**Виды тестовой документации**

**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) —**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:**

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

1. Возможность приоритезации задач по тестированию.
2. Построение стратегии тестирования, согласованной со всей командой.
3. Возможность вести учет всех требуемых ресурсов (как технических, так и человеческих).
4. Планирование использования ресурсов на тестирование.
5. 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:

1. Частью общего тест-плана.
2. Отдельным документом.

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

В стратегии тестирования описывают:

1. Тестовую среду.
2. Анализ рисков проекта.
3. Инструменты, которые будут использовать, чтобы провести автоматизированное тестирование и для других целей.
4. План действий при непредвиденных обстоятельствах.

Стратегия может быть представлена как в виде традиционно расписанного документа, так и в более наглядном формате, например, используя таблицу:

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> .**

П**равила на написание User Story**

1. Есть один actor.
2. Есть одно действие.
3. 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:

1. Разделите всех актеров на группы. Целевая группа, важная группа, менее важная группа и тп.
2. Дайте уникальные названия актерам в этих группах. Даже если в системе у них будет одинаковые роли «Пользователя системы».
3. Пишите истории с точки зрения этих актеров указывая их уникальные названия.
4. В результате вы сможете визуально 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 пишите «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 обязательна 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 Аcceptance Сriteria — 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-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 компонуются теми тестовыми случаями, которые показательны для определенного требования.

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

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

**Чек-лист должен обладать рядом важных свойств:**

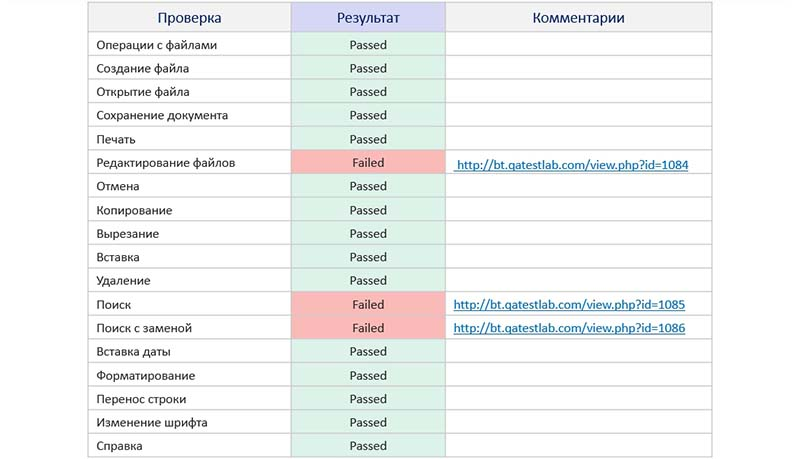
* **Логичность.**Чек-лист пишется не «просто так», а на основе целей и для того, чтобы помочь в достижении этих целей. К сожалению, одной из самых частых и опасных ошибок при составлении чек-листа является превращение его в свалку мыслей, которые никак не связаны друг с другом.
* **Последовательность 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 неизбыточность.**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.



**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»).

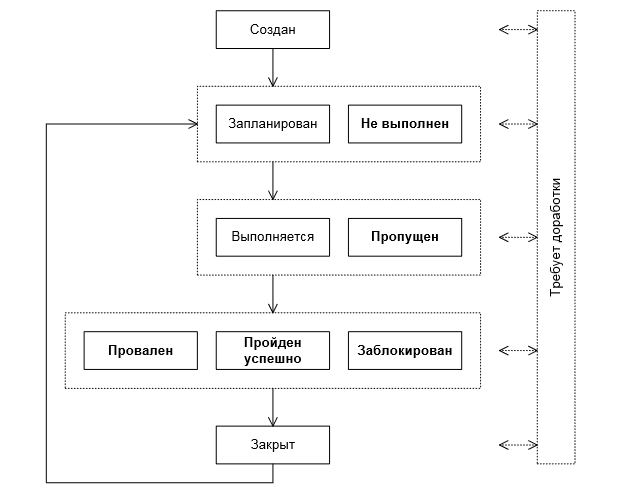
#### ****Цель написания тест-кейсов:****

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:

1. Структурировать и систематизировать подход к тестированию (без чего крупный проект почти гарантированно обречён на провал).
2. 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).
3. Отслеживать соответствие текущей ситуации плану (сколько примерно понадобится тест-кейсов, сколько уже есть, сколько выполнено из запланированного на данном этапе количества и т.д.).
4. Уточнить взаимопонимание между заказчиком, разработчиками и тестировщиками (тест-кейсы зачастую намного более наглядно показывают поведение приложения, чем это отражено в требованиях)..
5. Хранить информацию для длительного использования и обмена опытом между сотрудниками и командами (или, как минимум, не пытаться удержать в голове сотни страниц текста).
6. Проводить регрессионное тестирование и повторное тестирование (которые без тест-кейсов было бы вообще невозможно выполнить).
7. Повышать качество требований (написание чек-листов и тест-кейсов — хорошая техника тестирования требований).
8. 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).

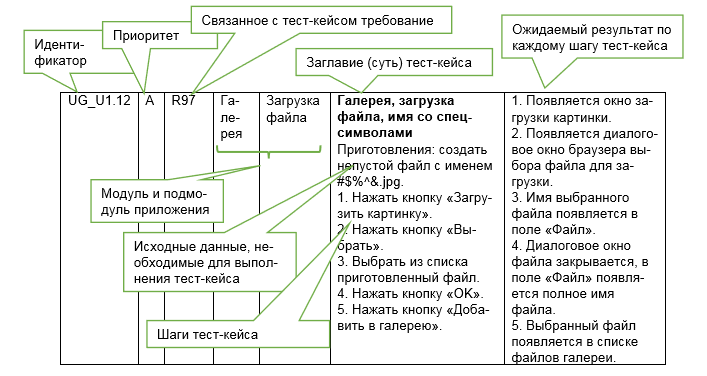
**Пройден успешно (passed)** — данное состояние означает, что в процессе выполнения тест-кейса не было обнаружено дефектов, связанных с расхождением ожидаемых и фактических результатов его шагов.

**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.

**Требует 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 (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.

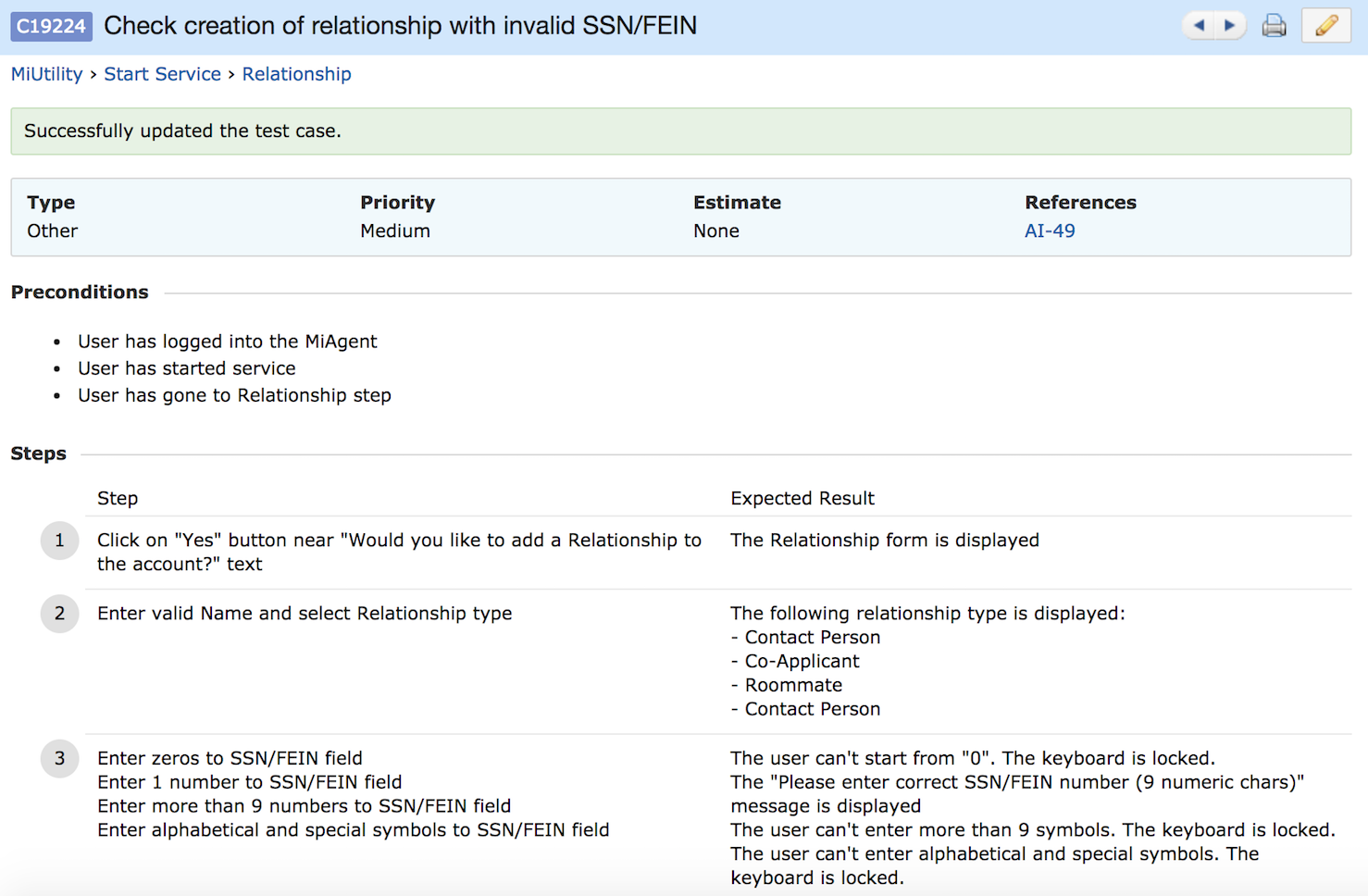
#### Общие рекомендации по написанию шагов таковы:

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:**

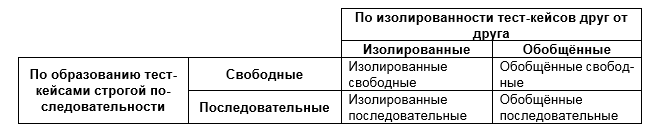
* Тест-кейсы можно выполнять в любом удобном порядке, а также создавать «наборы внутри наборов».
* Если какой-то тест-кейс завершился ошибкой, это не повлияет на возможность выполнения других тест-кейсов.

**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****



* **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).

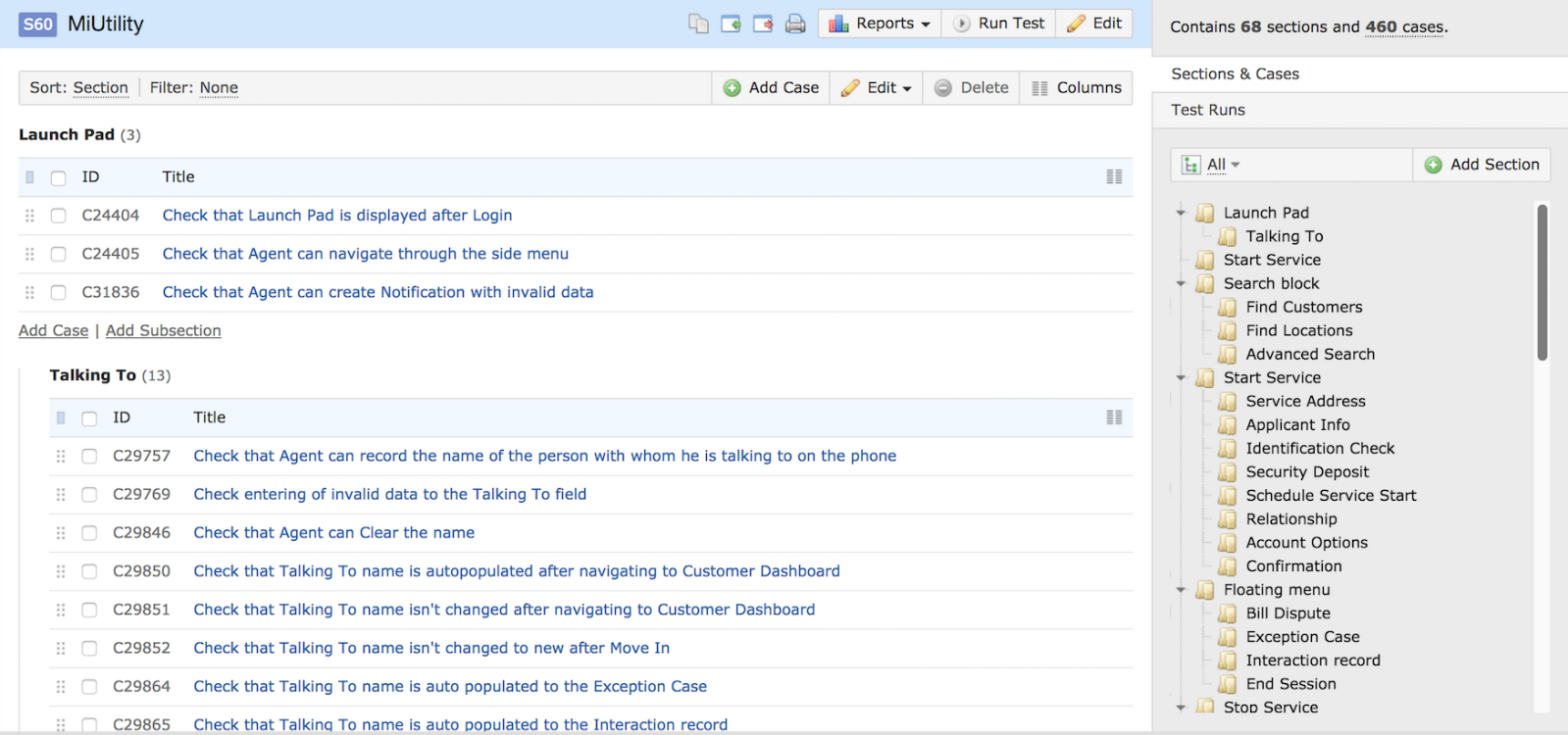
**Главное преимущество последовательности:**ощутимое сокращение шагов в каждом тест-кейсе, т.к. результат выполнения предыдущего тест-кейса является начальной ситуацией для следующего.

**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.

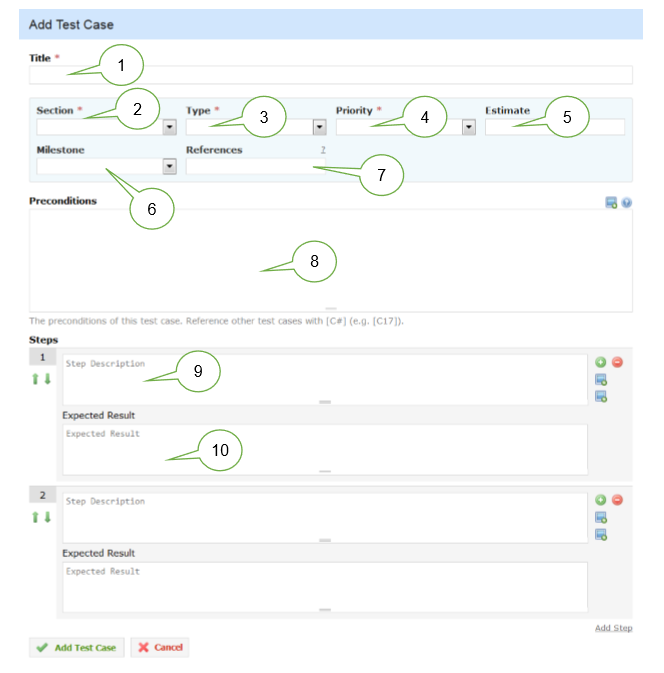
#### ****Подходы к составлению наборов тест-кейсов:****

* 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.



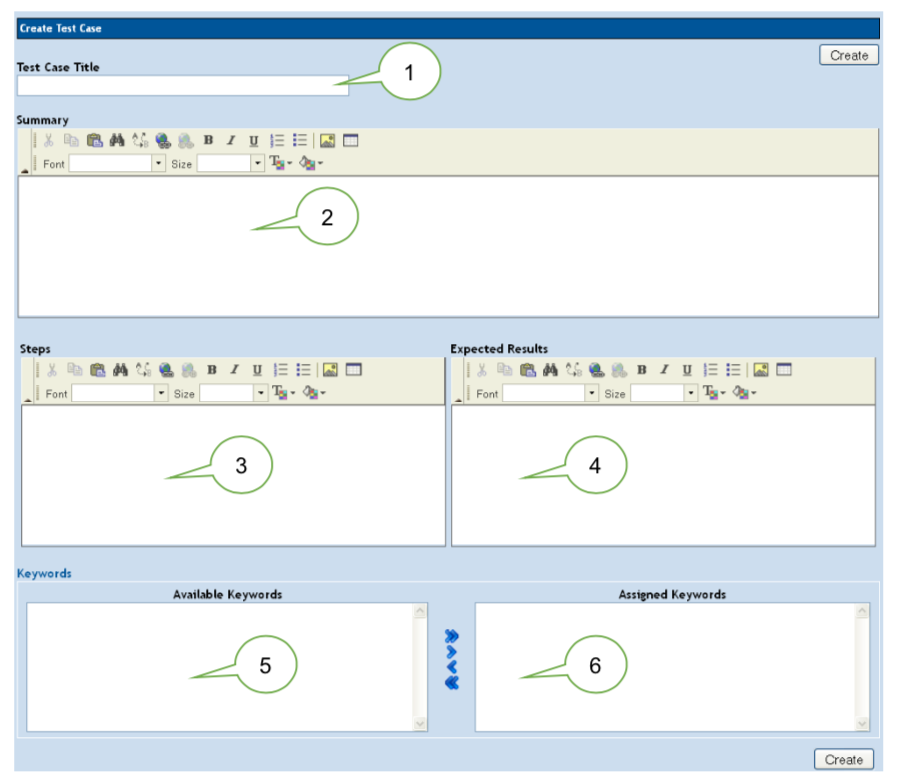
# Программное обеспечение для управления тест-кейсами

Инструментальным средством управления тест кейсами является **TestRail.**



1. **Title (заглавие)**здесь данное поле является обязательным для заполнения.
2. **Section (секция)** — очередная вариация на тему «Модуль» и «Подмодуль», позволяющая создавать иерархию секций, в которых можно размещать тест-кейсы.
3. **Type (тип)**здесь по умолчанию предлагает выбрать один из вариантов: automated (автоматизированный), functionality (проверка функциональности), performance (производительность), regression (регрессионный), usability (удобство использования), other (прочее).
4. **Priority (приоритет)**здесь представлен числами, по которым распределены следующие словесные описания: must test (обязательно выполнять), test if time (выполнять, если будет время), don’t test (не выполнять).
5. **Estimate (оценка)**содержит оценку времени, которое необходимо затратить на выполнение тест-кейса.
6. **Milestone (ключевая точка)** позволяет указать ключевую точку проекта, к которой данный тест-кейс должен устойчиво показывать положительный результат (выполняться успешно).
7. **References (ссылки)**позволяет хранить ссылки на такие артефакты, как требования, пользовательские истории, отчёты о дефектах и иные документы (требует дополнительной настройки).
8. **Preconditions (приготовления)**представляет собой классику описания предварительных условий и необходимых приготовлений к выполнению тест-кейса.
9. **Step Description (описание шага)**позволяет добавлять описание отдельного шага тест-кейса.
10. **Expected Results (ожидаемые результаты)**позволяет описать ожидаемый результат по каждому шагу.

Инструментом для управления тест кейсами является **TestLink**



1. **Title (заглавие)** здесь тоже является обязательным для заполнения.
2. **Summary (описание)** позволяет добавить любую полезную информацию о тест-кейсе (включая особенности выполнения, приготовления и т.д.).
3. **Steps (шаги выполнения)**позволяет описать шаги выполнения.
4. **Expected Results (ожидаемые результаты)** позволяет описать ожидаемые результаты, относящиеся к шагам выполнения.
5. **Available Keywords (доступные ключевые слова)**содержит список ключевых слов, которые можно проассоциировать с тест-кейсом для упрощения классификации и поиска тест-кейсов.Это ещё одна вариация идеи «Модулей» и «Подмодулей» (в некоторых системах реализованы оба механизма).
6. **Assigned Keywords (назначенные ключевые слова)**содержит список ключевых слов, проассоциированных с тест-кейсом.

#### ****JIRA + Zephyr****

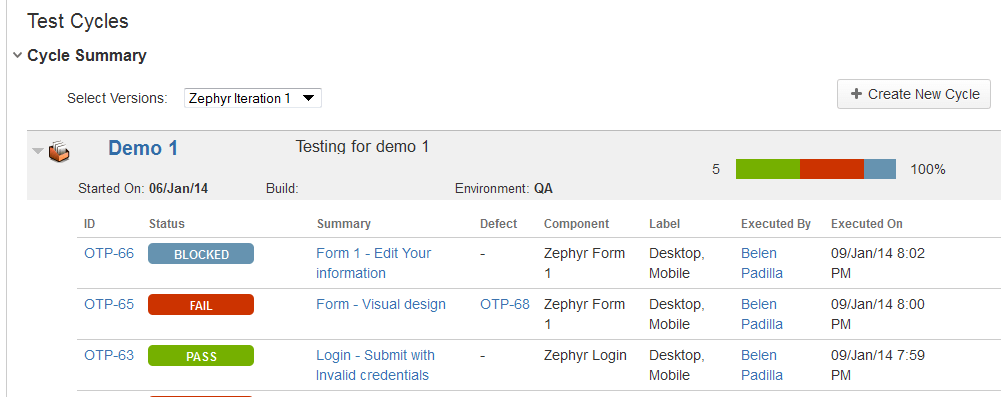
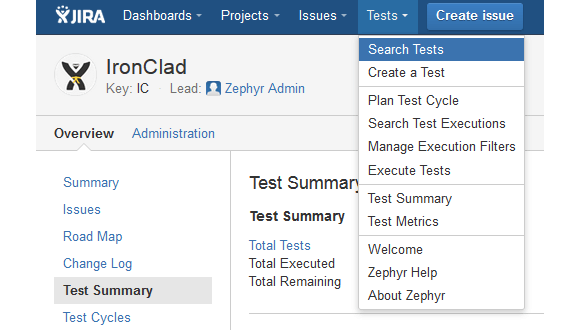
**JIRA**— это, главным образом, средство отслеживания ошибок, целью которого является контроль процесса разработки с задачами, ошибками и другими типами гибких карт.

**Zephyr** — один из многих плагинов JIRA, расширяющих возможности JIRA.

С помощью их комбинации Вы получите полный сервис, в соответствии с функциональностью инструментов управления тестированием:

* Создание тест-плана.
* Описание тестовых случаев.
* Выполнение тестирования.
* Создание отчетов.

Если тестовый продукт ведет себя неправильно, вы можете немедленно создать отчет о ошибке.



# Отчётность в тестировании

**Отчётность** — сбор и распространение информации о результатах работы (включая текущий статус, оценку прогресса и прогноз развития ситуации).

**К высокоуровневым задачам отчётности относятся:**

* Сбор, агрегация и предоставление в удобной для восприятия форме объективной информации о результатах работы.
* Формирование оценки текущего статуса и прогресса (в сравнении с планом).
* Обозначение существующих и возможных проблем (если такие есть).
* Формирование прогноза развития ситуации и фиксация рекомендаций по устранению проблем и повышению эффективности работы.

**Отчёт о результатах тестирования** — документ, обобщающий результаты работ по тестированию и содержащий информацию, достаточную для соотнесения текущей ситуации с тест-планом и принятия необходимых управленческих решений.

**К низкоуровневым задачам отчётности в тестировании относятся:**

* Оценка объёма и качества выполненных работ.
* Сравнение текущего прогресса с тест-планом (в том числе с помощью анализа значений метрик).
* Описание имеющихся сложностей и формирование рекомендаций по их устранению.
* Предоставление лицам, заинтересованным в проекте, полной и объективной информации о текущем состоянии качества проекта, выраженной в конкретных фактах и числах.

**Качественный отчёт** о результатах тестирования обладает многими свойствами качественных требований, а также расширяет их набор следующими пунктами:

* информативность (в идеале, после прочтения отчёта не должно оставаться никаких открытых вопросов о том, что происходит с проектом в контексте качества);
* точность и объективность (ни при каких условиях в отчёте не допускается искажение фактов, а личные мнения должны быть подкреплены твёрдыми обоснованиями).

Отчёт о результатах тестирования создаётся по заранее оговорённому расписанию (зависящему от модели управления проектом) при участии большинства представителей проектной команды, задействованных в обеспечении качества.  
Большое количество фактических данных для отчёта может быть легко извлечено в удобной форме из системы управления проектом. Ответственным за создание отчёта, как правило, является ведущий тестировщик («тест-лид»). При необходимости, отчёт может обсуждаться на небольших собраниях.

**Отчёт о результатах тестирования предоставляется:**

* менеджеру проекта — как источник информации о текущей ситуации и основа для принятия управленческих решений;
* руководителю команды разработчиков («дев-лиду») — как дополнительный объективный взгляд на происходящее на проекте;
* руководителю команды тестировщиков («тест-лиду») — как способ структурировать собственные мысли и собрать необходимый материал для обращения к менеджеру проекта по насущным вопросам, если в этом есть необходимость;
* заказчику — как наиболее объективный источник информации о том, что происходит на проекте, за который он платит свои деньги.

**Отчёт о результатах тестирования включает следующие разделы:**

* **Краткое описание.** В предельно краткой форме отражает основные достижения, проблемы, выводы и рекомендации. В идеальном случае, прочтения краткого описания может быть достаточно для формирования полноценного представления о происходящем, что избавит от необходимости читать весь отчёт (это важно, т.к. отчёт о результатах тестирования может попадать в руки очень занятым людям).
* **Команда тестировщиков.** Список участников проектной команды, задействованных в обеспечении качества, с указанием их должностей и ролей в подотчётный период.
* **Описание процесса тестирования.**Последовательное описание того, какие работы были выполнены за подотчётный период.
* **Расписание.** Детализированное расписание работы команды тестировщиков и/или личные расписания участников команды.
* **Статистика по новым дефектам.** Таблица, в которой представлены данные по обнаруженным за подотчётный период дефектам (с классификацией по стадии жизненного цикла и важности).
* **Список новых дефектов.** Список обнаруженных за подотчётный период дефектов с их краткими описаниями и важностью.
* **Статистика по всем дефектам.** Таблица, в которой представлены данные по обнаруженным за всё время существования проекта дефектам (с классификацией по стадии жизненного цикла и важности). Как правило, в этот же раздел добавляется график, отражающий такие статистические данные.
* **Рекомендации.**Обоснованные выводы и рекомендации по принятию тех или иных управленческих решений (изменению тест-плана, запросу или освобождению ресурсов и т.д.) Здесь этой информации можно отвести больше места, чем в кратком описании (summary), сделав акцент именно на том, что и почему рекомендуется сделать в имеющейся ситуации.
* **Приложения.** Фактические данные (как правило, значения метрик и графическое представление их изменения во времени).

**Логика построения отчёта о результатах тестирования**

Для того, чтобы отчёт о результатах тестирования был действительно полезным, при его создании следует постоянно помнить об универсальной логике отчётности:

1. Выводы строятся на основе целей (которые были отражены в плане).
2. Выводы дополняются рекомендациями.
3. Выводы, так же, как и рекомендации, строго обосновываются.
4. Обоснование опирается на объективные факты.

**Выводы должны быть:**

* Краткими.
* Информативными.
* Полезными для читающего отчёт.

**Рекомендации должны быть:**

* Краткими.
* Реально выполнимыми.
* Дающими как понимание того, что надо сделать.

**Вы это уже сделали?**

Что отвечать на такой вопрос рядовому сотруднику? Варианта два, но и два пути, которые данному работнику могут и не понравиться. Если он отвечает «да», то это означает, что он может взять еще работу на исполнение, а потом окажется, что ему нужно доделать старую. Также если он отвечает «нет», то его могут посчитать медлительным и он тормозит процесс.

Такой вопрос обычно задается бесчисленное количество раз при разработке какого-либо программного продукта, да и, в целом, во время рабочего процесса. В этих моментах, как и в любых других, должна быть оптимизация способная минимизировать или исключить полностью негативные стороны этого процесса. Вообще, понимание выражения «**Definition of Done»**в полной мере понятна людям, знакомыми с философией Scrum. Определенно сделанная задача – это та задача, которая не нуждается в доработках, но тут встает законный вопрос: «А как оценить, что задача действительно выполнена?»

Как может показаться изначально, вопрос так себе. Ну что значит «как оценить?» Выполнена — значит, выполнена, не выполнена — значит ,не выполнена.

Давайте посмотрим банальный пример про разработку чего-нибудь, например, части программы. Скажем, мы написали какой-то функционал, и вроде бы всё, однако, мы понимаем что наш функционал может иметь баги (ошибки), которые, например, сейчас не могут быть проверены, так как не готовы другие модули, позволяющие протестировать. Получается, ставя напротив этой задачи «Выполнено», мы немного лукавим, так как нам к ней придется возвращаться. Даже если есть модули для тестирования, необходимо ли тестировать? Может, следует поставить «Выполнено» только после результатов code review? Как мы видим, у нас очень много вопросов, которые мешают нам правильно построить работу. Из-за нашей неопределенности вытекают очень большие проблемы. Мало того, что мы сами можем не понять ,сделали ли мы до конца, а уж другие члены группы точно не смогут разобраться. Как Вы уже, наверное, догадались, **Definition of Done**и призвана исправить это и не дать нам повода беспокоиться!

#### ****Definition of Done на страже нашего спокойствия****

На самом деле, на страже общего спокойствия. Мы действительно можем не понять до конца степень законченности нашей задачи, однако оповестить команду, что же мы все-таки сделали, обязаны. **Definition of Done,** как и всё в Scrum, должно быть лаконично, поэтому зачастую отводится для этого одно предложение, однако это не единственный вариант.

**Для примера приведем несколько выполненных задач с использованием Definition of Done:**

* Done = функционал оплаты реализован, проведено тестирование с тестировщиком Алексеем
* Done = разработан документ по спецификации, проведено обсуждение с клиентами
* Done = модуль авторизации разработан полностью, протестирован, продемонстрирован на Sprint Review Meeting
* Done = модуль полностью реализован и выгружен для использования

Как видно, любое описание начинается с “done=”, что дает понять, на что обращать внимание. Вообще, в листе принято писать такие результаты, которые можно проверить. Нет смысла описывать мысли, ведь, скажем, «done = придуман внешний вид интерфейса корзины» звучит странно и никак это проверить нельзя.

Желательно, изначально разработать список правил, которые будут описывать Definition of Done, чтобы все в команде писали в одном стиле. Это приведет к более быстрому пониманию того, что хотел передать коллега.

Еще одним из знаменитых способов записи **Definition of Done**— является простой список.

В заключении, хочется сказать, что не стоит пренебрегать Definition of Done, ведь это приводит не только к осознанию того, что было сделано, но и к тому, как это нужно сделать в будущем. Благодаря использованию Definition of Done все будут стараться делать задачи более четкими и конкретными, чтобы потом гордо сказать: «Точно сделано!». Помимо этого, набор рейтинга в Velocity будет гораздо выше.

В большинстве своем, мы привыкли к графикам идущими вверх, что означает положительную динамику, однако они могут идти и вниз и также показывать положительную динамику. Одним из таких ярких примеров является Диаграмма сгорания задач (Burndown chart). Само сочетание «Burn Down» дословно переводится как «гореть вниз» и это действительно так. Данный график является основным средством для отслеживания выполненных задач в спринте или во всем проекте. Хотя, по сути, он может использоваться как угодно, но мы его рассматриваем внутри методологии Scrum.

#### ****Пример Диаграммы сгорания задач:****

Синим цветом на диаграмме сгорания отмечена идеальная линия выполнения задач, на которую и следует опираться.

Красным цветом отмечена реальная история выполнения задач.

**По шкале Y**отмечают количество запланированных баллов (в данном случае), идеальные часы, количество задач и так далее.

**По шкале X**отмечают количество дней до окончания Sprint.

Как может показаться на первый взгляд, данная диаграмма сгорания задач / Burndown chart служит всего лишь для самоконтроля и самоотчета, однако ее использование может рассказать об очень многом.

#### Читаем Диаграмму сгорания задач / Burndown chart

Начнем с примеров негативных результатов как ведения графика, так и самой работы команды и закончим более качественными.

#### 1. Burndown Chart: Слишком рано

е

По диаграмме сгорания задач/Burndown chart отчетливо видно, что команда все задачи выполнила раньше срока. Такая ситуация тоже не является позитивной, так как это означает ряд совершенных проблем:

1. Команда сделала неправильную оценку предстоящей работы.
2. В случае быстрого выполнения задач, разработчики не добавляли задачи из следующего спринта.
3. Команда сильно перестраховалась, включив изначально дополнительный срок.

В случае такой проблемы, чаще всего Scrum Master спрашивает команду о возможности добавления дополнительных задач из Product Backlog.

#### 2. Burndown Chart: Опоздали

Также один из видов негативных диаграмм сгорания задач.

Одной из возможных причин здесь может быть постоянное добавление новых задач во время спринта, что увеличило нагрузку.

Второй частой проблемой является недоделанность задач, когда задачи сделаны наполовину. Такие задачи, как выразился Джефф Сазерленд, являются хламом.

В такой ситуации, на Daily Scrum Meeting обязательно нужно говорить о проблемах, мешающих идти к цели ровной дорогой. Как только линия реальных задач пошла выше, сразу надо решать проблему – это также один из постулатов методологии Scrum.

#### 3. Burndown Chart: Без оценок

Может быть даже команда и работала, только забыла или не захотела использовать диаграмму сжигания задач, что является прямо сказать дурным тоном и противоречит эффективной работе. Команда не может контролировать себя, не может совершенствоваться и так далее.

#### 4. Burndown Chart: Конечная оценка

Собственно, ситуация равна предыдущей. Не смотря на законченный Sprint, все итоговые оценки были внесены в диаграмму сгорания в самый последний день после завершения работы. Это равносильно тому, когда вообще законченные задачи не вносятся. По данному графику невозможно сделать вывода о правильности работы команды и, даже более того, можно предположить, что команда не стремится к развитию.

#### 5. Burndown Chart: Zero

Отсутствие показателя реальных задач в диаграмме не является поводом считать, что работа не производилась, ведь она могла быть просто не оценена. Как и в предыдущих пунктах, такая позиция не позволяет контролировать работу собственной команды и совершенствоваться.

#### 6. Burndown Chart: Релаксирующая команда

Этот пример диаграммы сгорания задач уже значительно лучше, нежели другие, ведь в нем можно увидеть, как усовершенствовать команду. Возможные проблемы здесь такие же, как и в пункте «Слишком рано», но Scrum Team решили не заканчивать Sprint раньше, а более расслаблено продолжить работу, что также является ошибкой.

#### 7. Burndown Chart: Совершенствование

Scrum Team на текущих показателях выглядит достаточно хорошо. По линиям видно, что в самом начале были трудности, но во время Daily Scrum Meeting все вопросы вскрывались и Scrum Master исправлял работу, ведя команду к цели.

Также, возможно, группа делала принципиальное ускорение для достижения цели.

Еще одной причиной, к примеру, может быть то, что команда брала дополнительные задачи.

#### 8. Burndown Chart: Опыт

На лицо опытная группа, которая, после начала работы, сразу исправляет все возникающие трудности и совершенствуется так, что резко переходит к активному сжиганию.

#### 9. Burndown Chart: A++

Бесконечно можно смотреть на три вещи: как горит огонь, как течет вода и как строится идеальный график =).

## **Матрица соответствия требований (Requirements Traceability Matrix)**

Это двумерная таблица, содержащая соответствие функциональных требований (functional requirements) продукта и подготовленных тестовых сценариев (test cases). В заголовках колонок таблицы расположены требования, а в заголовках строк — тестовые сценарии. На пересечении — отметка, означающая, что требование текущей колонки покрыто тестовым сценарием текущей строки. Матрица обычно хранится в виде электронной таблицы.

Матрица соответствия требований используется QA-инженерами для валидации покрытия требований по продукту тестами.

**Цель «Traceability Matrix»:**

* какие требования «покрыты» тестами, а какие нет.
* избыточность тестов (одно функциональное требование покрыто большим количеством тестов).

Данный тестовый артефакт является неотъемлемой частью тестирования.

