**Чек-листы vs Тест-кейсы**

Чек-лист отличается от тест-кейса степенью подробности. В чек-листе вы не встретите подробных шагов кейса, для использования чек-листа при тестировании очень много информации нужно держать в голове в момент прогона тестов и знать логику работы приложения на отлично.   
  
В нашей компании всегда использовались чек-листы, поскольку на написание тест-кейсов уходит неоправданно много времени, и они тяжеловесны — на прочтение кейса и его осознание тоже уходит время. Кроме того, не стоит забывать про эффект пестицида — баги кода имеют свойство приспосабливаться к тестам. При использовании чек-листа сохраняется некоторая свобода действий, а тест-кейсы этой свободы полностью лишают, увеличивая упомянутый выше эффект. Однако, при прогоне чек-листа в седьмой раз за последние сутки перед релизом часть функциональности, заложенной под одним пунктом чек-листа, теряется по причине человеческого фактора.  
  
Было принято решение расширять чек-листы и делать их подробнее. Так тестировщик, в беспамятстве прогоняющий фичу перед релизом, не забудет проверить ошибки сети в ответ на каждый запрос, не потеряет проверку какой-нибудь «неважной» кнопки или какого-нибудь одного статуса из двенадцати. Так мы пришли к написанию подробной юзер-стори, полностью покрывающей фичу приложения, но по факту являющей собой один громадный тест-кейс.   
  
Плюсы такого подхода:

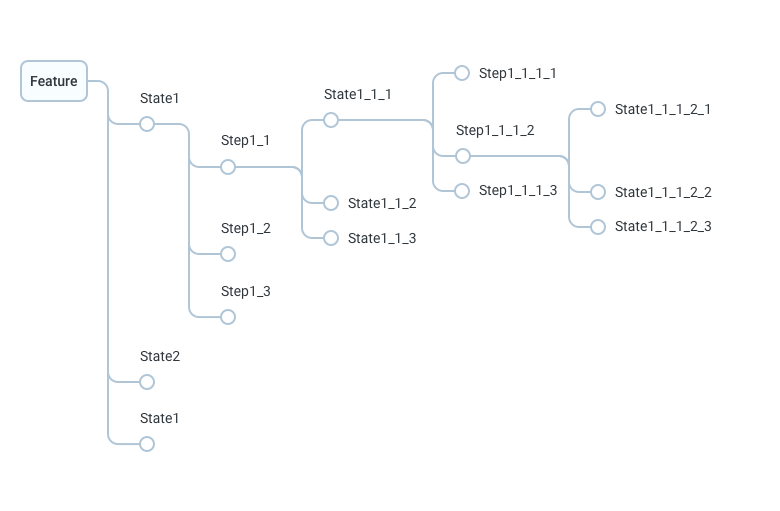
* чек-лист подразумевает непрерывное выполнение его тестировщиком. Соответственно, все кейсы расположены в порядке, удобном для прогона, и времени на переход к следующему кейсу не тратится
* чек-лист покрывает большое количество пользовательских сценариев
* чек-лист содержит как позитивные, так и негативные кейсы. Проверяет восстановление после ошибок и прерываний, что в случае мобильных приложений очень важно
* чек-лист подразумевает длинные сессии, что повышает вероятность обнаружения утечек памяти и навигационных проблем
* информация о требованиях поступает тестировщику последовательно, что дает лучшее понимание логики работы приложения

Минусы такого подхода:

* чек-лист подразумевает декомпозицию по экранам приложения и сильно завязан на дизайн. Соответственно, все изменения в дизайне подразумевают изменения в чек-листе
* с учетом, что кейсы такого чек-листа должны быть связаны в единую стори, а кейсов в одном чек-листе может быть до двух сотен, поддерживать его сложно
* эффект пестицида значительно повышается
* чек-лист не может заменить требования для разработчиков — разрабатывать по таким требованиям неудобно
* чек-лист довольно долго писать и из-за его громоздкости его не всегда удобно использовать — по факту он пригождается только в финале перед релизом

Долгое время мы жили так. Вели свои чек-листы в системе тест-менеджмента Sitechko. Но компания менялась, менялись бизнес-процессы, и вопрос о переходе на другую TMS висел в воздухе. Он стал толчком для пересмотра процесса ведения тестовой документации.

**Таблицы vs Деревья**

Однажды я выпила поливитаминов, и меня осенило, что вместо того, чтобы хранить тесты в табличных форматах, куда удобнее использовать деревья, а точнее, вложенные списки. Ведь при написании той самой большой юзер-стори мы так часто сталкиваемся с проблемой, что не знаем, где расположить альтернативные действия. Например, когда у нас несколько кнопок на алерте. Чтобы проверить каждую из них, нам приходится прописывать вызов этого алерта несколько раз. Вместо этого мы могли бы подвесить проверку каждой из кнопок к вызову алерта.  
  
В целом идея была в том, чтобы прописывать те же самые пользовательские сценарии в виде дерева, в котором переход — это действие, а узел — это состояние, в котором оказывается приложение после этого действия. По факту диаграмма состояний-переходов, только объектами выступают экраны приложения. Каждая ветвь такого дерева — тест-кейс.  
  
  
  
Когда стали пробовать, столкнулись с проблемами. Оказалось, что привычная нам декомпозиция по экранам приложения не работает: опираться на дизайн при проектировании тестов неудобно. Ветки дерева росли далеко в глубину, и это было неудобно визуально. В погоне за сценарием мы воротили циклы. А еще стало понятно, что отказаться от таблиц нельзя.  
  
Решение крылось в смене подхода к декомпозиции, большей осознанности и отказе от «решений по умолчанию». Древовидная структура тестовой документации действительно удобна, поскольку дает большую свободу при проектировании. Вид декомпозиции определяет, что именно станет узлами нашего дерева. А это в свою очередь определяется особенностями продукта и приоритетами заказчика.   
  
В целом, плюсы использования древовидной структуры:

* структура такой документации в итоге очень гибкая и позволяет вести как чек-листы, так и тест-кейсы в зависимости от нужд проекта
* дерево представляется в виде вложенных списков: у узлов дерева есть некоторый порядок, что сохраняет возможность последовательной и структурированной передачи информации о требованиях тестировщику в случае отсутствия на проекте задокументированного ТЗ
* такая структура позволяет спроектировать тестовую документацию, которую можно передать разработчикам вместо ТЗ
* временные затраты на написание чек-листов и тест-кейсов снижаются, поскольку структура позволяет избежать копипасты

Минусы подхода:

* такая структура тестовой документации требует тщательного проектирования и предварительной аналитики — при плохом проектировании все упомянутые выше плюсы теряются

**Итоговые паттерны**

**Экраны приложения**

Источником знаний является навсхема. Первый уровень дерева составляет список кейсов навсхемы, который обычно соответствует разделам приложения. Далее к ним подвешивается список экранов каждого раздела, к каждому экрану — список его состояний. В каждом узле дерева, начиная с третьего уровня, может содержаться чек-лист в табличном формате, описывающий каждый элемент дизайна и способы взаимодействия с ним. Если элементы дизайна сложные и имеют много состояний или на экране есть повторяющиеся элементы, можно декомпозировать еще глубже. Таким образом, одна ветвь дерева описывает жизненный цикл одного экрана.  
  
Ниже в качестве примера приведена общая схема рассуждений при декомпозиции раздела заказов агрегатора авиабилетов.