



ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

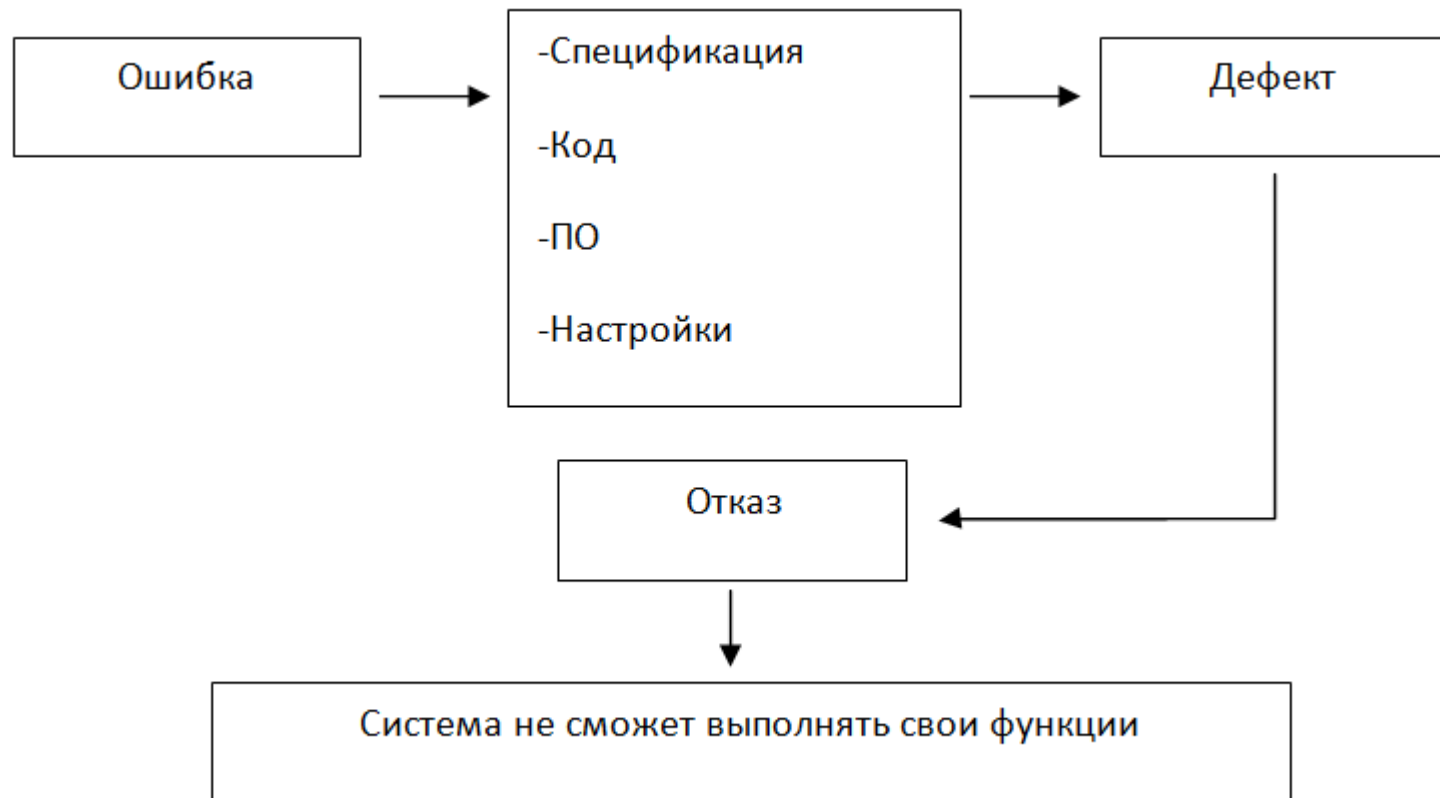
ЛЕКЦИЯ № 9

ПРЕПОДАВАТЕЛЬ: ХУСТОЧКА А.В.



ОТЧЁТЫ О ДЕФЕКТАХ

- Баг(дефект)— расхождение ожидаемого и фактического результата.
- Ожидаемый результат — поведение системы, описанное в требованиях.
- Фактический результат — поведение системы, наблюдаемое в процессе тестирования.



- Ошибка — действие человека, приводящее к некорректным результатам. Этот термин очень часто используют как наиболее универсальный, описывающий любые проблемы («ошибка человека», «ошибка в коде», «ошибка в документации», «ошибка выполнения операции», «ошибка передачи данных», «ошибочный результат» и т.п.). Ошибки, сделанные программистом, известны как «Ошибка», что, в свою очередь, приводит к ошибке в программе.
- Дефект — это отклонение от требований спецификаций или ожиданий пользователей. Другими словами, дефект является ошибкой в кодировании или логике, что приводит к сбою программы или созданию неправильного / неожиданного результата. Это могут быть аппаратные средства, программное обеспечение, сеть, производительность, формат или функциональность. Недостаток в компоненте или системе, способный привести к ситуации сбоя или отказа.
- Сбой — самоустраняющийся отказ или однократный отказ, устраняемый незначительным вмешательством оператора.
- Отказ — событие, заключающееся в нарушении работоспособного состояния объекта. Это неспособность системы или компонента выполнять требуемые функции в рамках определенных требований к производительности. Неисправность возникает при выполнении ошибки.

Эти термины скорее относятся к теории надёжности и нечасто встречаются в повседневной работе тестировщика, но именно сбои и отказы являются тем, что тестировщик замечает в процессе тестирования (и отталкиваясь от чего, проводит исследование с целью выявить дефект и его причины).

- Отчёт о дефекте - это документ, описывающий ситуацию или последовательность действий приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.

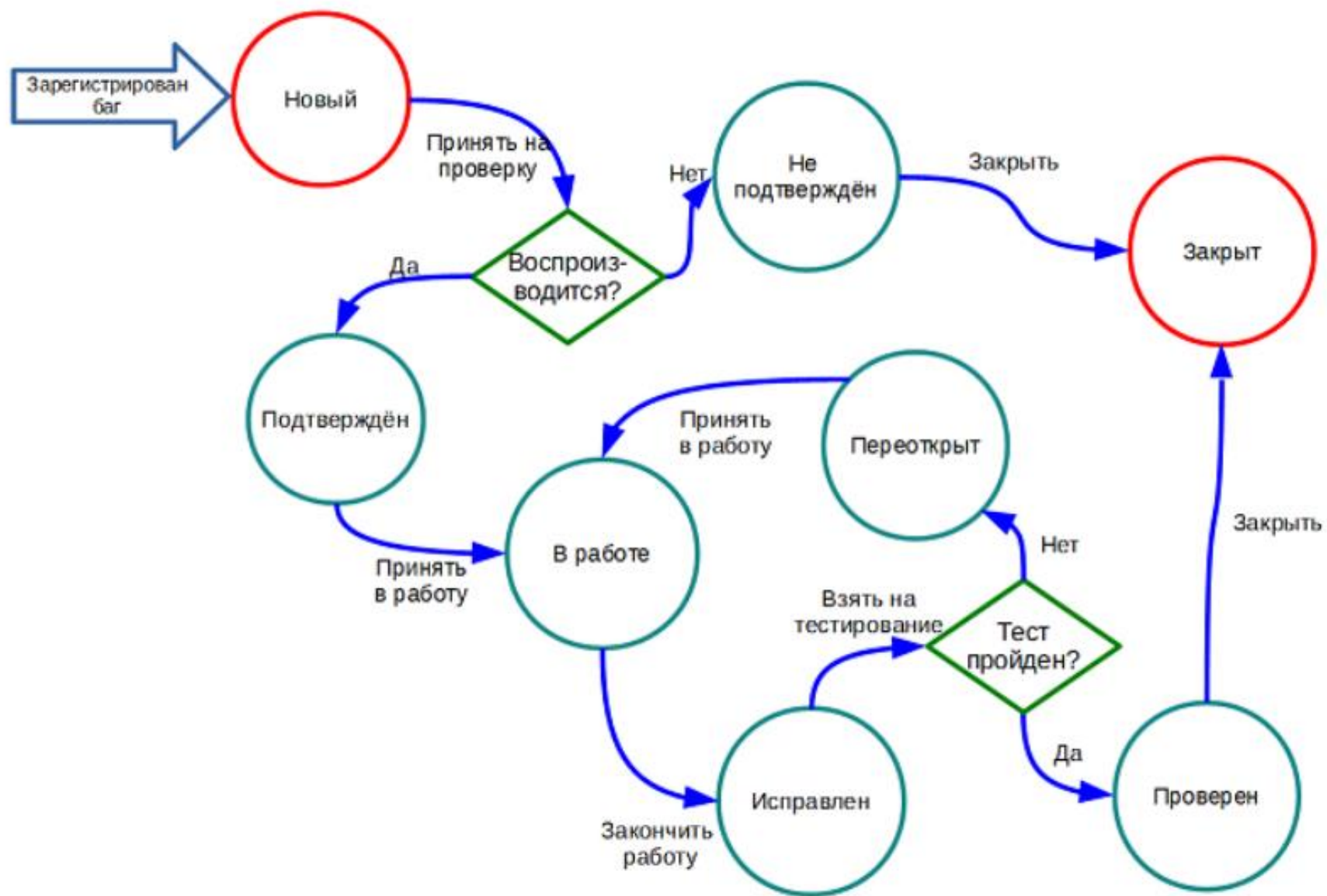
- Цели написания отчёта о дефекте
 - Как следует из самого определения, отчёт о дефекте пишется со следующими основными целями:
 - Предоставить информацию о проблеме — уведомить проектную команду и иных заинтересованных лиц о наличии проблемы, описать суть проблемы.
 - Приоритизировать проблему — определить степень опасности проблемы для проекта и желаемые сроки её устранения.
 - Содействовать устранению проблемы — качественный отчёт о дефекте не только предоставляет все необходимые подробности для понимания сути случившегося, но также может содержать анализ причин возникновения проблемы и рекомендации по исправлению ситуации.

ЖИЗНЕННЫЙ ЦИКЛ “БАГА”

Отчёт о дефекте (и сам дефект вместе с ним) проходит определённые стадии жизненного цикла:

- Обнаружен (submitted) — начальное состояние отчёта (иногда называется «Новый» (new)), в котором он находится сразу после создания. Некоторые средства также позволяют сначала создавать черновик (draft) и лишь потом публиковать отчёт.
- Назначен (assigned) — в это состояние отчёт переходит с момента, когда кто-то из проектной команды назначается ответственным за исправление дефекта. Назначение ответственного может производиться решением лидера команды разработки, коллегиально, по добровольному принципу, иным принятым в команде способом или выполняется автоматически на основе определённых правил.
- Исправлен (fixed) — в это состояние отчёт переводит ответственный за исправление дефекта член команды после выполнения соответствующих действий по исправлению.

- Проверен (verified) — в это состояние отчёт переводит тестировщик, удостоверившись, что дефект на самом деле был устранён. Как правило, такую проверку выполняет тестировщик, изначально написавший отчёт о дефекте.
- Закрит (closed) — состояние отчёта, означающее, что по данному дефекту не планируется никаких дальнейших действий. Здесь есть некоторые расхождения в жизненном цикле, принятом в разных инструментальных средствах управления отчётами о дефектах.
- Открыт заново (reopened) — в это состояние (как правило, из состояния «Исправлен») отчёт переводит тестировщик, удостоверившись, что дефект по-прежнему воспроизводится на билде, в котором он уже должен быть исправлен.
- Рекомендован к отклонению (to be declined) — в это состояние отчёт о дефекте может быть переведён из множества других состояний, чтобы вынести на рассмотрение вопрос об отклонении отчёта по той или иной причине. Если рекомендация является обоснованной, то отчёт переводится в состояние «Отклонён».
- Отклонён (declined) — в это состояние отчёт переводится в случаях, подробно описанных в пункте «Закрит»: если средство управления отчётами о дефектах предполагает использование этого состояния вместо состояния «Закрит» для тех или иных резолюций по отчёту.
- Отложен (deferred) — в это состояние отчёт переводится в случае, если исправление дефекта в ближайшее время является нерациональным или не представляется возможным, однако есть основания полагать, что в обозримом будущем ситуация исправится (выйдет новая версия библиотеки, вернётся из отпуска специалист по необходимой технологии, изменятся требования заказчика и т.д.)



ИСПОЛЬЗОВАНИЕ ДАННЫХ ОТЧЕТА О ДЕФЕКТЕ

Информация по дефектам – это записи о просчетах в качестве, а значит – список возможностей для улучшения качества на ваших будущих проектах. Если вы собираете некую информацию о дефектах (например, после релиза или только на больших проектах), тогда, возможно, Вы захотите улучшить этот процесс.

Информация о дефектах, которая может быть полезна для улучшения качества, включает следующие вопросы:

- Что было не так? Решать нужно саму проблему, а не ее симптомы. Например, заикливание - это симптом, а изменение индекса цикла - это проблема.
- Когда была создана эта проблема? Какое именно действие при разработке явилось ее источником? Это была проблема в требованиях? Проектировании системы? Коде? Тестировании?
- Когда проблема была выявлена? Может, она и не была сразу же устранена, но что нас интересует: сколько она существовала до того, как мы ее обнаружили?
- Каким образом была найдена эта проблема? Способ обнаружения можно внедрить в постоянно используемую практику.
- Можно ли было обнаружить ее раньше? Есть ли какой-либо процесс контроля качества, который мог бы ее выявить, будь он эффективнее?
- Сколько стоило устранение этой проблемы? Этот момент очень легко недооценить. Убедитесь, что Вы учли процесс диагностики проблемы и всю работу по ее устранению, которую Вам пришлось сделать, включая ре-дизайн, переписывание кода, ре-компиляцию, переработку тестов, повторное тестирование, повторный релиз, выпуск заплатки, формирование отчета по дефекту, отчет по статусу проекта и т.д. (не забудьте еще возможные затраты на исправление подпорченной репутации на рынке ПО).
- Какого рода была эта проблема? Когда у Вас огромное количество дефектов, то их категоризация облегчает анализ и обучение.

Когда Вы анализируете информацию о дефектах, то ищите те дефекты, которые обнаруживаются регулярно, и те, затраты на устранение которых высоки. Вот как раз таких дефектов и нужно избегать в будущем (или, по крайней мере, устранять их на более ранней стадии разработки), именно такая тактика гарантированно будет способствовать улучшению качества.

АТРИБУТЫ ОТЧЁТА О ДЕФЕКТЕ

В зависимости от инструментального средства управления отчётами о дефектах внешний вид их записи может немного отличаться, могут быть добавлены или убраны отдельные поля, но концепция остаётся неизменной.

- Идентификатор (identifier) представляет собой уникальное значение, позволяющее однозначно отличить один отчёт о дефекте от другого и используемое во всевозможных ссылках. В общем случае идентификатор отчёта о дефекте может представлять собой просто уникальный номер, но (если позволяет инструментальное средство управления отчётами) может быть и куда сложнее: включать префиксы, суффиксы и иные осмысленные компоненты, позволяющие быстро определить суть дефекта и часть приложения (или требований), к которой он относится.
- Краткое описание (summary) должно в предельно лаконичной форме давать исчерпывающий ответ на вопросы «Что произошло?», «Где это произошло?», «При каких условиях это произошло?». Например: «Отсутствует логотип на странице приветствия, если пользователь является администратором». Что произошло? Отсутствует логотип. Где это произошло? На странице приветствия. При каких условиях это произошло? Если пользователь является администратором.

- Подробное описание (description) представляет в развёрнутом виде необходимую информацию о дефекте, а также (обязательно!) описание фактического результата, ожидаемого результата и ссылку на требование (если это возможно). В отличие от краткого описания, которое, как правило, является одним предложением, здесь можно и нужно давать подробную информацию. Если одна и та же проблема (вызванная одним источником) проявляется в нескольких местах приложения, можно в подробном описании перечислить эти места.
- Шаги по воспроизведению (steps to reproduce, STR) описывают действия, которые необходимо выполнить для воспроизведения дефекта. Это поле похоже на шаги тест-кейса, за исключением одного важного отличия: здесь действия прописываются максимально подробно, с указанием конкретных вводимых значений и самых мелких деталей, т.к. отсутствие этой информации в сложных случаях может привести к невозможности воспроизведения дефекта.
- Воспроизводимость (reproducibility) показывает, при каждом ли прохождении по шагам воспроизведения дефекта удаётся вызвать его проявление. Это поле принимает всего два значения: всегда (always) или иногда (sometimes). Можно сказать, что воспроизводимость «иногда» означает, что тестировщик не нашёл настоящую причину возникновения дефекта. Тестировщику нужно потратить много времени на то, чтобы удостовериться в наличии дефекта (т.к. однократный сбой в работе приложения мог быть вызван огромным количеством посторонних причин). Разработчику тоже нужно потратить время, чтобы добиться проявления дефекта и убедиться в его наличии. После внесения исправлений в приложение разработчик фактически должен полагаться только на свой профессионализм, т.к. даже многократное прохождение по шагам воспроизведения в таком случае не гарантирует, что дефект был исправлен (возможно, через ещё 10–20 повторений он бы проявился).

- Важность (severity) показывает степень ущерба, который наносится проекту существованием дефекта. В общем случае выделяют следующие градации важности:
 - критическая (critical) — существование дефекта приводит к масштабным последствиям катастрофического характера: потеря данных, раскрытие конфиденциальной информации, нарушение ключевой функциональности приложения и т.д.
 - высокая (major) — существование дефекта приносит ощутимые неудобства многим пользователям в рамках их типичной деятельности: недоступность вставки из буфера обмена, неработоспособность общепринятых клавиатурных комбинаций, необходимость перезапуска приложения при выполнении типичных сценариев работы.
 - средняя (medium) — существование дефекта слабо влияет на типичные сценарии работы пользователей, и/или существует обходной путь достижения цели, например: диалоговое окно не закрывается автоматически после нажатия кнопок «ОК»/«Cancel», при распечатке нескольких документов подряд не сохраняется значение поля «Двусторонняя печать», перепутаны направления сортировок по некоему полю таблицы.
 - низкая (minor) — существование дефекта редко обнаруживается незначительным процентом пользователей и почти не влияет на их работу, например: опечатка в глубоко вложенном пункте меню настроек, некое окно сразу при отображении расположено неудобно (нужно перетянуть его в удобное место), неточно отображается время до завершения операции копирования файлов.
- Срочность (priority) показывает, как быстро дефект должен быть устранён. В общем случае выделяют следующие градации срочности:
 - наивысшая (ASAP, as soon as possible) срочность указывает на необходимость устранить дефект настолько быстро, насколько это возможно. В зависимости от контекста «насколько быстро, насколько возможно» может варьироваться от «в ближайшем билде» до единиц минут.
 - высокая (high) срочность означает, что дефект следует исправить вне очереди, т.к. его существование или уже объективно мешает работе, или начнёт создавать такие помехи в самом ближайшем будущем.
 - обычная (normal) срочность означает, что дефект следует исправить в порядке общей очерёдности. Такое значение срочности получает большинство дефектов.
 - низкая (low) срочность означает, что, в обозримом будущем, исправление данного дефекта не окажет существенного влияния на повышение качества продукта.

- Фактический результат (actual result) - результат, полученный после прохождения шагов к воспроизведению.
- Ожидаемый результат (expected result) - описывает ожидаемое поведение ПО после прохождения шагов к воспроизведению.
- Симптом (symptom) — позволяет классифицировать дефекты по их типичному проявлению. Не существует никакого общепринятого списка симптомов. Более того, далеко не в каждом инструментальном средстве управления отчётами о дефектах есть такое поле, а там, где оно есть, его можно настроить. В качестве примера есть следующие значения симптомов дефекта: Косметический дефект (cosmetic flaw), Повреждение/потеря данных (data corruption/loss), Проблема в документации (documentation issue), Некорректная операция (incorrect operation), Проблема инсталляции (installation problem), Ошибка локализации (localization issue), Нереализованная функциональность (missing feature), Проблема масштабируемости (scalability), Низкая производительность (low performance), Крах системы (system crash), Неожиданное поведение (unexpected behavior), Недружественное поведение (unfriendly behavior), Расхождение с требованиями (variance from specs), Предложение по улучшению (enhancement).
- Комментарий (comments, additional info) — может содержать любые полезные для понимания и исправления дефекта данные. Иными словами, сюда можно писать всё то, что нельзя писать в остальные поля.
- Приложения (attachments) — представляет собой не столько поле, сколько список прикрепленных к отчёту о дефекте приложений (копий экрана, вызывающих сбой файлов и т.д.)

СВОЙСТВА КАЧЕСТВЕННЫХ ОТЧЁТОВ О ДЕФЕКТАХ

Отчёт о дефекте может оказаться некачественным (а следовательно, вероятность исправления дефекта понизится), если в нём нарушено одно из следующих свойств:

- Тщательное заполнение всех полей точной и корректной информацией. Нарушение этого свойства происходит по множеству причин: недостаточный опыт тестировщика, невнимательность, лень и т.д.
- Правильный технический язык. Это свойство в равной мере относится и к требованиям, и к тест-кейсам, и к отчётам о дефектах — к любой документации, а потому не будем повторяться.
- Специфичность описания шагов. Говоря о тест-кейсах, мы подчёркивали, что в их шагах стоит придерживаться золотой середины между специфичностью и общностью. В отчётах о дефектах предпочтение, как правило, отдаётся специфичности по очень простой причине: нехватка какой-то мелкой детали может привести к невозможности воспроизведения дефекта. Потому, если у вас есть хоть малейшее сомнение в том, важна ли какая-то деталь, считайте, что она важна.

- Отсутствие лишних действий и/или их длинных описаний. Чаще всего, это свойство подразумевает, что не нужно в шагах по воспроизведению дефекта долго и по пунктам расписывать то, что можно заменить одной фразой.
- Отсутствие дубликатов. Когда в проектной команде работает большое количество тестировщиков, то может возникнуть ситуация, при которой один и тот же дефект будет описан несколько раз разными людьми.
- Очевидность и понятность. Описывайте дефект так, чтобы у читающего Ваш отчёт не возникло ни малейшего сомнения в том, что это действительно дефект. Лучше всего это свойство достигается за счёт тщательного объяснения фактического и ожидаемого результата, а также указания ссылки на требование в поле «Подробное описание».
- Прослеживаемость. Из содержащейся в качественном отчёте о дефекте информации должно быть понятно, какую часть приложения, какие функции и какие требования затрагивает дефект. Лучше всего это свойство достигается правильным использованием возможностей инструментального средства управления отчётами о дефектах: указывайте в отчёте о дефекте компоненты приложения, ссылки на требования, тест-кейсы, смежные отчёты о дефектах (похожих дефектах; зависимых и зависящих от данного дефектах), расставляйте теги и т.д.
- Отдельные отчёты для каждого нового дефекта. Существует два незыблемых правила:
 - в каждом отчёте описывается ровно один дефект (если один и тот же дефект проявляется в нескольких местах, то эти проявления перечисляются в подробном описании).
 - при обнаружении нового дефекта создаётся новый отчёт. Нельзя для описания нового дефекта править старые отчёты, переводя их в состояние «открыт заново».
- Соответствие принятым шаблонам оформления и традициям. Как и в случае с тест-кейсами, с шаблонами оформления отчётов о дефектах проблем не возникает: они определены имеющимся образцом или экранной формой инструментального средства управления жизненным циклом отчётов о дефектах. Но, что касается традиций, которые могут различаться даже в разных командах в рамках одной компании, единственный совет: почитайте уже готовые отчёты о дефектах, перед тем как писать свои. Это может сэкономить вам много времени и сил.

ЛОГИКА СОЗДАНИЯ ЭФФЕКТИВНЫХ ОТЧЁТОВ О ДЕФЕКТАХ

При создании отчёта о дефекте рекомендуется следовать следующему алгоритму:

1. Обнаружить дефект.
2. Понять суть проблемы.
3. Воспроизвести дефект.
4. Проверить наличие описания найденного вами дефекта в системе управления дефектами.
5. Сформулировать суть проблемы в виде «что сделали, что получили, что ожидали получить».
6. Заполнить поля отчёта, начиная с подробного описания.
7. После заполнения всех полей внимательно перечитать отчёт, исправить неточности и добавить подробности.
8. Ещё раз перечитать отчёт, т.к. в пункте 6 вы точно что-то упустили.

ТИПИЧНЫЕ ОШИБКИ ПРИ НАПИСАНИИ ОТЧЁТОВ О ДЕФЕКТАХ

Ошибки оформления и формулировок:

- Плохие краткие описания (summary). Формально, эта проблема относится к оформлению, но фактически она куда опаснее: чтение отчёта о дефекте и осознание сути проблемы начинается именно с краткого описания. Суть отчёта о дефекте: ответить на вопросы «что?», «где?», «при каких условиях?».
- Идентичные краткие и подробные описания (summary и description). Да, изредка бывают настолько простые дефекты, что для них достаточно одного краткого описания (например, «опечатка в имени пункта главного меню “File” (сейчас “Fille”)»), но, если дефект связан с неким более-менее сложным поведением приложения, стоит продумать как минимум три способа описания проблемы:
 - краткий для поля «краткое описание» (его лучше формулировать в самом конце размышлений);
 - подробный для поля «подробное описание» (поясняющий и расширяющий информацию из «краткого описания»);
 - ещё один краткий для последнего шага в шагах по воспроизведению дефекта.

- Отсутствие в подробном описании явного указания фактического результата, ожидаемого результата и ссылки на требование, если они важны, и их представляется возможным указать.
- Игнорирование кавычек, приводящее к искажению смысла. Как Вы поймёте такое краткое описание, как «запись исчезает при наведении мыши»? Какая-то запись исчезает при наведении мыши? А вот и нет. Оказывается, «поле "Запись" исчезает при наведении мыши». Даже если не дописать слово «поле», кавычки подскажут, что имеется в виду имя собственное, т.е. название некоего элемента.
- Общие проблемы с формулировками фраз на русском и английском языках.
- Лишние пункты в шагах воспроизведения.
- Копии экрана в виде «копий всего экрана целиком». Чаще всего нужно сделать копию какого-то конкретного окна приложения, а не всего экрана, тогда поможет Alt+PrintScreen. Даже если важно захватить больше, чем одно окно, практически любой графический редактор позволяет отрезать ненужную часть картинки.
- Копии экрана, на которых не отмечена проблема. Если обвести проблемную область красной линией, то это в разы повысит скорость и простоту понимания сути проблемы в большинстве случаев.
- Откладывание написания отчёта «на потом». Стремление сначала найти побольше дефектов, а уже потом их описывать приводит к тому, что какие-то важные детали (а иногда и сами дефекты!) забываются. Если «на потом» измеряется не минутами, а часами или даже днями, то проектная команда не получает важную информацию вовремя. Вывод простой: описывайте дефект сразу же, как только обнаружили его.
- Пунктуационные, орфографические, синтаксические и им подобные ошибки.

Логические ошибки:

- Выдуманные дефекты. Одной из наиболее обидных для тестировщика причин отклонения отчёта о дефекте является так называемое «описанное поведение не является дефектом» («not a bug»), когда по какой-то причине корректное поведение приложения описывается как ошибочное.
- Отнесение расширенных возможностей приложения к дефектам. Самым ярким примером этого случая является описание как дефекта того факта, что приложение запускается под операционными системами, не указанными явно в списке поддерживаемых. Лишь в очень редких случаях при разработке неких системных утилит и тому подобного ПО, крайне зависящего от версии ОС и потенциально способного «поломать» неподдерживаемую, это можно считать ошибкой (с точки зрения общего здравого смысла, такое приложение действительно должно показывать предупреждение или даже сообщение об ошибке и завершать работу на неподдерживаемой ОС).
- Неверно указанные симптомы. Это не смертельно и всегда можно подправить, но, если изначально отчёты будут сгруппированы по симптомам, это создаёт множество раздражающих неудобств.
- Чрезмерно заниженные (или завышенные) важность и срочность. С этой бедой достаточно эффективно борются проведением общих собраний и пересмотром отчётов о дефектах силами всей команды (или хотя бы нескольких человек), но, если эти показатели занижены именно чрезмерно, есть высокая вероятность, что пройдёт очень много времени, прежде чем до такого отчёта просто дойдёт очередь на следующих собраниях по пересмотру.

- Концентрация на мелочах в ущерб главному. Здесь стоит упомянуть хрестоматийный пример, когда тестировщик нашёл проблему, приводящую к краху приложения с потерей пользовательских данных, но записал её как косметический дефект (в сообщении об ошибке, которое «перед смертью» показывало приложение, была опечатка). Всегда думайте о том, как произошедшая с приложением неприятность повлияет на пользователей, какие сложности они могут из-за этого испытать, насколько это для них важно, тогда шанс увидеть реальную проблему резко повышается.
- Техническая безграмотность. Представьте себе такое краткое описание (оно же идентично продублировано в подробном, т.е. это и есть всё описание дефекта): «Количество найденных файлов не соответствует реальной глубине вложенности каталога». А что должно? Это ведь почти то же самое, что «цвет кошки не соответствует её размеру».
- Указание в шагах воспроизведения неважной для воспроизведения ошибки информации. Стремление прописать всё максимально подробно иногда принимает нездоровую форму, когда в отчёт о дефекте начинает попадать чуть ли не информация о погоде за окном и курс национальной валюты.
- Отсутствие в шагах воспроизведения информации, важной для воспроизведения дефекта.
- Игнорирование «последовательных дефектов». Иногда один дефект является следствием другого (допустим, файл повреждается при передаче на сервер, а затем приложение некорректно обрабатывает этот повреждённый файл). Да, если файл будет передан без повреждений, то второй дефект может не проявиться. Но может и проявиться в другой ситуации, т.к. проблема никуда не исчезла: приложение некорректно обрабатывает повреждённые файлы. Потому стоит описать оба дефекта.

ИНСТРУМЕНТЫ УПРАВЛЕНИЯ ОТЧЁТАМИ О ДЕФЕКТАХ

Инструментальных средств управления отчётами о дефектах (bug tracking system, defect management tool) очень много, к тому же, многие компании разрабатывают свои внутренние средства решения этой задачи. Зачастую такие инструментальные средства являются частями инструментальных средств управления тестированием.

Общий набор функций багтрекинг-систем:

- создание отчётов о дефектах, управление их жизненным циклом с учётом контроля версий, прав доступа и разрешённых переходов из состояния в состояние;
- сбор, анализ и предоставление статистики в удобной для восприятия человеком форме;
- рассылка уведомлений, напоминаний и иных артефактов соответствующим сотрудникам;
- организация взаимосвязей между отчётами о дефектах, тест-кейсами, требованиями и анализ таких связей с возможностью формирования рекомендаций;
- подготовка информации для включения в отчёт о результатах тестирования;
- интеграция с системами управления проектами.



ТЕСТОВАЯ ДОКУМЕНТАЦИЯ

Создание тестовой документации является вторым этапом жизненного цикла ПО.

Тестовая документация включает в себя:

- тест план;
- тестовая стратегия;
- чек-лист;
- тестовый сценарий;
- тестовый комплект;
- пользовательская история (User Story);
- отчет о дефекте.

ТЕСТ ПЛАН (TEST PLAN)

Тест план (Test Plan) - это документ, описывающий весь объем работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.

Хороший тест план должен описывать следующее:

- Что надо тестировать? Описание объекта тестирования: системы, приложения, оборудования.
- Что будете тестировать? Список функций и описание тестируемой системы и её компонент в отдельности.
- Как будете тестировать? Стратегия тестирования, а именно: виды тестирования и их применение по отношению к объекту тестирования.
- Когда будете тестировать? Последовательность проведения работ: подготовка (Test Preparation), тестирование (Testing), анализ результатов (Test Result Analysis) в разрезе запланированных фаз разработки.

Критерии начала тестирования:

- готовность тестовой платформы (тестового стенда);
- законченность разработки требуемого функционала;
- наличие всей необходимой документации.



Критерии окончания тестирования - результаты тестирования удовлетворяют критериям качества продукта:

- требования к количеству открытых багов выполнены;
- выдержка определенного периода без изменения исходного кода приложения Code Freeze (CF);
- выдержка определенного периода без открытия новых багов Zero Bug Bounce (ZBB).

Преимущества тест плана:

- Возможность приоритезации задач по тестированию.
- Построение стратегии тестирования, согласованной со всей командой.
- Возможность вести учет всех требуемых ресурсов (как технических, так и человеческих).
- Планирование использования ресурсов на тестирование.
- Просчет рисков, возможных при проведении тестирования.
- Составляющей частью планирования тестирования (как отдельного документа или же процесса планирования в целом) является стратегия тестирования. Стратегия может быть:
 - Частью общего тест-плана.
 - Отдельным документом.

ТЕСТОВАЯ СТРАТЕГИЯ

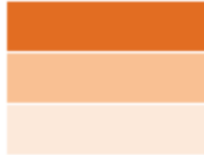
Тестовая стратегия определяет то, как мы тестируем продукт. Это набор мыслей и идей, которые направляют процесс тестирования.

В стратегии тестирования описывают:

- Тестовую среду.
- Анализ рисков проекта.
- Инструменты, которые будут использовать, чтобы провести автоматизированное тестирование и для других целей.
- План действий при непредвиденных обстоятельствах.
- Стратегия может быть представлена как в виде традиционно расписанного документа, так и в более наглядном формате, например, используя таблицу

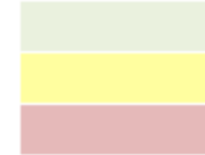
Приоритет объектов системы и действий

Высокий
Средний
Низкий



Готовность тестов

Готово
В процессе
Не готово



Объект системы и действия	Стратегия	Функциональное тестирование	Нефункциональное тестирование				
			Юзабилити	Нагрузка	Скорость	Безопасность	Окружения
		Тесты	Тесты	Тесты	Тесты		Тесты
Объект 1	Т.к. это важный объект (один из ключевых объектов с высочайшим приоритетом), нам требуется детальное тестирование и подготовка тест-анализа на уровне S&T. После подготовки, эти тесты необходимо задокументировать, т.к. мы будем часто их проходить.	link	link	link		link	
Действие 1	Одно из ключевых действий. Требуется ТА по ДПЗ и согласование этого ТА со всей командой. Чаще всего здесь сразу вкратце выписывается, что именно планируем тестировать, какие параметры?	link			link	link	link
Действие 2	Одно из ключевых действий. Требуется ТА по ДПЗ и согласование этого ТА со всей командой.	link	link	link			
Действие 3	Важное действие, делаем ТА без согласования	link	link	link			
Действие 4	Сделаем полноценное ТА, если будет время, пока - исследовательское тестирование						
Объект 2	S&T						
Действие 1	Важное действие, делаем ТА без согласования	link	link	link	link		link
Действие 2	Сделаем полноценное ТА, если будет время, пока - исследовательское тестирование						


В общем, план тестирования устанавливает цели процесса тестирования, он определяет, что будет проверяться, а стратегия тестирования описывает, как достичь целей, поставленных в плане тестирования.

ПОЛЬЗОВАТЕЛЬСКИЕ ИСТОРИИ

- Пользовательские истории (User Story) — способ описания требований к разрабатываемой системе, сформулированных как одно или более предложений на повседневном или деловом языке пользователя. Пользовательские истории используются гибкими методологиями разработки программного обеспечения для спецификации требований.
- User Story — это короткая формулировка намерения, описывающая что-то, что система должна делать для пользователя.
- Пользовательская история фиксирует короткую формулировку функции на карточке или, возможно, с помощью онлайн-средства.

Примеры:

- Залогиниться в мой портал мониторинга энергопотребления.
- Посмотреть ежедневный уровень потребления.
- Проверить мой текущий тариф.



Несмотря на то, что стори играют в огромной степени роль, ранее принадлежавшую спецификациям требований (SRS), сценариям использования и т. п., они все же ощутимо отличаются рядом тонких, но критических нюансов:

- Они не являются детальным описанием требований (то-есть того, что система должна бы делать), а представляют собой скорее обсуждаемое представление намерения (нужно сделать что-то вроде этого).
- Они являются короткими и легко читаемыми, понятными разработчикам, стейкхолдерам и пользователям.
- Они представляют собой небольшие инкременты ценной функциональности, которая может быть реализована в рамках нескольких дней или недель.
- Они относительно легко поддаются эстимированию, таким образом, усилия, необходимые для реализации, могут быть быстро определены.
- Они не занимают огромных, громоздких документов, а скорее организованы в списки, которые легче упорядочить и переупорядочить по ходу поступления новой информации.
- Они не детализированы в самом начале проекта, а уже более детально разрабатываются «точно в срок», избегая таким образом слишком ранней определенности, задержек в разработке, нагромождения требований и чрезмерно ограниченной формулировки решения.
- Они требуют минимум или вовсе не требуют сопровождения и могут быть безопасно отменены после имплементации.

Структура юзер стори

Текст самой юзер стори должен объяснять роль/действия юзера в системе, его потребность и профит, который юзер получит после того как история случится.

- К примеру: Как, <роль/персона юзера>, я <что-то хочу получить>, <с такой-то целью> .

Правила на написание User Story

- Есть один actor.
- Есть одно действие.
- Есть одна ценность / value / impact.

Actor

Вы выделили персоны, или у вас есть роли, и вы легко их вписываете в начало истории. Есть одна проблема. Убери часть истории про актера. Если история ничего при этом не потеряла - значит эта часть бесполезна.

Джеф Паттон предлагает следующее:

- Разделите всех актеров на группы. Целевая группа, важная группа, менее важная группа и тп.
- Дайте уникальные названия актерам в этих группах. Даже если в системе у них будет одинаковые роли "Пользователя системы".
- Пишите истории с точки зрения этих актеров указывая их уникальные названия.
- В результате вы сможете визуальнo увидеть какие истории необходимы для актеров целевой группы, какие - для каждой группы и тп. Вы не просто можете использовать это при разборе истории и выстраивания анализа вокруг указанного актера. Вы сможете более правильно выстроить приоритет, так как истории актеров целевой группы для нас более важны.

Действие

- Это суть истории, "что нужно сделать". Что можно улучшить. Действие должно быть одно - основное. Нет смысла описывать "авторизуется и выполняется поиск" или "указывает параметры поиска и выполняет поиск". Укажите то действие, что вам действительно нужно.
- Важно описывать историю на уровне "ЧТО?" делает, а не "КАК?". Это главное в истории. Опишите проблему, а не ее решение. Лучше вы потом с командой это обсудите и найдете более оптимальное "КАК" - решение.

Ценность

- Главная проблема с User Story. Вы всегда знаете первую часть истории, но всегда сложно указать для чего это делается. Но это Scrum, все должно быть указано как User story согласно шаблону, и потому вы пишете "чтобы ...".
- Уберите эту часть из истории. Если ничего не потеряли - значит формализация ценности в истории была бесполезна. Что же можно сделать?
- Отказаться от формулировки "чтобы". Для каких-то историй можно указать ценность истории в таком формате, но не для большинства.
- Перейти с понятия ценности (value) на влияние (impact). Ваша история не обязательно должна иметь ценность, но обязательно должна оказывать влияние на кого актера, что указан в истории. А уже это влияние ведет в конечном итоге к цели, которая имеет для вас ценность.
- Представим что вы создали историю - "Как инвестиционный аналитик я получаю отчет №17 об инвестициях чтобы БЫСТРЕЕ принять решение".
- У меня Acceptance Criteria - это метрика на value в US. Как померить такой value? Как понять что аналитик принял решение быстрее? Как вы поймете в конце что история выполнена?
- Переделаем историю на влияние - "Как инвестиционный аналитик я получаю отчет №17 об инвестициях БЫСТРЕЕ". То есть сейчас этот отчет формируется за 60 сек. Вы указываете в АС что отчет должен формироваться за 15 сек. В конце понятно выполнено ли АС, понятно какие влияние вы оказали на работу аналитика.

Практические советы по написанию пользовательских историй

- Лучше написать много историй поменьше, чем несколько громоздких.
- Каждая история в идеале должна быть написана избегая технического жаргона—чтобы клиент мог приоритезировать истории и включать их в итерации.
- Истории должны быть написаны таким образом, чтобы их можно было протестировать
- Тесты должны быть написаны до кода.
- Как можно дольше стоит избегать UI. История должна выполняться без привязки к конкретным элементам.
- Каждая история должна содержать оценку.
- История должна иметь концовку—т.е. приводить к конкретному результату.
- История должна вмещаться в итерацию.

ЧЕК-ЛИСТЫ

Чек-лист - набор идей по тестированию, разработке, планированию и управлению. А также, это перечень формализованных тестовых случаев в удобном для проведения проверок виде. Тестовые случаи в чек-листе не должны быть зависимыми друг от друга.

Обязательно должен содержать в себе следующую информацию:

- идея проверок;
- набор входных данных;
- ожидаемые результаты;
- булевая отметка о прохождении/непрохождении тестового случая;
- булевая отметка о совпадении/несовпадении фактического и ожидаемого результата по каждой проверке.
- Может также содержать шаги для проведения проверки, данные об особенностях окружения и прочую информацию необходимую для проведения проверок.

Цель – обеспечить стабильность покрытия требований проверками необходимыми и достаточными для заключения о соответствии им продукта. Особенностью является то, что чек-листы komponуются теми тестовыми случаями, которые показательны для определенного требования.

Чек-лист, чаще всего, представляет собой обычный и привычный нам список, который может быть:

- Списком, в котором последовательность пунктов не имеет значения (например, список значений некоего поля);
- Списком, в котором последовательность пунктов важна (например, шаги в краткой инструкции).
- Структурированным (многоуровневым) списком (вне зависимости от учёта последовательности пунктов), что позволяет отразить иерархию идей.

Чек-лист должен обладать рядом важных свойств:

- Логичность. Чек-лист пишется не «просто так», а на основе целей и для того, чтобы помочь в достижении этих целей. К сожалению, одной из самых частых и опасных ошибок при составлении чек-листа является превращение его в свалку мыслей, которые никак не связаны друг с другом.
- Последовательность и структурированность. Со структурированностью всё достаточно просто - она достигается за счёт оформления чек-листа в виде многоуровневого списка. Что касается последовательности, то, даже в том случае, когда пункты чек-листа не описывают цепочку действий, человеку всё равно удобнее воспринимать информацию в виде неких небольших групп идей, переход между которыми является понятным и очевидным (например, сначала можно прописать идеи простых позитивных тест-кейсов, потом идеи простых негативных тест-кейсов, потом постепенно повышать сложность тест-кейсов, но не стоит подавать эти идеи вперемешку).
- Полнота и избыточность. Чек-лист должен представлять собой аккуратную «сухую выжимку» идей, в которых нет дублирования (которые часто появляется из-за разных формулировок одной и той же идеи) и, в то же время ничто важное не упущено.

Правила составления чек-листов:

- Одна операция.
- Пункты чек-листа - это минимальные полные операции. Например, заказать изготовление визиток и доставить визитки в офис - это 2 разных операции. Поэтому в чек-листе они отображаются отдельными пунктами: визитки заказаны и визитки доставлены в офис.
- Пункты пишутся в утвердительной форме. Цель чек-листа – проверка готовности задачи, поэтому лучше составлять пункты в утвердительной форме - «заказаны, доставлены». Сравните формулировку: «заказать визитки» и «визитки заказаны».
- Оптимальное количество пунктов - до 20. Чек-листы не должны быть длинными. Если все же это требуется, то лучше разбить задачу на несколько этапов и составить к каждому этапу отдельный чек-лист.

Преимущества использования чек-листов:

- Структурирование информации у сотрудника. При записи необходимых действий у сотрудника чётко вырисовывается нужная последовательность задач.
- Повышение скорости обучения новых сотрудников. Не нужно повторять несколько раз последовательность операций. Достаточно провести короткий инструктаж и дать чек-лист для самостоятельной работы.
- Высокий результат, уменьшение количества ошибок. Как уже говорилось ранее, чек-листы помогают избежать проколов и ошибок по невнимательности.
- Взаимозаменяемость сотрудников.
- Экономия рабочего времени. Сотрудники будут значительно меньше времени тратить на переделывание задач.

ТЕСТ КЕЙСЫ

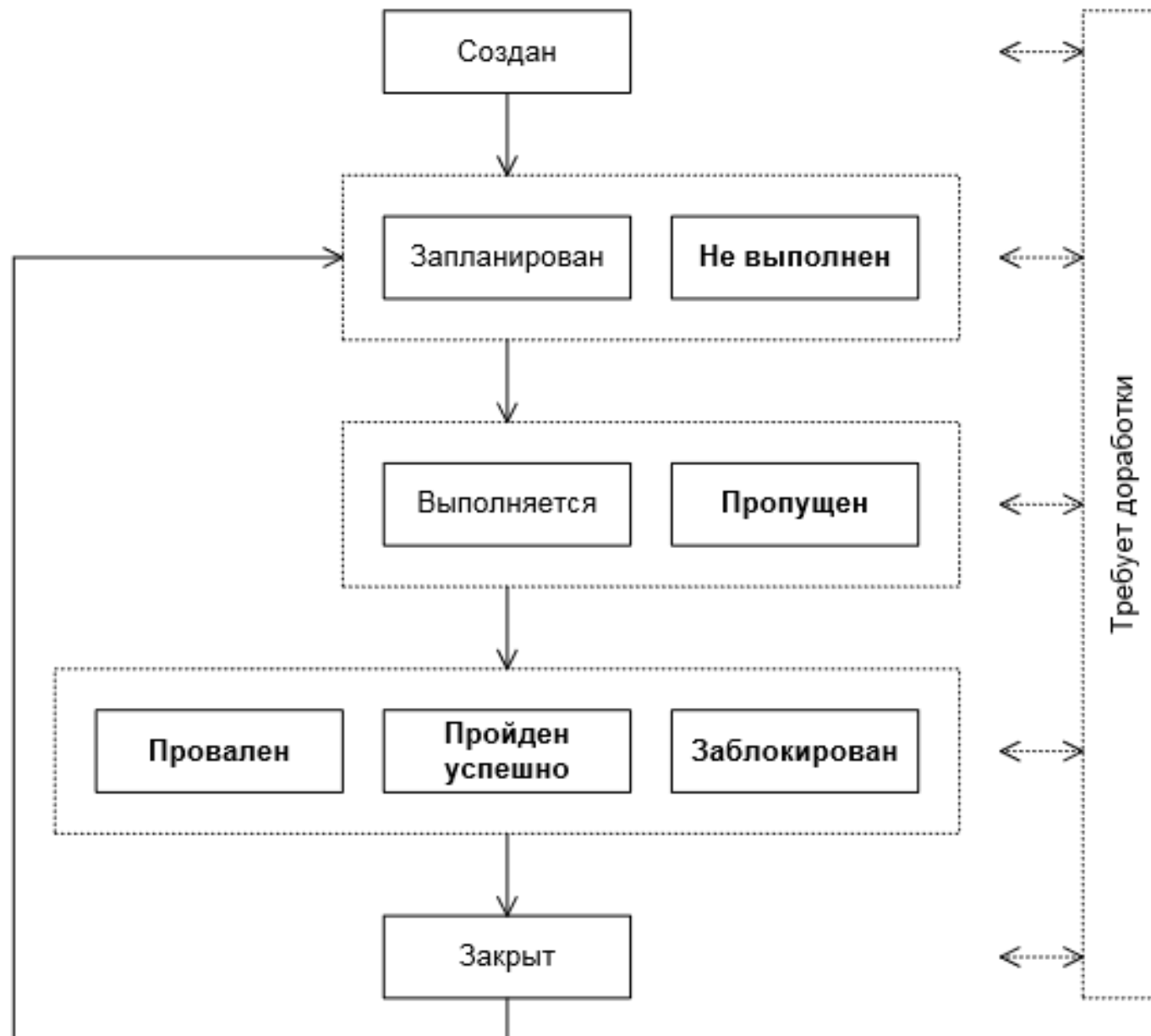
- Тестовый случай (Test Case) - это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части.
- Высокоуровневый тест-кейс - тест-кейс без конкретных входных данных и ожидаемых результатов. Как правило, ограничивается общими идеями и операциями, схож по своей сути с подробно описанным пунктом чек-листа. Достаточно часто встречается в интеграционном тестировании и системном тестировании, а также на уровне дымового тестирования. Может служить отправной точкой для проведения исследовательского тестирования или для создания низкоуровневых тест-кейсов.
- Низкоуровневый тест-кейс - тест-кейс с конкретными входными данными и ожидаемыми результатами. Представляет собой полностью готовый к выполнению тест-кейс и является наиболее классическим видом тест-кейсов. Начинающих тестировщиков чаще всего учат писать именно такие тесты, поскольку прописать все данные подробно намного проще, чем понять, какой информацией можно пренебречь, при этом не снизив ценность тест-кейса.
- Спецификация тест-кейса - документ, описывающий набор тест-кейсов (включая их цели, входные данные, условия и шаги выполнения, ожидаемые результаты) для тестируемого элемента.
- Спецификация теста - документ, состоящий из спецификации тест-дизайна, спецификации тест-кейса (test case specification) и/или спецификации тест-процедуры (test procedure specification).
- Тест-сценарий (test scenario, test procedure specification, test script) - документ, описывающий последовательность действий по выполнению теста (также известен как «тест-скрипт»).

Цель написания тест-кейсов:

Тестирование можно проводить и без тест-кейсов (не нужно, но можно; да, эффективность такого подхода варьируется в очень широком диапазоне, в зависимости от множества факторов). Наличие же тест-кейсов позволяет:

- Структурировать и систематизировать подход к тестированию (без чего крупный проект почти гарантированно обречён на провал).
- Вычислять метрики тестового покрытия (test coverage metrics) и принимать меры по его увеличению (тест-кейсы здесь являются главным источником информации, без которого существование подобных метрик теряет смысл).
- Отслеживать соответствие текущей ситуации плану (сколько примерно понадобится тест-кейсов, сколько уже есть, сколько выполнено из запланированного на данном этапе количества и т.д.).
- Уточнить взаимопонимание между заказчиком, разработчиками и тестировщиками (тест-кейсы зачастую намного более наглядно показывают поведение приложения, чем это отражено в требованиях)..
- Хранить информацию для длительного использования и обмена опытом между сотрудниками и командами (или, как минимум, не пытаться удержать в голове сотни страниц текста).
- Проводить регрессионное тестирование и повторное тестирование (которые без тест-кейсов было бы вообще невозможно выполнить).
- Повышать качество требований (написание чек-листов и тест-кейсов - хорошая техника тестирования требований).
- Быстро вводить в курс дела нового сотрудника, недавно подключившегося к проекту.

- Жизненный цикл тест-кейса
- В отличие от отчёта о дефекте, у которого есть полноценный развитый жизненный цикл, для тест-кейса речь идёт о наборе состояний, в которых он может находиться



- Создан (new) - типичное начальное состояние практически любого артефакта. Тест-кейс автоматически переходит в это состояние после создания.
- Запланирован (planned, ready for testing) - в этом состоянии тест-кейс находится, когда он или явно включён в план ближайшей итерации тестирования, или, как минимум, готов для выполнения.
- Не выполнен (not tested) - в некоторых системах управления тест-кейсами это состояние заменяет собой предыдущее («запланирован»). Нахождение тест-кейса в данном состоянии означает, что он готов к выполнению, но ещё не был выполнен.
- Выполняется (work in progress) - если тест-кейс требует длительное время для выполнения, то он может быть переведён в это состояние для подчёркивания того факта, что работа идёт, и скоро можно ожидать её результатов. Если выполнение тест-кейса занимает мало времени, это состояние, как правило, пропускается, а тест-кейс сразу переводится в одно из трёх следующих состояний - «провален», «пройден успешно» или «заблокирован».
- Пропущен (skipped) - бывают ситуации, когда выполнение тест-кейса отменяется по соображениям нехватки времени или изменения логики тестирования.

- Провален (failed) - данное состояние означает, что в процессе выполнения тест-кейса был обнаружен дефект, заключающийся в том, что ожидаемый результат по как минимум одному шагу тест-кейса не совпадает с фактическим результатом. Если в процессе выполнения тест-кейса был «случайно» обнаружен дефект, никак не связанный с шагами тест-кейса и их ожидаемыми результатами, тест-кейс считается пройденным успешно (при этом, естественно, по обнаруженному дефекту создаётся отчёт о дефекте).
- Пройден успешно (passed) - данное состояние означает, что в процессе выполнения тест-кейса не было обнаружено дефектов, связанных с расхождением ожидаемых и фактических результатов его шагов.
- Заблокирован (blocked) - данное состояние означает, что по какой-то причине выполнение тест-кейса невозможно (как правило, такой причиной является наличие дефекта, не позволяющего реализовать некий пользовательский сценарий).
- Закрыт (closed) - очень редкий случай, т.к. тест-кейс, как правило, оставляют в состояниях «провален / пройден успешно / заблокирован / пропущен». В некоторых системах управления тест-кейс переводят в данное состояние, чтобы подчеркнуть тот факт, что на данной итерации тестирования все действия с ним завершены.
- Требуется доработки (not ready) - как видно из схемы, в это состояние (или из него) тест-кейс может быть переведён в любой момент времени, если в нём будет обнаружена ошибка, если изменятся требования, по которым он был написан, или наступит иная ситуация, не позволяющая считать тест-кейс пригодным для выполнения и перевода в иные состояния.

СТРУКТУРА ТЕСТ КЕЙСА

Идентификатор	Приоритет	Связанное с тест-кейсом требование	Заглавие (суть) тест-кейса	Ожидаемый результат по каждому шагу тест-кейса
UG_U1.12	A	R97	Галерея	Загрузка файла
Модуль и подмодуль приложения			Галерея, загрузка файла, имя со спец-символами	1. Появляется окно загрузки картинки. 2. Появляется диалоговое окно браузера выбора файла для загрузки. 3. Имя выбранного файла появляется в поле «Файл».
Исходные данные, необходимые для выполнения тест-кейса			Приготовление: создать непустой файл с именем #\$\$%^&.jpg.	4. Диалоговое окно файла закрывается, в поле «Файл» появляется полное имя файла.
Шаги тест-кейса			1. Нажать кнопку «Загрузить картинку».	5. Выбранный файл появляется в списке файлов галереи.
			2. Нажать кнопку «Выбор».	
			3. Выбрать из списка подготовленный файл.	
			4. Нажать кнопку «ОК».	
			5. Нажать кнопку «Добавить в галерею».	

- Идентификатор (identifier) представляет собой уникальное значение, позволяющее однозначно отличить один тест-кейс от другого и используемое во всевозможных ссылках. В общем случае идентификатор тест-кейса может представлять собой просто уникальный номер, но (если позволяет инструментальное средство управления тест-кейсами) может быть и куда сложнее: включать префиксы, суффиксы и иные осмысленные компоненты, позволяющие быстро определить цель тест-кейса и часть приложения (или требований), к которой он относится (например, UR216_S12_DB_Neg).
- Приоритет (priority) показывает важность тест-кейса. Он может быть выражен буквами (A, B, C, D, E), цифрами (1, 2, 3, 4, 5), словами («крайне высокий», «высокий», «средний», «низкий», «крайне низкий») или иным удобным способом. Количество градаций также не фиксировано, но, чаще всего, лежит в диапазоне от трёх до пяти.

Приоритет тест-кейса может коррелировать с:

- важностью требования, пользовательского сценария или функции, с которыми связан тест-кейс;
- потенциальной важностью дефекта, на поиск которого направлен тест-кейс;
- степенью риска, связанного с проверяемым тест-кейсом требованием, сценарием или функцией.

Основная задача этого атрибута - упрощение распределения внимания и усилий команды (более высокоприоритетные тест-кейсы получают их больше), а также упрощение планирования и принятия решения о том, чем можно пожертвовать в некоей форс-мажорной ситуации, не позволяющей выполнить все запланированные тест-кейсы.


- Связанное с тест-кейсом требование (requirement) показывает то основное требование, проверке выполнения которого посвящён тест-кейс (основное, поскольку один тест-кейс может затрагивать несколько требований). Наличие этого поля улучшает такое свойство тест-кейса, как прослеживаемость. Заполнение этого поля является не обязательным.
- Модуль и подмодуль приложения (module and submodule) указывают на части приложения, к которым относится тест-кейс, и позволяют лучше понять его цель. Идея деления приложения на модули и подмодули проистекает из того, что в сложных системах практически невозможно охватить взглядом весь проект целиком, и вопрос «как протестировать это приложение» становится недопустимо сложным. Тогда приложение логически разделяется на компоненты (модули), а те, в свою очередь, на более мелкие компоненты (подмодули). Как правило, иерархия модулей и подмодулей создаётся как единый набор для всей проектной команды, чтобы исключить путаницу из-за того, что разные люди будут использовать разные подходы к такому разделению или даже просто разные названия одних и тех же частей приложения. В реальности проще всего отталкиваться от архитектуры и дизайна приложения. Например, в уже знакомом нам приложении можно выделить такую иерархию модулей и подмодулей:

Механизм запуска:

- механизм анализа параметров;
- механизм сборки приложения;
- механизм обработки ошибочных ситуаций.

Механизм взаимодействия с файловой системой:


- механизм обхода дерева SOURCE_DIR;
- механизм обработки ошибочных ситуаций.

- 
- Заглавие (суть) тест-кейса (title) призвано упростить и ускорить понимание основной идеи (цели) тест-кейса без обращения к его остальным атрибутам.
 - Исходные данные, необходимые для выполнения тест-кейса (precondition, preparation, initial data, setup), позволяют описать всё то, что должно быть подготовлено до начала выполнения тест-кейса, например:
 - состояние базы данных;
 - состояние файловой системы и её объектов;
 - состояние серверов и сетевой инфраструктуры.
 - Шаги тест-кейса (steps) описывают последовательность действий, которые необходимо реализовать в процессе выполнения тест-кейса.

ОБЩИЕ РЕКОМЕНДАЦИИ ПО НАПИСАНИЮ ШАГОВ ТАКОВЫ:

1. Начинайте с понятного и очевидного места, не пишите лишних начальных шагов (запуск приложения, очевидные операции с интерфейсом и т.п.).
2. Даже если в тест-кейсе всего один шаг, нумеруйте его (иначе возрастает вероятность в будущем случайно «приклеить» описание этого шага к новому тексту).
3. Если вы пишете на русском языке, то используйте безличную форму (например, «открыть», «ввести», «добавить» вместо «откройте», «введите», «добавьте»), в английском языке не надо использовать частицу «to» (т.е. «запустить приложение» будет «start application», не «to start application»).
4. Соотносите степень детализации шагов и их параметров с целью тест-кейса, его сложностью, уровнем и т.д. В зависимости от этих и многих других факторов степень детализации может варьироваться от общих идей до предельно чётко прописанных значений и указаний.
5. Ссылайтесь на предыдущие шаги и их диапазоны для сокращения объёма текста (например, «повторить шаги 3–5 со значением...»).
6. Пишите шаги последовательно, без условных конструкций вида «если... то...».

- Ожидаемые результаты (expected results) по каждому шагу тест-кейса описывают реакцию приложения на действия, описанные в поле «шаги тест-кейса». Номер шага соответствует номеру результата.
- По написанию ожидаемых результатов можно порекомендовать следующее:
 - Описывайте поведение системы так, чтобы исключить субъективное толкование (например, «приложение работает верно» — плохо, «появляется окно с надписью...» — хорошо).
 - Пишите ожидаемый результат по всем шагам без исключения, если у вас есть хоть малейшие сомнения в том, что результат некоего шага будет совершенно тривиальным и очевидным (если вы всё же пропускаете ожидаемый результат для какого-то тривиального действия, лучше оставить в списке ожидаемых результатов пустую строку — это облегчает восприятие).
 - Пишите кратко, но не в ущерб информативности.
 - Избегайте условных конструкций вида «если... то...».



Набор тест-кейсов (test case suite, test suite, test set) - совокупность тест-кейсов, выбранных с некоторой общей целью или по некоторому общему признаку.

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.

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

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

- 
- 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: ощутимое 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 какого-то тест-кейса (приложение не пришло в ожидаемое состояние) остальные тест-кейсы по-прежнему можно выполнять.


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.

ПОДХОДЫ К СОСТАВЛЕНИЮ НАБОРОВ ТЕСТ-КЕЙСОВ:

- На основе чек-листов.
- На основе разбиения приложения на модули и подмодули. 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.
- По принципу группировки тест-кейсов для проверки некоего уровня требований или типа требований, 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: 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».
- По видам тестирования.

ТЕХНИКИ ТЕСТ-ДИЗАЙНА

- 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.
- Роли в тест дизайне:
 - 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. На большинстве проектов эти роли выполняет QA инженер.

- 
- Тестовое покрытие - это одна из метрик оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода.
 - Существуют следующие подходы к оценке и измерению тестового покрытия:
 - Покрытие требований (Requirements Coverage) - оценка покрытия тестами функциональных и нефункциональных требований к продукту, путем построения матриц трассировки (traceability matrix).
 - Покрытие кода (Code Coverage) - оценка покрытия исполняемого кода тестами, путем отслеживания непроверенных в процессе тестирования частей программного обеспечения.
 - Тестовое покрытие на базе анализа потока управления - оценка покрытия основанная на определении путей выполнения кода программного модуля и создания выполняемых тест кейсов для покрытия этих путей.

ТЕХНИКИ ТЕСТ ДИЗАЙНА:

- Эквивалентное Разделение (Equivalence Partitioning - EP). Как пример, 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.
- Анализ Граничных Значений (Boundary Value Analysis - BVA). Если взять пример выше, в качестве значений для позитивного тестирования выберем минимальную и максимальную границы (1 и 10) и значения больше и меньше границ (0 и 11). Анализ Граничных значений может быть применен к полям, записям, файлам, или к любого рода сущностям, имеющим ограничения.
- Причина / Следствие (Cause/Effect - CE). Это, как правило, 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".

- Предугадывание ошибки (Error Guessing - EG). Когда тест-аналитик 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.
- Исчерпывающее тестирование (Exhaustive Testing - ET) - это крайний случай. В пределах этой техники Вы должны 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.
- Парное тестирование (Pairwise Testing - PT) — это техника формирования наборов тестовых данных. Сформулировать суть можно, например, таким образом: формирование таких наборов данных, в которых каждое тестируемое значение каждого из проверяемых параметров хотя бы единожды сочетается с каждым тестируемым значением всех остальных проверяемых параметров.