ~~Запуск двух и более копий приложения с:~~
 - * ~~Одинаковыми параметрами SOURCE_DIR, DESTINATION_DIR, LOG_FILE_NAME.~~
 - * ~~Одинаковыми SOURCE_DIR и LOG_FILE_NAME, но разными DESTINATION_DIR.~~
 - * ~~Одинаковыми DESTINATION_DIR и LOG_FILE_NAME, но разными SOURCE_DIR.~~
- ~~Обработка файлов:~~
 - ~~Файл верного формата, в котором текст представлен в двух и более поддерживаемых кодировках одновременно.~~
 - ~~Размер входного файла:~~
 - * ~~2–4 ГБ.~~
 - * ~~4+ ГБ.~~

Да, мы сейчас действительно повысили риск пропустить какой-то дефект. Но — дефект, вероятность возникновения которого мала в силу малой вероятности возникновения описанных в этих проверках ситуаций. При этом по самым скромным

прикидкам мы на треть сократили общее количество проверок, которые нам нужно будет выполнять, а значит — высвободили силы и время для более тщательной проработки типичных каждодневных сценариев использования⁽¹⁴¹⁾ приложения.

Весь оптимизированный чек-лист (он же — и черновик для плана выполнения проверок) теперь выглядит так:

- 1) Подготовить файлы (см. таблицу 2.4.f).
- 2) Для «дымового теста» использовать командные файлы (см. приложение «Командные файлы для Windows и Linux, автоматизирующие выполнение дымового тестирования»⁽²⁷⁹⁾).
- 3) Для основных проверок использовать файлы из пункта 1 и следующие идеи (таблица 2.4.h).

Таблица 2.4.h — Основные проверки для приложения «Конвертер файлов»

Суть проверки	Ожидаемая реакция
Запуск без параметров.	Отображение инструкции к использованию.
Запуск с недостаточным количеством параметров.	Отображение инструкции к использованию и указание имён недостающих параметров.
Запуск с неверными значениями параметров: <ul style="list-style-type: none"> ○ Недопустимый путь SOURCE_DIR. ○ Недопустимый путь DESTINATION_DIR. ○ Недопустимое имя LOG_FILE_NAME. ○ DESTINATION_DIR находится внутри SOURCE_DIR. ○ Значения DESTINATION_DIR и SOURCE_DIR совпадают. 	Отображение инструкции к использованию и указание имени неверного параметра, значения неверного параметра и пояснения сути проблемы.
Недоступные входные файлы: <ul style="list-style-type: none"> ○ Нет прав доступа. ○ Файл открыт и заблокирован. ○ Файл с атрибутом «только для чтения». 	Отображение сообщения в консоль и файл журнала, дальнейшее игнорирование недоступных файлов.
Журнал работы приложения: <ul style="list-style-type: none"> ○ Автоматическое создание (при отсутствии журнала), имя журнала указано явно. ○ Продолжение (дополнение журнала) при повторных запусках, имя журнала не указано. 	Создание или продолжение ведения файла журнала по указанному или вычисленному пути.

- 4) В случае наличия времени использовать первоначальную редакцию чек-листа для уровня расширенного тестирования⁽⁷⁶⁾ как основу для выполнения исследовательского тестирования⁽⁸⁰⁾.

И почти всё. Остаётся только ещё раз подчеркнуть, что представленная логика выбора проверок не претендует на то, чтобы считаться единственно верной, но она явно позволяет сэкономить огромное количество усилий, при этом практически не снизив качество тестирования наиболее востребованной заказчиком функциональности приложения.



Задание 2.4.g: подумайте, какие проверки из таблицы 2.4.h можно автоматизировать с помощью командных файлов. Напишите такие командные файлы.

2.4.8. Типичные ошибки при разработке чек-листов, тест-кейсов и наборов тест-кейсов

Ошибки оформления и формулировок

Отсутствие заглавия тест-кейса или плохо написанное заглавие. В абсолютном большинстве систем управления тест-кейсами поле для заглавия вынесено отдельно и является обязательным к заполнению — тогда эта проблема отпадает. Если же инструментальное средство позволяет создать тест-кейс без заглавия, возникает риск получить N тест-кейсов, для понимания сути каждого из которых нужно прочесть десятки строк вместо одного предложения. Это гарантированное убийство рабочего времени и снижение производительности команды на порядок.

Если заглавие тест-кейса приходится вписывать в поле с шагами и инструментальное средство допускает форматирование текста, заглавие стоит писать **жирным шрифтом**, чтобы его было легче отделять от основного текста.

Продолжением этой ошибки является создание одинаковых заглавий, по которым объективно невозможно отличить один тест-кейс от другого. Более того, возникает подозрение, что одинаково озаглавленные тест-кейсы и внутри одинаковы. Потому следует формулировать заглавия по-разному, при этом подчёркивая в них суть тест-кейса и его отличие от других, похожих тест-кейсов.

И, наконец, в заглавии недопустимы «мусорные слова» вида «проверка», «тест» и т.д. Ведь это заглавие тест-кейса, т.е. речь по определению идёт о проверке, и не надо этот факт подчёркивать дополнительно. Также см. более подробное пояснение этой ошибки ниже в пункте «Постоянное использование слов «проверить» (и ему подобных) в чек-листах».

Отсутствие нумерации шагов и/или ожидаемых результатов (даже если таковой всего лишь один). Наличие этой ошибки превращает тест-кейс в «поток сознания», в котором нет структурированности, модифицируемости и прочих полезных свойств (да, многие свойства качественных требований^[41] в полной мере применимы и к тест-кейсам) — становится очень легко перепутать, что к чему относится. Даже выполнение такого тест-кейса усложняется, а доработка и вовсе превращается в каторжный труд.

Ссылка на множество требований. Иногда высокоуровневый тест-кейс^[115] действительно затрагивает несколько требований, но в таком случае рекомендуется писать ссылку на максимум 2–3 самых ключевых (наиболее соответствующих цели тест-кейса), а ещё лучше — указывать общий раздел этих требований (т.е. не ссылаться, например, на требования 5.7.1, 5.7.2, 5.7.3, 5.7.7, 5.7.9, 5.7.12, а просто сослаться на раздел 5.7, включающий в себя все перечисленные пункты). В большинстве инструментальных средств управления тест-кейсами это поле представляет собой выпадающий список, и там эта проблема теряет актуальность.

Использование личной формы глаголов. Если вы пишете требования на русском, то пишите «нажать» вместо «нажмите», «ввести» вместо «введите», «перейти» вместо «перейдите» и т.д. В технической документации вообще не рекомендуется «переходить на личности», а также существует мнение, что личная форма глаголов подсознательно воспринимается как «чужие бессмысленные команды» и приводит к повышенной утомляемости и раздражительности.

Использование прошедшего или будущего времени в ожидаемых результатах. Это не очень серьёзная ошибка, но всё равно «введённое значение отображается в поле» читается лучше, чем «введённое значение отобразилось в поле» или «введённое значение отобразится в поле».

Постоянное использование слов «проверить» (и ему подобных) в чек-листах. В итоге почти каждый пункт чек-листа начинается с «проверить ...», «проверить...», «проверить...». Но ведь весь чек-лист — это и есть список проверок! Зачем писать это слово? Сравните:

Плохо	Хорошо
Проверить запуск приложения.	Запуск приложения.
Проверить открытие корректного файла.	Открытие корректного файла.
Проверить модификацию файла.	Модификация файла.
Проверить сохранение файла.	Сохранение файла.
Проверить закрытие приложения.	Закрытие приложения.

Сюда же относится типичное слово «попытаться» в негативных тест-кейсах («попытаться поделить на ноль», «попытаться открыть несуществующий файл», «попытаться ввести недопустимые символы»): «деление на ноль», «открытие несуществующего файла», «ввод спецсимволов» намного короче и информативнее. А реакцию приложения, если она не очевидна, можно указать в скобках (так будет даже информативнее): «деление на ноль» (сообщение «Division by zero detected»), «открытие несуществующего файла» (приводит к автоматическому созданию файла), «ввод спецсимволов» (символы не вводятся, отображается подсказка).

Описание стандартных элементов интерфейса вместо использования их устоявшихся названий. «Маленький крестик справа сверху окна приложения» — это системная кнопка «Закрыть» (system button «Close»), «быстро-быстро дважды нажать на левую клавишу мыши» — это двойной щелчок (double click), «маленькое окошечко с надписью появляется, когда наводишь мышь» — это всплывающая подсказка (hint).

Пунктуационные, орфографические, синтаксические и им подобные ошибки. Без комментариев.

Логические ошибки

Ссылка на другие тест-кейсы или шаги других тест-кейсов. За исключением случаев написания строго оговорённого явно обозначенного набора последовательных тест-кейсов⁽¹⁴³⁾ это запрещено делать. В лучшем случае вам повезёт, и тест-кейс, на который вы ссылались, будет просто удалён — повезёт потому, что это будет сразу заметно. Не повезёт в случае, если тест-кейс, на который вы ссылаетесь, будет изменён — ссылка по-прежнему ведёт в некое существующее место, но описано там уже совершенно не то, что было в момент создания ссылки.

Детализация, не соответствующая уровню функционального тестирования⁽⁷⁴⁾. Например, не нужно на уровне дымового тестирования⁽⁷⁴⁾ проверять работоспособность каждой отдельной кнопки или прописывать некий крайне сложный, нетривиальный и редкий сценарий — поведение кнопок и без явного указания будет проверено множеством тест-кейсов, объективно задействующих эти кнопки, а сложному сценарию место на уровне тестирования критического пути⁽⁷⁵⁾ или даже на уровне расширенного тестирования⁽⁷⁶⁾ (в которых, напротив, недостатком можно считать излишнее обобщение без должной детализации).

Расплывчатые двусмысленные описания действий и ожидаемых результатов. Помните, что тест-кейс с высокой вероятностью будете выполнять не вы (автор тест-кейса), а другой сотрудник, и он — не телепат. Попробуйте догадаться по этим примерам, что имел в виду автор:

- «Установить приложение на диск С». (Т.е. в «С:\»? Прямо в корень? Или как?)
- «Нажать на иконку приложения». (Например, если у меня есть iso-файл с иконкой приложения, и я по нему кликну — это оно? Или нет?)
- «Окно приложения запустится». (Куда?)
- «Работает верно». (Ого! А верно — это, простите, как?)
- «ОК». (И? Что «ОК»?)
- «Количество найденных файлов совпадает». (С чем?)
- «Приложение отказывается выполнять команду». (Что значит «отказывается»? Как это выглядит? Что должно происходить?)

Описание действий в качестве наименований модуля/подмодуля. Например, «запуск приложения» — это НЕ модуль или подмодуль. Модуль или подмодуль⁽¹²⁰⁾ — это всегда некие части приложения, а не его поведение. Сравните: «дыхательная система» — это модуль человека, но «дыхание» — нет.

Описание событий или процессов в качестве шагов или ожидаемых результатов. Например, в качестве шага сказано: «Ввод спецсимволов в поле X». Это было бы сносным заглавием тест-кейса, но не годится в качестве шага, который должен быть сформулирован как «Ввести спецсимволы (перечень) в поле X».

Куда страшнее, если подобное встречается в ожидаемых результатах. Например, там написано: «Отображение скорости чтения в панели X». И что? Оно должно начаться, продолжиться, завершиться, не начинаться, неким образом измениться (например, измениться должна размерность данных), как-то на что-то повлиять? Тест-кейс становится полностью бессмысленным, т.к. такой ожидаемый результат невозможно сравнить с фактическим поведением приложения.

«Выдумывание» особенностей поведения приложения. Да, часто в требованиях отсутствуют самоочевидные (без кавычек, они на самом деле самоочевидные) вещи, но нередко встречаются и некачественные (например, неполные) требования, которые нужно улучшать, а не «телепатически компенсировать».

Например, в требованиях сказано, что «приложение должно отображать диалоговое окно сохранения с указанным по умолчанию каталогом». Если из контекста (соседних требований, иных документов) ничего не удаётся узнать об этом таинственном «каталоге по умолчанию», нужно задать вопрос. Нельзя просто записать в ожидаемых результатах «отображается диалоговое окно сохранения с указанным по умолчанию каталогом» (как мы проверим, что выбран именно указанный по умолчанию каталог, а не какой-то иной?). И уж тем более нельзя в ожидаемых результатах писать «отображается диалоговое окно сохранения с выбранным каталогом “C:/SavedDocuments”» (откуда взялось это «C:/SavedDocuments», — не ясно, т.е. оно явно выдумано из головы и, скорее всего, выдумано неправильно).

Отсутствие описания приготовления к выполнению тест-кейса. Часто для корректного выполнения тест-кейса необходимо каким-то особым образом настроить окружение. Предположим, что мы проверяем приложение, выполняющее резервное копирование файлов. Если тест выглядит примерно так, то выполняющий его сотрудник будет в замешательстве, т.к. ожидаемый результат кажется просто бредом. Откуда взялось «~200»? Что это вообще такое?

Шаги выполнения	Ожидаемые результаты
<ol style="list-style-type: none"> 1. Нажать на панели «Главная» кнопку «Быстрая дедубликация». 2. Выбрать каталог «C:/MyData». 	<ol style="list-style-type: none"> 1. Кнопка «Быстрая дедубликация» переходит в утопленное состояние и меняет цвет с серого на зелёный. 2. На панели «Состояние» в поле «Дубликаты» отображается «~200».

И совсем иначе этот тест-кейс воспринимался бы, если бы в приготовлениях было сказано: «Создать каталог “C:/MyData” с произвольным набором подкаталогов (глубина вложенности не менее пяти). В полученном дереве каталогов разместить 1000 файлов, из которых 200 имеют одинаковые имена и размеры, но НЕ внутреннее содержимое».

Полное дублирование (копирование) одного и того же тест-кейса на уровнях дымового тестирования, тестирования критического пути, расширенного тестирования. Многие идеи естественным образом развиваются от уровня к уровню^[74], но они должны именно развиваться, а не дублироваться. Сравните:

	Дымовое тестирование	Тестирование критического пути	Расширенное тестирование
Плохо	Запуск приложения.	Запуск приложения.	Запуск приложения.
Хорошо	Запуск приложения.	Запуск приложения из командной строки. Запуск приложения через ярлык на рабочем столе. Запуск приложения через меню «Пуск».	Запуск приложения из командной строки в активном режиме. Запуск приложения из командной строки в фоновом режиме. Запуск приложения через ярлык на рабочем столе от имени администратора. Запуск приложения через меню «Пуск» из списка часто запускаемых приложений.

Слишком длинный перечень шагов, не относящихся к сути (цели) тест-кейса. Например, мы хотим проверить корректность одностороннего режима печати из нашего приложения на дуплексном принтере. Сравните:

Плохо	Хорошо
Односторонняя печать <ol style="list-style-type: none"> 1. Запустить приложение. 2. В меню выбрать «Файл» -> «Открыть». 3. Выбрать любой DOCX-файл, состоящий из нескольких страниц. 4. Нажать кнопку «Открыть». 5. В меню выбрать «Файл» -> «Печать». 6. В списке «Двусторонняя печать» выбрать пункт «Нет». 7. Нажать кнопку «Печать». 8. Заккрыть файл. 9. Заккрыть приложение. 	Односторонняя печать <ol style="list-style-type: none"> 1. Открыть любой DOCX-файл, содержащий три и более непустых страницы. 2. В диалоговом окне «Настройка печати» в списке «Двусторонняя печать» выбрать «Нет». 3. Произвести печать документа на принтере, поддерживающем двустороннюю печать.

Слева мы видим огромное количество действий, не относящихся непосредственно к тому, что проверяет тест-кейс. Тем более что запуск и закрытие приложения, открытие файла, работа меню и прочее или будут покрыты другими тест-кейсами (со своими соответствующими целями), или на самом деле являются самоочевидными (логично ведь, что нельзя открыть приложением файл, если приложение не запущено) и не нуждаются в описании шагов, которые только создают информационный шум и занимают время на написание и прочтение.

Некорректное наименование элементов интерфейса или их свойств.

Иногда из контекста понятно, что автор тест-кейса имел в виду, но иногда это становится реальной проблемой. Например, мы видим тест-кейс с заголовком «Закрытие приложения кнопками "Close" и "Close window"». Уже тут возникает недоумение по поводу того, в чём же различие этих кнопок, да и о чём вообще идёт речь. Ниже (в шагах тест-кейса) автор поясняет: «В рабочей панели внизу экрана нажать "Close window"». Ага! Ясно. Но «Close window» — это НЕ кнопка, это пункт системного контекстного меню приложения в панели задач.

Ещё один отличный пример: «Окно приложения свернётся в окно меньшего диаметра». Хм. Окно круглое? Или должно стать круглым? А, может, тут и вовсе речь про два разных окна, и одно должно будет оказаться внутри второго? Или, всё же «размер окна уменьшается» (кстати, насколько?), а его геометрическая форма остаётся прямоугольной?

И, наконец, пример, из-за которого вполне можно написать отчёт о дефекте на вполне корректно работающее приложение: «В системном меню выбрать “Фиксация расположения”». Казалось бы, что ж тут плохого? Но потом выясняется, что имелось в виду главное меню приложения, а вовсе не системное меню.

Непонимание принципов работы приложения и вызванная этим некорректность тест-кейсов. Классикой жанра является закрытие приложения: тот факт, что окно приложения «исчезло» (сюрприз: например, оно свернулось в область уведомлений панели задач (system tray, taskbar notification area)), или приложение отключило интерфейс пользователя, продолжив функционировать в фоновом режиме, вовсе не является признаком того, что оно завершило работу.

Проверка типичной «системной» функциональности. Если только ваше приложение не написано с использованием каких-то особенных библиотек и технологий и не реализует какое-то нетипичное поведение, нет необходимости проверять системные кнопки, системные меню, сворачивание-разворачивание окна и т.д. Вероятность встретить здесь ошибку стремится к нулю. Если всё же очень хочется, можно вписать эти проверки как уточнения некоторых действий на уровне тестирования критического пути⁽⁷⁵⁾, но создавать для них отдельные тест-кейсы не нужно.

Неверное поведение приложения как ожидаемый результат. Такое не допускается по определению. Не может быть тест-кейса с шагом в стиле «поделить на ноль» с ожидаемым результатом «крах приложения с потерей пользовательских данных». Ожидаемые результаты всегда описывают правильное поведение приложения — даже в самых страшных стрессовых тест-кейсах.

Общая некорректность тест-кейсов. Может быть вызвана множеством причин и выражаться множеством образов, но вот классический пример:

Шаги выполнения	Ожидаемые результаты
<p>...</p> <p>4. Закрыть приложение нажатием Alt+F4.</p> <p>5. Выбрать в меню «Текущее состояние».</p>	<p>...</p> <p>4. Приложение завершает работу.</p> <p>5. Отображается окно с заголовком «Текущее состояние» и содержимым, соответствующим рисунку 999.99.</p>

Здесь или не указано, что вызов окна «Текущее состояние» происходит где-то в другом приложении, или остаётся загадкой, как вызвать это окно у завершившего работу приложения. Запустить заново? Возможно, но в тест-кейсе этого не сказано.

Неверное разбиение наборов данных на классы эквивалентности. Действительно, иногда классы эквивалентности⁽⁸⁹⁾ могут быть очень неочевидными. Но ошибки встречаются и в довольно простых случаях. Допустим, в требованиях сказано, что размер некоего файла может быть от 10 до 100 КБ (включительно). Разбиение по размеру 0–9 КБ, 10–100 КБ, 101+ КБ **ошибочно**, т.к. килобайт не является неделимой единицей. Такое ошибочное разбиение не учитывает, например, размеры в 9.5 КБ, 100.1 КБ, 100.7 КБ и т.д. Потому здесь стоит применять неравенства: $0 \text{ КБ} \leq \text{размер} < 10 \text{ КБ}$, $10 \text{ КБ} \leq \text{размер} \leq 100 \text{ КБ}$, $100 \text{ КБ} < \text{размер}$. Также можно писать с использованием синтаксиса скобок: $[0, 10) \text{ КБ}$, $[10, 100] \text{ КБ}$, $(100, \infty) \text{ КБ}$, но вариант с неравенствами более привычен большинству людей.

Тест-кейсы, не относящиеся к тестируемому приложению. Например, нам нужно протестировать фотогалерею на сайте. Соответственно, следующие тест-кейсы никак не относятся к фотогалерее (они тестируют браузер, операционную систему пользователя, файловый менеджер и т.д. — но НЕ наше приложение, ни его серверную, ни даже клиентскую часть):

- Файл с сетевого диска.
- Файл со съёмного носителя.
- Файл, заблокированный другим приложением.
- Файл, открытый другим приложением.
- Файл, к которому у пользователя нет прав доступа.
- Вручную указать путь к файлу.
- Файл из глубоко расположенной поддиректории.

Формальные и/или субъективные проверки. Чаще всего данную ошибку можно встретить в пунктах чек-листа. Возможно, у автора в голове и был чёткий и подробный план, но из следующих примеров совершенно невозможно понять, что будет сделано с приложением, и какой результат мы должны ожидать:

- «Сконвертировать».
- «Проверить метод getMessage()».
- «Некорректная работа в корректных условиях».
- «Скорость».
- «Объём данных».
- «Должно работать быстро».

В отдельных исключительных ситуациях можно возразить, что из контекста и дальнейшей детализации становится понятно, что имелось в виду. Но чаще всего никакого контекста и никакой дальнейшей детализации нет, т.е. приведённые примеры оформлены как отдельные полноправные пункты чек-листа. Так — нельзя.

Как можно и нужно – см. в примере чек-листа⁽¹¹¹⁾ и всём соответствующем разделе⁽¹¹⁰⁾.

Теперь для лучшего закрепления рекомендуется заново прочитать про оформление атрибутов тест-кейсов⁽¹¹⁹⁾, свойства качественных тест-кейсов⁽¹³¹⁾ и логику построения⁽¹⁴⁷⁾ качественных тест-кейсов и качественных наборов тест-кейсов.