Раздел 5. Тестирование

Тестирование программного обеспечения — процесс анализа программного средства и сопутствующей документации с целью выявления дефектов и повышения качества продукта.

Модель разработки ПО (Software Development Model, SDM) — структура, систематизирующая различные виды проектной деятельности, их взаимодействие и последовательность в процессе разработки ПО. Выбор той или иной модели зависит от масштаба и сложности проекта, предметной области, доступных ресурсов и множества других факторов.

Выбор модели разработки ПО серьёзно влияет на процесс тестирования, определяя выбор стратегии, расписание, необходимые ресурсы и т.д.

Моделей разработки ПО много, но в общем случае классическими можно считать водопадную, v-образную, итерационную инкрементальную, спиральную и гибкую.

Знать и понимать модели разработки ПО необходимо затем, чтобы уже с первых дней работы понимать, что происходит вокруг, что, зачем и почему вы делаете. Многие начинающие тестировщики отмечают, что ощущение бессмысленности происходящего посещает их, даже если текущие задания интересны. Чем полнее вы будете представлять картину происходящего на проекте, тем яснее вам будет виден ваш собственный вклад в общее дело и смысл того, чем вы занимаетесь.

Ещё одна важная вещь, которую следует понимать, состоит в том, что никакая модель не является догмой или универсальным решением. Нет идеальной модели. Есть та, которая хуже или лучше.

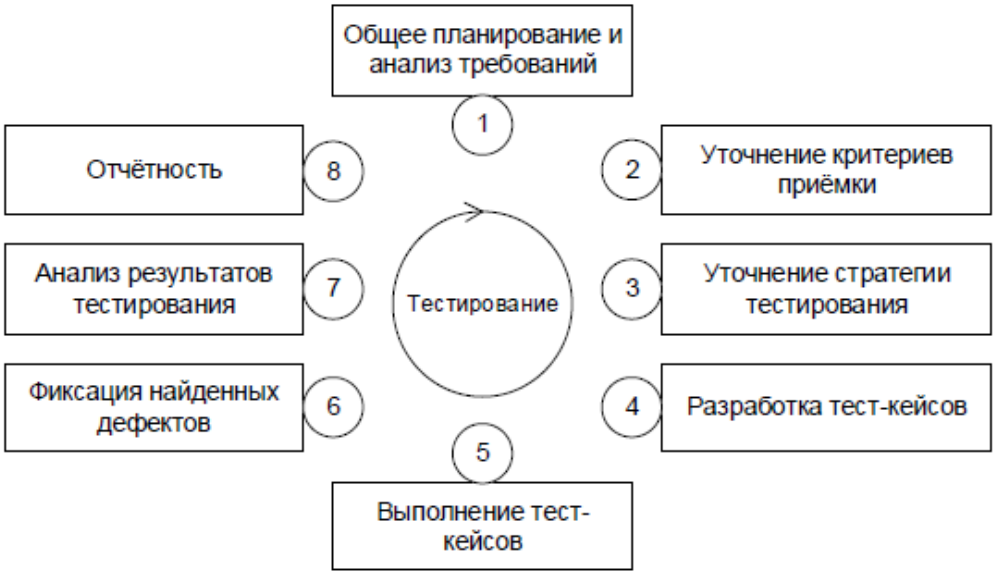
1.1 Жизненный цикл тестирования

Следуя общей логике итеративности, преобладающей во всех современных моделях разработки ПО, жизненный цикл тестирования также выражается замкнутой последовательностью действий.

Важно понимать, что длина такой итерации (и, соответственно, степень подробности каждой стадии) может варьироваться в широчайшем диапазоне — от единиц часов до десятков месяцев. Как правило, если речь идёт о длительном промежутке времени, он разбивается на множество относительно коротких итераций, но сам при этом

«тяготеет» к той или иной стадии в каждый момент времени (например, в начале проекта больше планирования, в конце — больше отчётности).

Также ещё раз подчеркнём, что приведённая схема — не догма, и вы легко можете найти альтернативы, но общая суть и ключевые принципы остаются неизменными. Их и



рассмотрим .

Стадия 1 (общее планирование и анализ требований) объективно необходима как минимум для того, чтобы иметь ответ на такие вопросы, как: что нам предстоит тестировать; как много будет работы; какие есть сложности; всё ли необходимое у нас есть и т.п. Как правило, получить ответы на эти вопросы невозможно без анализа требований, т.к. именно требования являются первичным источником ответов.

Стадия 2 (уточнение критериев приёмки) позволяет сформулировать или уточнить метрики и признаки возможности или необходимости начала тестирования (entry criteria), приостановки (suspension criteria) и возобновления (resumption criteria) тестирования, завершения или прекращения тестирования (exit criteria).

Стадия 3 (уточнение стратегии тестирования) представляет собой ещё одно обращение к планированию, но уже на локальном уровне: рассматриваются и уточняются те части стратегии тестирования (test strategy), которые актуальны для текущей итерации.

Стадия 4 (разработка тест-кейсов) посвящена разработке, пересмотру, уточнению, доработке, переработке и прочим действиям с тест-кейсами, наборами тест-кейсов, тестовыми сценариями и иными артефактами, которые будут использоваться при непосредственном выполнении тестирования.

Стадия 5 (выполнение тест-кейсов) и **стадия 6** (фиксация найденных дефектов) тесно связаны между собой и фактически выполняются параллельно: дефекты фиксируются сразу по факту их обнаружения в процессе выполнения тест-кейсов. Однако

зачастую после выполнения всех тест-кейсов и написания всех отчётов о найденных дефектах проводится явно выделенная стадия уточнения, на которой все отчёты о дефектах рассматриваются повторно с целью формирования единого понимания проблемы и уточнения таких характеристик дефекта, как важность и срочность.

Стадия 7 (анализ результатов тестирования) и **стадия 8** (отчётность) также тесно связаны между собой и выполняются практически параллельно. Формулируемые на стадии анализа результатов выводы напрямую зависят от плана тестирования, критериев приёмки и уточнённой стратегии, полученных на стадиях 1, 2 и 3. Полученные выводы оформляются на стадии 8 и служат основной для стадий 1, 2 и 3 следующей итерации тестирования. Таким образом, цикл замыкается.

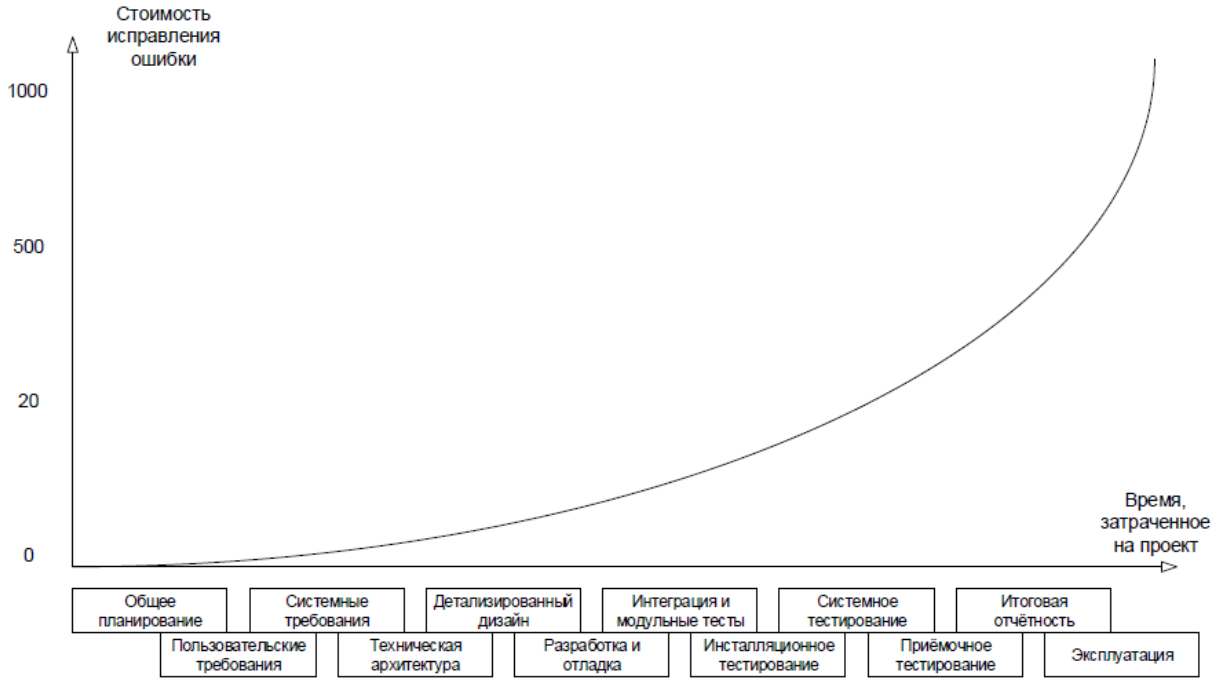
Тестирование документации и требований

Требования являются отправной точкой для определения того, что проектная команда будет проектировать, реализовывать и тестировать. Элементарная логика говорит нам, что если в требованиях что-то «не то», то и реализовано будет «не то», т.е. колоссальная работа множества людей будет выполнена впустую.

Вне зависимости от того, какая модель разработки ПО используется на проекте, чем позже будет обнаружена проблема, тем сложнее и дороже будет её решение. А в самом начале идёт планирование и работа с требованиями.

Если проблема в требованиях будет выяснена на этой стадии, её решение может свестись к исправлению пары слов в тексте, в то время как недоработка, вызванная пропущенной проблемой в требованиях и обнаруженная на стадии эксплуатации, может даже полностью уничтожить проект.

Стоимость исправления ошибки в зависимости от момента её обнаружения:



Важно подчеркнуть, что требования:

- Позволяют понять, что и с соблюдением каких условий система должна делать.
- Предоставляют возможность оценить масштаб изменений и управлять изменениями.
- Являются основой для формирования плана проекта (в том числе плана тестирования).
- Помогают предотвращать или разрешать конфликтные ситуации.
- Упрощают расстановку приоритетов в наборе задач.
- Позволяют объективно оценить степень прогресса в разработке проекта.

1.2 Источники и пути выявления требований

Требования начинают свою жизнь на стороне заказчика. Их сбор (gathering) и выявление (elicitation) осуществляются с помощью следующих основных техник (рис.1.2)

Рис. 1.2 Основные техники сбора и выявления требований

Интервью. Самый универсальный путь выявления требований, заключающийся в общении проектного специалиста (как правило, специалиста по бизнес-анализу) и представителя заказчика (или эксперта, пользователя и т.д.) Интервью может проходить в классическом понимании этого слова (беседа в виде «вопрос-ответ»), в виде переписки и т.п. Главным здесь является то, что ключевыми фигурами выступают двое — интервьюируемый и интервьюер (хотя это и не исключает наличия «аудитории слушателей», например, в виде лиц, поставленных в копию переписки).

Работа с фокусными группами. Может выступать как вариант «расширенного интервью», где источником информации является не одно лицо, а группа лиц.

Анкетирование. Этот вариант выявления требований вызывает много споров, т.к. при неверной реализации может привести к нулевому результату при объёмных затратах. В то же время при правильной организации анкетирование позволяет автоматически собрать и обработать огромное количество ответов от огромного количества респондентов. Ключевым фактором успеха является правильное составление анкеты, правильный выбор аудитории и правильное преподнесение анкеты.

Семинары и мозговой штурм. Семинары позволяют группе людей очень быстро обменяться информацией (и наглядно продемонстрировать те или иные идеи), а также хорошо сочетаются с интервью, анкетированием, прототипированием и моделированием — в том числе для обсуждения результатов и формирования выводов и решений. Мозговой штурм может проводиться и как часть семинара, и как отдельный вид деятельности. Он позволяет за минимальное время сгенерировать большое количество идей, которые в дальнейшем можно не спеша рассмотреть с точки зрения их использования для развития проекта.

Наблюдение. Может выражаться как в буквальном наблюдении за некими процессами, так и во включении проектного специалиста в эти процессы в качестве участника. С одной стороны, наблюдение позволяет увидеть то, о чём (по совершенно различным соображениям) могут умолчать интервьюируемые, анкетируемые и представители фокусных групп, но с другой — отнимает очень много времени и чаще всего позволяет увидеть лишь часть процессов.

Прототипирование. Состоит в демонстрации и обсуждении промежуточных версий продукта (например, дизайн страниц сайта может быть сначала представлен в виде картинок, и лишь затем свёрстан). Это один из лучших путей поиска единого понимания и уточнения требований, однако он может привести к серьёзным дополнительным затратам при отсутствии специальных инструментов (позволяющих быстро создавать прототипы) и слишком раннем применении (когда требования ещё не стабильны, и высока вероятность создания прототипа, имеющего мало общего с тем, что хотел заказчик).

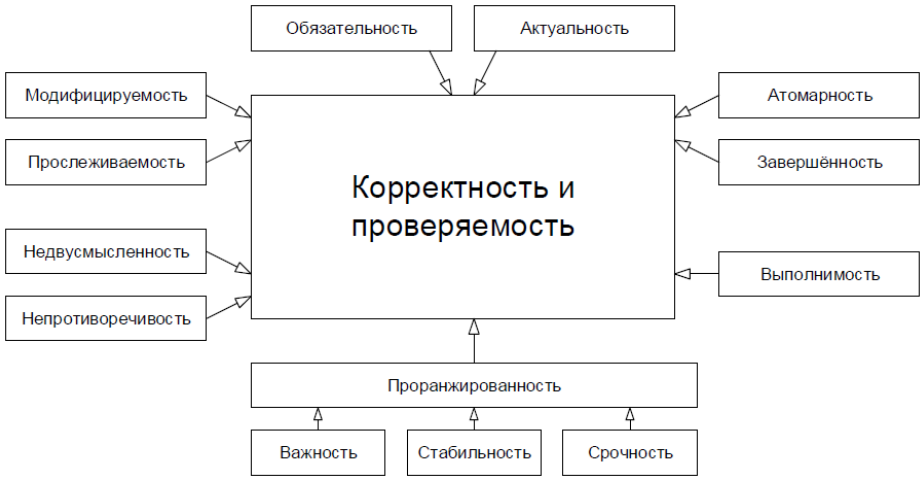
Анализ документов. Хорошо работает тогда, когда эксперты в предметной области (временно) недоступны, а также в предметных областях, имеющих общепринятую устоявшуюся регламентирующую документацию. Также к этой технике относится и просто изучение документов, регламентирующих бизнес-процессы в предметной области заказчика или в конкретной организации, что позволяет приобрести необходимые для лучшего понимания сути проекта знания.

Моделирование процессов и взаимодействий. Может применяться как к «бизнес-процессам и взаимодействиям» ,так и к «техническим процессам и взаимодействиям» Данная техника требует высокой квалификации специалиста по бизнес-анализу, т.к. сопряжена с обработкой большого объёма сложной (и часто плохо структурированной) информации.

Самостоятельное описание. Является не столько техникой выявления требований, сколько техникой их фиксации и формализации. Очень сложно (и даже нельзя!) пытаться самому «придумать требования за заказчика», но в спокойной обстановке можно самостоятельно обработать собранную информацию и аккуратно оформить её для дальнейшего обсуждения и уточнения.

Свойства качественных требований

В процессе тестирования требований проверяется их соответствие определённому набору свойств



Завершённость (completeness). Требование является полным и законченным с точки зрения представления в нём всей необходимой информации, ничто не пропущено по соображениям «это и так всем понятно».

Типичные проблемы с завершённостью:

- Указана лишь часть некоторого перечисления (например: «*экспорт осуществляется в форматы PDF, PNG и т.д.*» — что мы должны понимать под «и т.д.»?).

Атомарность, единичность (atomicity). Требование является атомарным, если его нельзя разбить на отдельные требования без потери завершённости, и оно описывает одну и только одну ситуацию.

Непротиворечивость, последовательность (consistency85). Требование не должно содержать внутренних противоречий и противоречий другим требованиям и документам. (Например, в одном месте сказано, что кнопка должна быть синей, а в другом – зеленой. Так какой?)

Недвусмысленность (unambiguousness86, clearness). Требование описано без использования жаргона, неочевидных аббревиатур и расплывчатых формулировок и допускает только однозначное объективное понимание.

Выполнимость (feasibility). Требование технологически выполнимо и может быть реализовано в рамках бюджета и сроков разработки проекта.

Обязательность, нужность (obligation) и **актуальность** (up-to-date). Если требование не является обязательным к реализации, оно должно быть просто исключено из набора требований. Если требование нужное, но «не очень важное», для указания этого факта используется указание приоритета. Также исключены или переработаны должны быть требования, утратившие актуальность.

Прослеживаемость (traceability). Прослеживаемость бывает вертикальной (vertical traceability) и горизонтальной (horizontal traceability). Вертикальная позволяет соотносить между собой требования на различных уровнях требований, горизонтальная позволяет соотносить требование с тест-планом, тест-кейсами, архитектурными решениями и т.д.
Для обеспечения прослеживаемости часто используются специальные инструменты по управлению требованиями (requirements management tool) и/или матрицы прослеживаемости (traceability matrix).

Модифицируемость (modifiability). Это свойство характеризует простоту внесения изменений в отдельные требования и в набор требований. Можно говорить о наличии модифицируемости в том случае, если при доработке требований искомую информацию легко найти, а её изменение не приводит к нарушению иных описанных в этом перечне свойств.

Проранжированность по важности, стабильности, срочности (ranked for importance, stability, priority). Важность характеризует зависимость успеха проекта от успеха реализации требования. Стабильность характеризует вероятность того, что в обозримом будущем в требование не будет внесено никаких изменений. Срочность определяет распределение во времени усилий проектной команды по реализации того или иного требования.

Корректность (correctness) и **проверяемость** (verifiability). Фактически эти свойства вытекают из соблюдения всех вышеперечисленных (или можно сказать, что они не выполняются, если нарушено хотя бы одно из вышеперечисленных). В дополнение можно отметить, что проверяемость подразумевает возможность создания объективного тест-кейса (тест-кейсов), однозначно показывающего, что требование реализовано верно и поведение приложения в точности соответствует требованию.

1.3 Виды и направления тестирования

Упрощённая классификация тестирования

Тестирование можно классифицировать по очень большому количеству признаков, и практически в каждой серьёзной книге о тестировании автор показывает свой (безусловно имеющий право на существование) взгляд на этот вопрос.

Вот самый простой, минимальный набор информации, необходимый начинающему тестировщику (рис.1.3).

Рис. 1.3 Упрощённая классификация тестирования

Методов тестирования на деле куда больше, а глубокое понимание каждого пункта в классификации требует определённого опыта. Для сравнения: переключить сайд.

Подробная классификация тестирования

Вот схема, на которой все способы классификации показаны одновременно. Многие авторы, создававшие подобные классификации, использовали интеллект-карты, однако такая техника не позволяет в полной мере отразить тот факт, что способы классификации пересекаются (т.е. некоторые виды тестирования можно отнести к разным способам классификации).

Зачем вообще нужна классификация тестирования? Она позволяет упорядочить знания и значительно ускоряет процессы планирования тестирования и разработки тест-кейсов, а также позволяет оптимизировать трудозатраты за счёт того, что тестировщику не приходится изобретать очередной велосипед. Теперь мы рассмотрим классификацию тестирования более подробно.

Классификация по запуску кода на исполнение

Далеко не всякое тестирование предполагает взаимодействие с работающим приложением. Потому в рамках данной классификации выделяют:

- **Статическое тестирование** (static testing) — тестирование без запуска кода на исполнение. В рамках этого подхода тестированию могут подвергаться:

o Документы (требования, тест-кейсы, описания архитектуры приложения, схемы баз данных и т.д.).

o Графические прототипы (например, эскизы пользовательского интерфейса).

o Код приложения (что часто выполняется самими программистами в рамках аудита кода (code review), являющегося специфической вариацией взаимного просмотра в применении к исходному коду). Код приложения также можно проверять с использованием техник тестирования на основе структур кода.

o Параметры (настройки) среды исполнения приложения.

o Подготовленные тестовые данные.

- **Динамическое тестирование** (dynamic testing) — тестирование с запуском кода на исполнение. Запускаться на исполнение может как код всего приложения целиком (системное тестирование), так и код нескольких взаимосвязанных частей (интеграционное тестирование), отдельных частей (модульное или компонентное тестирование) и даже отдельные участки кода. Основная идея этого вида тестирования состоит в том, что проверяется реальное поведение (части) приложения.

Классификация по доступу к коду и архитектуре приложения

- **Метод белого ящика** (white box testing113, open box testing, clear box testing, glass box testing) — у тестировщика есть доступ к внутренней структуре и коду приложения, а также есть достаточно знаний для понимания увиденного. Выделяют даже сопутствующую тестированию по методу белого ящика глобальную технику — тестирование на основе дизайна (design-based testing). Некоторые авторы склонны жёстко связывать этот метод со статическим тестированием, но ничто не мешает тестировщику запустить код на выполнение и при этом периодически обращаться к самому коду (а модульное тестирование и вовсе предполагает запуск кода на исполнение и при этом работу именно с кодом, а не с «приложением целиком»).

- **Метод чёрного ящика** (black box testing, closed box testing, specification-based testing) — у тестировщика либо нет доступа к внутренней структуре и коду приложения, либо недостаточно знаний для их понимания, либо он сознательно не обращается к ним в процессе тестирования. При этом абсолютное большинство видов тестирования работают по методу чёрного ящика, идею которого в альтернативном определении можно сформулировать так: тестировщик оказывает на приложение воздействия (и проверяет реакцию) тем же способом, каким при реальной эксплуатации приложения на

него воздействовали бы пользователи или другие приложения. В рамках тестирования по методу чёрного ящика основной информацией для создания тест-кейсов выступает документация (особенно — тебования) и общий здравый смысл (для случаев, когда поведение приложения в некоторой ситуации не регламентировано явно; иногда это называют «тестированием на основе не-явных требований», но канонического определения у этого подхода нет).

- **Метод серого ящика** (gray box testing) — комбинация методов белого ящика и чёрного ящика, состоящая в том, что к части кода и архитектуры у тестировщика доступ есть, а к части — нет. Его явное упоминание — крайне редкий случай: обычно говорят о методах белого или чёрного ящика в применении к тем или иным частям приложения, при этом понимая, что «приложение целиком» тестируется по методу серого ящика.

Метод черного ящика эффективен при тестировании совместимости (интеграционное тестирование), но с ним нельзя выявить ошибки в коде. С Белым ящиков все наоборот – он позволяет выявить ошибки на внутреннем устройстве программы, но мало эффективен при тестировании совместимости.

Классификация по степени автоматизации

- **Ручное тестирование** (manual testing) — тестирование, в котором тест-кейсы выполняются человеком вручную без использования средств автоматизации. Несмотря на то что это звучит очень просто, от тестировщика в те или иные моменты времени требуются такие качества, как терпеливость, наблюдательность, креативность, умение ставить нестандартные эксперименты, а также умение видеть и понимать, что происходит «внутри системы», т.е. как внешние воздействия на приложение трансформируются в его внутренние процессы.

- **Автоматизированное тестирование** (automated testing, test automation) — набор техник, подходов и инструментальных средств, позволяющий исключить человека из выполнения некоторых задач в процессе тестирования. Тест-кейсы частично или полностью выполняет специальное инструментальное средство, однако разработка тест-кейсов, подготовка данных, оценка результатов выполнения, написания отчётов об обнаруженных дефектах — всё это и многое другое по-прежнему делает человек.

У автоматизированного тестирования есть много как сильных, так и слабых сторон.

Если же выразить все преимущества и недостатки автоматизации тестирования одной фразой, то получается, что автоматизация позволяет ощутимо увеличить тестовое покрытие (test coverage), но при этом столь же ощутимо увеличивает риски.

Классификация по уровню детализации приложения (по уровню тестирования)

Модульное (компонентное) тестирование (unit testing, module testing, component testing) направлено на проверку отдельных небольших частей приложения, которые (как правило) можно исследовать изолированно от других подобных частей. При выполнении данного тестирования могут проверяться отдельные функции или методы классов, сами классы, взаимодействие классов, небольшие библиотеки, отдельные части приложения. Часто данный вид тестирования реализуется с использованием специальных технологий и инструментальных средств автоматизации тестирования, значительно упрощающих и ускоряющих разработку соответствующих тест-кейсов.

Из-за особенностей перевода на русский язык теряются нюансы степени детализации: «юнит-тестирование», как правило, направлено на тестирование атомарных участков кода, «модульное» — на тестирование классов и небольших библиотек, «компонентное» — на тестирование библиотек и структурных частей приложения. Но эта классификация не стандартизирована, и у различных авторов можно встретить совершенно разные взаимоисключающие трактовки.

Интеграционное тестирование (integration testing, component integration testing, pairwise integration testing, system integration testing, incremental testing, interface testing, thread testing) направлено на проверку взаимодействия между несколькими частями приложения (каждая из которых, в свою очередь, проверена отдельно на стадии модульного тестирования). К сожалению, даже если мы работаем с очень качественными отдельными компонентами, «на стыке» их взаимодействия часто возникают проблемы. Именно эти проблемы и выявляет интеграционное тестирование.

Системное тестирование (system testing) направлено на проверку всего приложения как единого целого, собранного из частей, проверенных на двух предыдущих стадиях. Здесь не только выявляются дефекты «на стыках» компонентов, но и появляется возможность полноценно взаимодействовать с приложением с точки зрения конечного пользователя, применяя множество других видов тестирования.

С классификацией по уровню детализации приложения связан интересный печальный факт: если предыдущая стадия обнаружила проблемы, то на следующей стадии эти проблемы точно нанесут удар по качеству; если же предыдущая стадия не обнаружила проблем, это ещё никоим образом не защищает нас от проблем на следующей стадии.

Классификация по (убыванию) степени важности тестируемых функций (по уровню функционального тестирования)

В некоторых источниках эту разновидность классификации также называют «по глубине тестирования».

Дымовое тестирование (smoke test, intake test, build verification test) направлено на проверку самой главной, самой важной, самой ключевой функциональности, неработоспособность которой делает бессмысленной саму идею использования приложения (или иного объекта, подвергаемого дымовому тестированию).

Дымовое тестирование проводится после выхода нового билда, чтобы определить общий уровень качества приложения и принять решение о целесообразности или нецелесообразности выполнения тестирования критического пути и расширенного тестирования. Поскольку тест-кейсов на уровне дымового тестирования относительно немного, а сами они достаточно просты, но при этом очень часто повторяются, они являются хорошими кандидатами на автоматизацию. В связи с высокой важностью тест-кейсов на данном уровне пороговое значение метрики их прохождения часто выставляется равным 100 % или близким к 100 %.

Тестирование критического пути (critical path test) направлено на исследование функциональности, используемой типичными пользователями в типичной повседневной деятельности.

Идея: существует большинство пользователей, которые чаще всего используют некое подмножество функций приложения. Именно эти функции и нужно проверить, как только мы убедились, что приложение «в принципе работает» (т.е. дымовой тест прошёл успешно). Если по каким-то причинам приложение не выполняет эти функции или выполняет их некорректно, очень многие пользователи не смогут достичь множества своих целей. Пороговое значение метрики успешного прохождения «теста критического пути» уже немного ниже, чем в дымовом тестировании, но всё равно достаточно высоко (как правило, порядка 70–80–90 % — в зависимости от сути проекта).

Расширенное тестирование (extended test) направлено на исследование всей заявленной в требованиях функциональности — даже той, которая низко проранжирована по степени важности. При этом здесь также учитывается, какая функциональность является более важной, а какая — менее важной. Но при наличии достаточного количества времени и иных ресурсов тест- кейсы этого уровня могут затронуть даже самые низкоприоритетные требования.

Ещё одним направлением исследования в рамках данного тестирования являются нетипичные, маловероятные, экзотические случаи и сценарии использования функций и свойств приложения, затронутых на предыдущих уровнях. Пороговое значение метрики успешного прохождения расширенного тестирования существенно ниже, чем в тестировании критического пути (иногда можно увидеть даже значения в диапазоне 30–50 %, т.к. подавляющее большинство найденных здесь дефектов не представляет угрозы для успешного использования приложения большинством пользователей).

Классификация тестирования по (убыванию) степени важности тестируемых функций (по уровню функционального тестирования)

Классификация по принципам работы с приложением

- **Позитивное тестирование** (positive testing) направлено на исследование приложения в ситуации, когда все действия выполняются строго по инструкции без каких бы то ни было ошибок, отклонений, ввода неверных данных и т.д. Если позитивные тест-кейсы завершаются ошибками, это тревожный признак — приложение работает неверно даже в идеальных условиях (и можно предположить, что в неидеальных условиях оно работает ещё хуже). Для ускорения тестирования несколько позитивных тест-кейсов

можно объединять (например, перед отправкой заполнить все поля формы верными значениями) — иногда это может усложнить диагностику ошибки, но существенная экономия времени компенсирует этот риск.

- **Негативное тестирование** (negative testing, invalid testing) — направлено на исследование работы приложения в ситуациях, когда с ним выполняются (некорректные) операции и/или используются данные, потенциально приводящие к ошибкам (классика жанра — деление на ноль). Поскольку в реальной жизни таких ситуаций значительно больше (пользователи допускают ошибки, злоумышленники осознанно «ломают» приложение, в среде работы приложения возникают проблемы и т.д.), негативных тест-кейсов оказывается значительно больше, чем позитивных (иногда — в разы или даже на порядки). В отличие от позитивных негативные тест-кейсы не стоит объединять, т.к. подобное решение может привести к неверной трактовке поведения приложения и пропуску (необнаружению) дефектов.

Позитивное тестирование первично, т.к. пользователей, который хотят нормально работать с приложением больше чем тех, кто хочет его поломать.

Классификация на основе выбора входных данных

- **Классы эквивалентности** (Equivalence partitioning) - это множество входящих данных, на которые система отреагирует одинаковым способом. Исходные данные необходимо разбить на конечное число классов эквивалентности. В одном классе эквивалентности содержатся такие тесты, что если один тест из класса эквивалентности обнаруживает некоторую ошибку, то и любой другой тест из этого класса эквивалентности должен обнаруживать эту же ошибку. Каждый тест должен включать, по возможности, максимальное количество классов эквивалентности, чтобы минимизировать общее число тестов.

- **Граничные условия** (Boundary value analysis) - это ситуации, возникающие на высших и нижних границах входных классов эквивалентности.

Существует несколько правил:

1. Построить тесты с неправильными входными данными для ситуации незначительного выхода за границы области значений. Если входные значения должны быть в интервале [-1.0 .. +1.0], проверяем −1.0, 1.0, −1.000001, 1.000001.

2. Обязательно писать тесты для минимальной и максимальной границы диапазона.

3. Если вход и выход программы представляет упорядоченное множество, сосредоточить внимание на первом и последнем элементах списка.

Анализ граничных значений, если он применён правильно, позволяет обнаружить большое число ошибок. Однако определение этих границ для каждой задачи может являться отдельной трудной задачей. Также этот метод не проверяет комбинации входных значений.

- **Метод пар** (Pairwise testing) – техника формирования наборов тестовых данных. 97 % ошибок возникают при взаимодействии двух пар параметров. Благодаря этому методу мы узнаем, как одни параметры взаимодействуют с другими. Этот метод тесно связан с комбинаторикой. Суть в том, что у нас получается много сочетаний, поэтому лишние попросту убираем. Однако минус в том, что могут возникнуть дополнительные условия, тогда задача усложнится.

Классификация по привлечению конечных пользователей

Все три перечисленных ниже вида тестирования относятся к операционному тестированию.

 Альфа-тестирование (alpha testing) выполняется внутри организации-разработчика с возможным частичным привлечением конечных пользователей. Может являться формой внутреннего приёмочного тестирования. В некоторых источниках отмечается, что это тестирование должно проводиться без привлечения команды разработчиков, но другие источники не выдвигают такого требования. Суть этого вида вкратце: продукт уже можно периодически показывать внешним пользователям, но он ещё достаточно «сырой», потому основное тестирование выполняется организацией-разработчиком.

 Бета-тестирование (beta testing) выполняется вне организации-разработчика с активным привлечением конечных пользователей/заказчиков. Может являться формой внешнего приёмочного тестирования. Суть этого вида вкратце: продукт уже можно открыто показывать внешним пользователям, он уже достаточно стабилен, но проблемы всё ещё могут быть, и для их выявления нужна обратная связь от реальных пользователей.

 Гамма-тестирование (gamma testing) — финальная стадия тестирования перед выпуском продукта, направленная на исправление незначительных дефектов, обнаруженных в бета-тестировании. Как правило, также выполняется с максимальным привлечением конечных пользователей/заказчиков. Может являться формой внешнего

приёмочного тестирования. Суть этого вида вкратце: продукт уже почти готов, и сейчас обратная связь от реальных пользователей используется для устранения последних недоработок.

Классификация по степени формализации

- **Тестирование на основе тест-кейсов** (scripted testing, test case based test-ing) — формализованный подход, в котором тестирование производится на основе заранее подготовленных тест-кейсов, наборов тест-кейсов и иной документации. Это самый распространённый способ тестирования, который также позволяет достичь максимальной полноты исследования приложения за счёт строгой систематизации процесса, удобства применения метрик и широкого набора выработанных за десятилетия и проверенных на практике рекомендаций.
- **Исследовательское тестирование** (exploratory testing) — частично формализованный подход, в рамках которого тестировщик выполняет работу с приложением по выбранному сценарию, который, в свою очередь, дорабатывается в процессе выполнения с целью более полного исследования приложения. Ключевым фактором успеха при выполнении исследовательского тестирования является именно работа по сценарию, а не выполнение разрозненных бездумных операций. Существует даже специальный сценарный подход, называемый сессионным тестированием (session-based test-ing). В качестве альтернативы сценариям при выборе действий с приложением иногда могут использоваться чек-листы, и тогда этот вид тестирования называют тестированием на основе чек-листов (checklist-based testing).
- **Свободное (интуитивное) тестирование** (ad hoc testing) — полностью неформализованный подход, в котором не предполагается использования ни тест-кейсов, ни чек-листов, ни сценариев — тестировщик полностью опирается на свой профессионализм и интуицию (experience-based testing) для спонтанного выполнения с приложением действий, которые, как он считает, могут обнаружить ошибку. Этот вид тестирования используется редко и исключительно как дополнение к полностью или частично формализованному тестированию в случаях, когда для исследования некоторого аспекта поведения приложения нет тест-кейсов.

Классификация по целям и задачам

- **Позитивное тестирование** (рассмотрено ранее).
- **Негативное тестирование** (рассмотрено ранее).

• **Функциональное тестирование** (functional testing) — вид тестирования, направленный на проверку корректности работы функциональности приложения (корректность реализации функциональных требований). Часто функциональное тестирование ассоциируют с тестированием по методу чёрного ящика, однако и по методу белого ящика вполне можно проверять корректность реализации функциональности.

Часто возникает вопрос, в чём разница между функциональным тестированием (functional testing) и тестированием функциональности (functionality testing).

Если вкратце, то:

➢ функциональное тестирование (как антоним нефункционального) направлено на проверку того, какие функции приложения реализованы, и что они работают верным образом;

➢ тестирование функциональности направлено на те же задачи, но акцент смещён в сторону исследования приложения в реальной рабочей среде, после локализации и в тому подобных ситуациях.

•**Нефункциональное тестирование** (non-functional testing) — вид тестирования, направленный на проверку нефункциональных особенностей приложения (корректность реализации нефункциональных требований{), таких как удобство использования, совместимость, производительность, безопасность и т.д.

•**Инсталляционное тестирование** (installation testing, installability testing) — тестирование, направленное на выявление дефектов, влияющих на протекание стадии инсталляции (установки) приложения. В общем случае такое тестирование проверяет множество сценариев и аспектов работы инсталлятора.

•**Регрессионное тестирование** (regression testing) — тестирование, направленное на проверку того факта, что в ранее работоспособной функциональности не появились ошибки, вызванные изменениями в приложении или среде его функционирования. «Фундаментальная проблема при сопровождении программ состоит в том, что исправление одной ошибки с большой вероятностью (20–50 %) влечёт появление новой». Потому регрессионное тестирование является неотъемлемым инструментом обеспечения качества и активно используется практически в любом проекте.

•**Повторное тестирование** (re-testing, confirmation testing) — выполнение тест-кейсов, которые ранее обнаружили дефекты, с целью подтверждения устранения дефектов. Фактически этот вид тестирования сводится к действиям на финальной стадии

жизненного цикла отчёта о дефекте, направленным на то, чтобы перевести дефект в состояние «проверен» и «закрыт».

- **Приёмочное тестирование** (acceptance testing) — формализованное тестирование, направленное на проверку приложения с точки зрения конечного пользователя/заказчика и вынесения решения о том, принимает ли заказчик работу у исполнителя (проектной команды).

- **Операционное тестирование** (operational testing) — тестирование, проводимое в реальной или приближенной к реальной операционной среде, включающей операционную систему, системы управления базами данных, серверы приложений, веб-серверы, аппаратное обеспечение и т.д.

- **Тестирование удобства использования** (usability testing) — тестирование, направленное на исследование того, насколько конечному пользователю понятно, как работать с продуктом, а также на то, насколько ему нравится использовать продукт. И это не оговорка — очень часто успех продукта зависит именно от эмоций, которые он вызывает у пользователей. Для эффективного проведения этого вида тестирования требуется реализовать достаточно серьёзные исследования с привлечением конечных пользователей.

- **Тестирование доступности** (accessibility testing) — тестирование, направленное на исследование пригодности продукта к использованию людьми с ограниченными возможностями (слабым зрением и т.д.)

В слайде и т.д.

- **Тестирование интерфейса** (interface testing) — тестирование, направленное на проверку интерфейсов приложения или его компонентов. По определению ISTQB-глоссария этот вид тестирования относится к интеграционному тестированию, и это вполне справедливо для таких его вариаций как тестирование интерфейса прикладного программирования (API testing) и интерфейса командной строки (CLI testing), хотя последнее может выступать и как разновидность тестирования пользовательского интерфейса, если че-рез командную строку с приложением взаимодействует пользователь, а не другое приложение. Однако многие источники предлагают включить в состав тестирования интерфейса и тестирование непосредственно интерфейса пользователя (GUI testing).

•**Тестирование безопасности** (security testing) — тестирование, направленное на проверку способности приложения противостоять злонамеренным попыткам получения доступа к данным или функциям, права на доступ к которым у злоумышленника нет.

•**Сравнительное тестирование** (comparison testing) — тестирование, направленное на сравнительный анализ преимуществ и недостатков разрабатываемого продукта по отношению к его основным конкурентам.

•**Демонстрационное тестирование** (qualification testing) — формальный процесс демонстрации заказчику продукта с целью подтверждения, что продукт соответствует всем заявленным требованиям. В отличие от приёмочного тестирования этот процесс более строгий и всеобъемлющий, но может проводиться и на промежуточных стадиях разработки продукта.

•**Избыточное тестирование** (exhaustive testing) — тестирование приложения со всеми возможными комбинациями всех возможных входных данных во всех возможных условиях выполнения. Для сколь бы то ни было сложной системы нереализуемо, но может применяться для проверки отдельных крайне простых компонентов.

•**Тестирование надёжности** (reliability testing199) — тестирование способности приложения выполнять свои функции в заданных условиях на протяжении заданного времени или заданного количества операций.

Как легко можно понять, тестировщику приходится работать с огромным количеством информации, выбирать из множества вариантов решения задач и изобретать новые. В процессе этой деятельности объективно невозможно удержать в голове все мысли, а потому продумывание и разработку тест-кейсов рекомендуется выполнять с использованием так называемых «чек-листов».

Чек-лист (checklist) — набор идей [тест-кейсов]. Последнее слово не зря взято в скобки, т.к. в общем случае чек-лист — это просто набор идей: идей по тестированию, идей по разработке, идей по планированию и управлению — **любых** идей.

Чек-лист чаще всего представляет собой обычный и привычный нам список.

Важно понять, что нет и не может быть никаких запретов и ограничений при разработке чек-листов — главное, чтобы они помогали в работе. Иногда чек-листы могут даже выражаться графически хотя традиционно их составляют в виде многоуровневых списков.

Поскольку в разных проектах встречаются однотипные задачи, хорошо продуманные и аккуратно оформленные чек-листы могут использоваться повторно, чем достигается экономия сил и времени.

Свойства:

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. человеку всё равно удобнее воспринимать информацию в виде неких небольших групп идей, переход между которыми является понятным и очевидным

Полнота и неизбыточность.

Чек-лист должен представлять собой аккуратную «сухую выжимку» идей, в которых нет дублирования (часто появляется из-за разных формулировок одной и той же идеи), и в то же время ничто важное не упущено.

Тест-кейсы

Терминология и общие сведения

Для начала определимся с терминологией.

Во главе всего лежит термин «тест». Официальное определение звучит так.

Тест (test) — набор из одного или нескольких тест-кейсов.

Поскольку среди всех прочих терминов этот легче и быстрее всего произносить, в зависимости от контекста под ним могут понимать и отдельный пункт чек-листа, и отдельный шаг в тест-кейсе, и сам тест-кейс, и набор тест-кейсов. Продолжать можно долго. Главное здесь одно: если вы слышите или видите слово «тест», воспринимайте его в контексте.

Теперь рассмотрим самый главный для нас термин — «тест-кейс».

Тест-кейс (test case) — набор входных данных, условий выполнения и ожидаемых результатов, разработанный с целью проверки того или иного свойства или поведения программного средства.

Под тест-кейсом также может пониматься соответствующий документ, представляющий формальную запись тест-кейса.

Если у тест-кейса не указаны входные данные, условия выполнения и ожидаемые результаты, и/или не ясна цель тест-кейса — это плохой тест-кейс (иногда он не имеет смысла, иногда его и вовсе невозможно выполнить).

Цель написания тест-кейсов

Тестирование можно проводить и без тест-кейсов. Наличие же тест-кейсов позволяет:

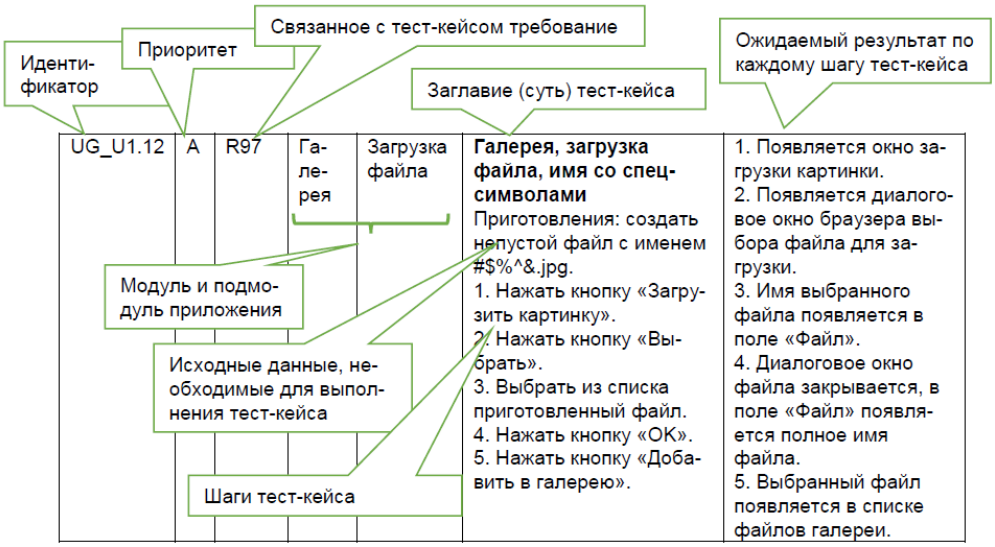
- ✓ Структурировать и систематизировать подход к тестированию (без чего крупный проект почти гарантированно обречён на провал).
- ✓ Вычислять метрики тестового покрытия и принимать меры по его увеличению (тест-кейсы здесь являются главным источником информации, без которого существование подобных метрик теряет смысл).
- ✓ Уточнить взаимопонимание между заказчиком, разработчиками и тестировщиками (тест-кейсы зачастую намного более наглядно показывают поведение приложения, чем это отражено в требованиях).
- ✓ Проводить регрессионное тестирование и повторное тестирование (которые без тест-кейсов было бы вообще невозможно выполнить).
- ✓ Повышать качество требований.
- ✓ Быстро вводить в курс дела нового сотрудника, недавно подключившегося к проекту.

Атрибуты (поля) тест-кейса

Как уже было сказано выше, термин «тест-кейс» может относиться к формальной записи тест-кейса в виде технического документа. Эта запись имеет общепринятую структуру, компоненты которой называются атрибутами тест-кейса.

В зависимости от инструмента управления тест-кейсам внешний вид их записи может немного отличаться, могут быть добавлены или убраны отдельные поля, но концепция остаётся неизменной.

Общий вид всей структуры тест-кейса представлен на рисунке



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.

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.
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 — потому, что один тест-кейс может затрагивать несколько требований). Наличие этого поля улучшает такое свойство тест-кейса, как прослеживаемость.
// (если трудно определится, его можно оставить пустым.)

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). И вот уже для таких небольших частей приложения придумать чек-листы и создать хорошие тест-кейсы становится намного проще.

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.

Заглавие тест-кейса (title) призвано упростить быстрое понимание основной идеи тест-кейса без обращения к его остальным атрибутам. Именно это поле является наиболее информативным при просмотре списка тест-кейсов.

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,

Шаги тест-кейса (steps) описывают последовательность действий, которые необходимо реализовать в процессе выполнения тест-кейса.

начинайте с понятного и очевидного места, не пишите лишних начальных шагов (запуск приложения, включить комп)

если вы пишете на русском языке, используйте безличную форму (например, «открыть», «ввести», «добавить» вместо «откройте», «введите», «добавьте»);

Ожидаемые результаты (expected results) по каждому шагу тест-кейса опи-сывают реакцию приложения на действия, описанные в поле «шаги тест-кейса». Номер шага соответствует номеру результата.

Инструментальные средства управления тестированием

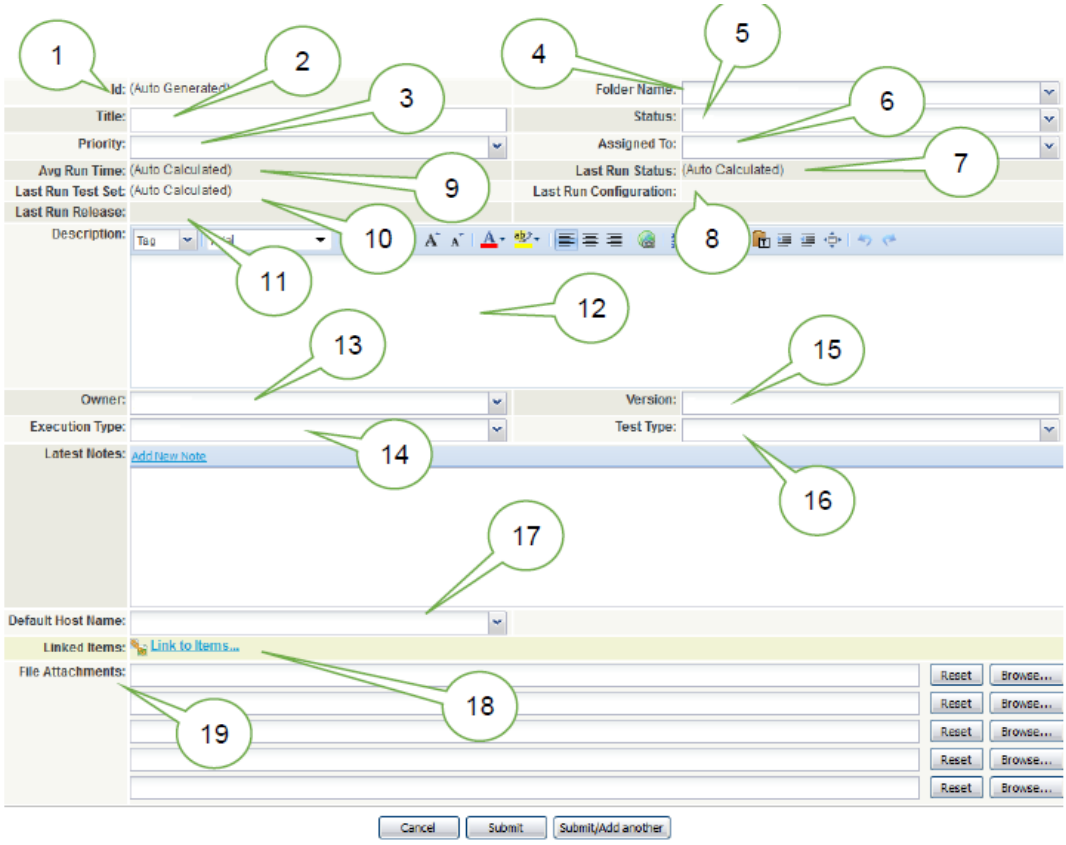
Инструментальных средств управления тестированием (test management tool) очень много, к тому же многие компании разрабатывают свои внутренние средства решения этой задачи.

Не имеет смысла заучивать, как работать с тест-кейсами в том или ином инструментальном средстве — принцип везде един, и соответствующие навыки

нарабатываются буквально за пару дней. Что на текущий момент важно понимать, так это общий набор функций, реализуемых такими инструментальными средствами.

Хорошее инструментальное средство управления тестированием берёт на себя все рутинные технические операции, которые объективно необходимо выполнять в процессе реализации жизненного цикла тестирования.

QAComplete



1. Id (идентификатор), как видно из соответствующей надписи, автогенерируемый.

2. Title (заглавие), как и в большинстве систем, является обязательным для заполнения.

3. Priority (приоритет) по умолчанию предлагает значения high (высокий), medium (средний), low (низкий).

4. Folder name (расположение) является аналогом полей «Модуль» и «Подмодуль» и позволяет выбрать из выпадающего древовидного списка соответствующее значение, описывающее, к чему относится тест-кейс.

5. Status (статус) показывает текущее состояние тест-кейса: new (новый), approved (утверждён), awaiting approval (на рассмотрении), in design (в разработке), outdated (устарел), rejected (отклонён).

6. Assigned to (исполнитель) указывает, кто в данный момент является «основной рабочей силой» по данному тест-кейсу (или кто должен принять решение о, например, утверждении тест-кейса).

7. Last Run Status (результат последнего запуска) показывает, прошёл ли тест успешно (passed) или завершился неудачей (failed).

8. Last Run Configuration (конфигурация, использованная для последнего запуска) показывает, на какой аппаратно-программной платформе тест-кейс выполнялся в последний раз.

9. Avg Run Time (среднее время выполнения) содержит вычисленное автоматически среднее время, необходимое на выполнение тест-кейса.

10. Last Run Test Set (в последний раз выполнялся в наборе) содержит информацию о наборе тест-кейсов, в рамках которого тест-кейс выполнялся последний раз.

11. Last Run Release (последний раз выполнялся на выпуске) содержит информацию о выпуске (билде) программного средства, на котором тест-кейс выполнялся последний раз.

12. Description (описание) позволяет добавить любую полезную информацию о тест-кейсе (включая особенности выполнения, приготовления и т.д.).

13. Owner (владелец) указывает на владельца тест-кейса (как правило — его автора).

14. Execution Type (способ выполнения) по умолчанию предлагает только значение manual (ручное выполнение), но при соответствующих настройках и интеграции с другими продуктами список можно расширить (как минимум добавить automated (автоматизированное выполнение)).

15. Version (версия) содержит номер текущей версии тест-кейса (фактически — счётчик того, сколько раз тест-кейс редактировали). Вся история изменений сохраняется, предоставляя возможность вернуться к любой из предыдущих версий.

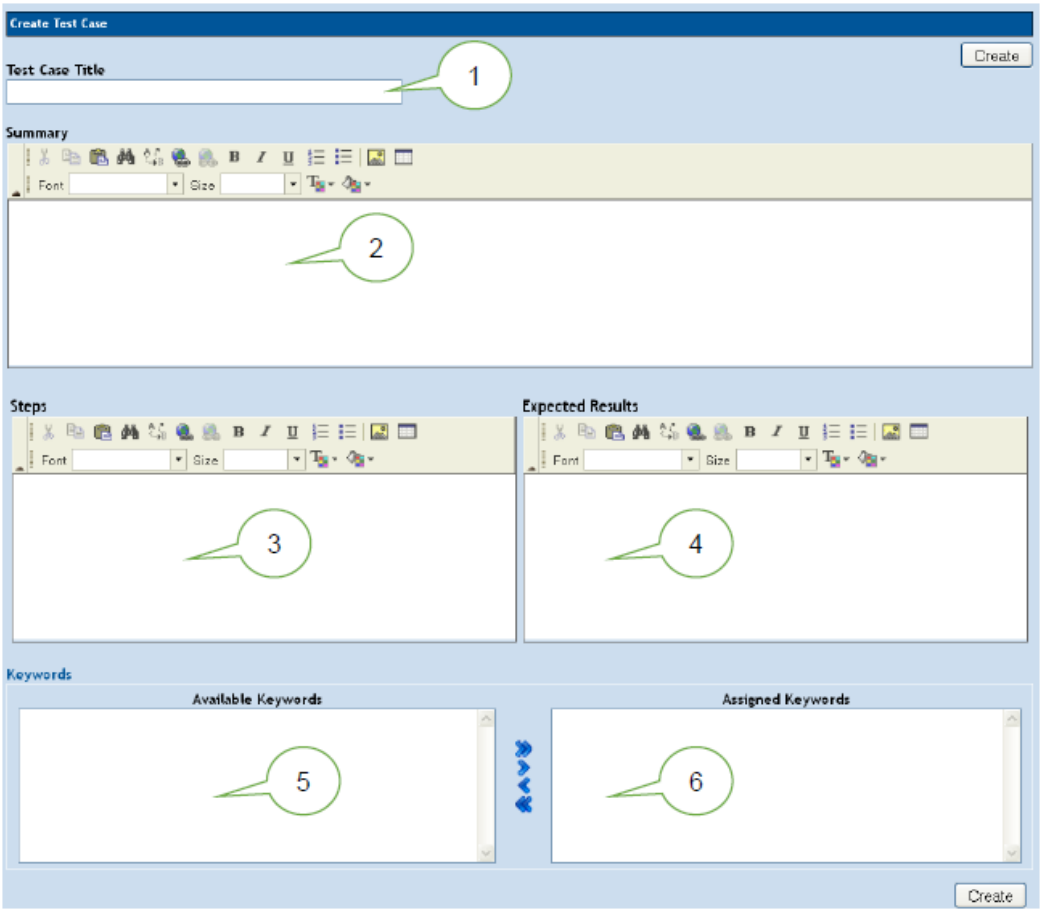
16. Test Type (вид теста) по умолчанию предлагает такие варианты, как negative (негативный), positive (позитивный), regression (регрессионный), smoke-test (дымный).

17. Default host name (имя хоста по умолчанию) в основном используется в автоматизированных тест-кейсах и предлагает выбрать из списка имя зарегистрированного компьютера, на котором установлен специальный клиент.

18. Linked Items (связанные объекты) представляют собой ссылки на требования, отчёты о дефектах и т.д.

19. File Attachments (вложения) могут содержать тестовые данные, поясняющие изображения, видеоролики и т.д.

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.

Как видите, инструментальные средства управления тест-кейсами могут быть и достаточно минималистичными.

Ошибки, дефекты, сбои, отказы и т.д.

Упрощённый взгляд на понятие дефекта

начнём с очень простого: дефектом упрощённо можно считать любое расхождение ожидаемого (свойства, результата, поведения и т.д., которое мы ожидали увидеть) и фактического (свойства, результата, поведения и т.д., которое мы на самом деле увидели).

В силлабусе ISTQB написано, что человек совершает ошибки, которые приводят к возникновению дефектов в коде, которые, в свою очередь, приводят к сбоям и отказам приложения (однако сбои и отказы могут возникать и из-за внешних условий, таких как электромагнитное воздействие на оборудование и т.д.)

При обнаружении дефекта создаётся отчёт о дефекте.

Ошибка (error303, mistake) — действие человека, приводящее к некорректным результатам.

Дефект (defect304, bug, problem, fault) — недостаток в компоненте или системе, способный привести к ситуации сбоя или отказа.

Дефект — отклонение (deviation308) фактического результата (actual re-sult309) от ожиданий наблюдателя (expected result310), сформированных на основе требований, спецификаций, иной документации или опыта и здра-вого смысла.

Сбой (interruption305) или **отказ** (failure306) — отклонение поведения си- стемы от ожидаемого.

Ожидаемый результат — поведение системы, описанное в требованиях.

Фактический результат — поведение системы, наблюдаемое в процессе тестирования.

Отчёт о дефекте и его жизненный цикл

Как было сказано ранее, при обнаружении дефекта тестировщик создаёт отчёт о дефекте.

Отчёт о дефекте (defect report) — документ, описывающий и приоритизирующий обнаруженный дефект, а также содействующий его устранению.

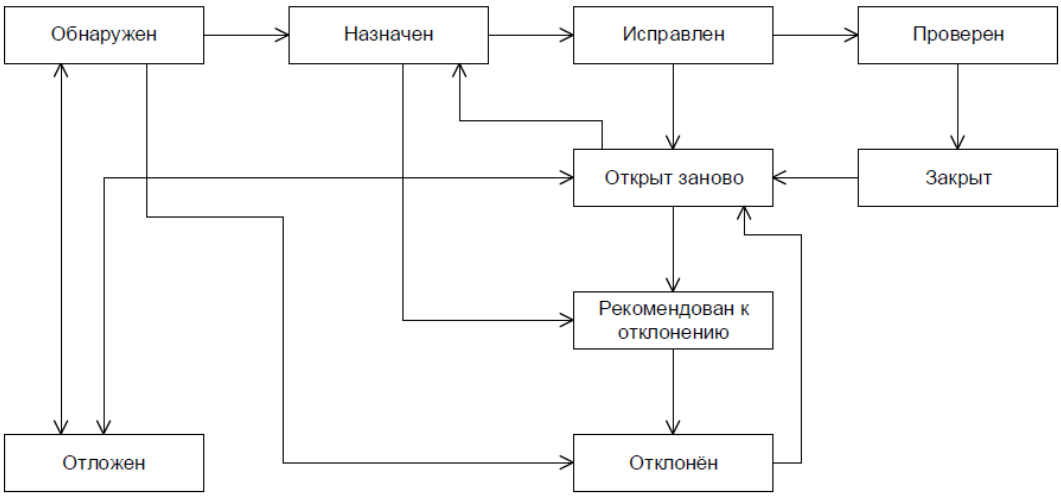
Как следует из самого определения, отчёт о дефекте пишется со следующими основными целями:

- предоставить информацию о проблеме — уведомить проектную команду и иных заинтересованных лиц о наличии проблемы, описать суть проблемы;

- 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.

На последней цели следует остановиться подробнее. Есть мнение, что «хорошо написанный отчёт о дефекте — половина решения проблемы для программиста». И действительно, от полноты, корректности, аккуратности, подробности и логичности отчёта о дефекте зависит очень многое — одна и та же проблема может быть описана так, что программисту останется буквально исправить пару строк кода, а может быть описана и так, что сам автор отчёта на следующий день не сможет понять, что же он имел в виду.

Отчёт о дефекте (и сам дефект вместе с ним) проходит определённые стадии жизненного цикла, которые схематично можно показать так

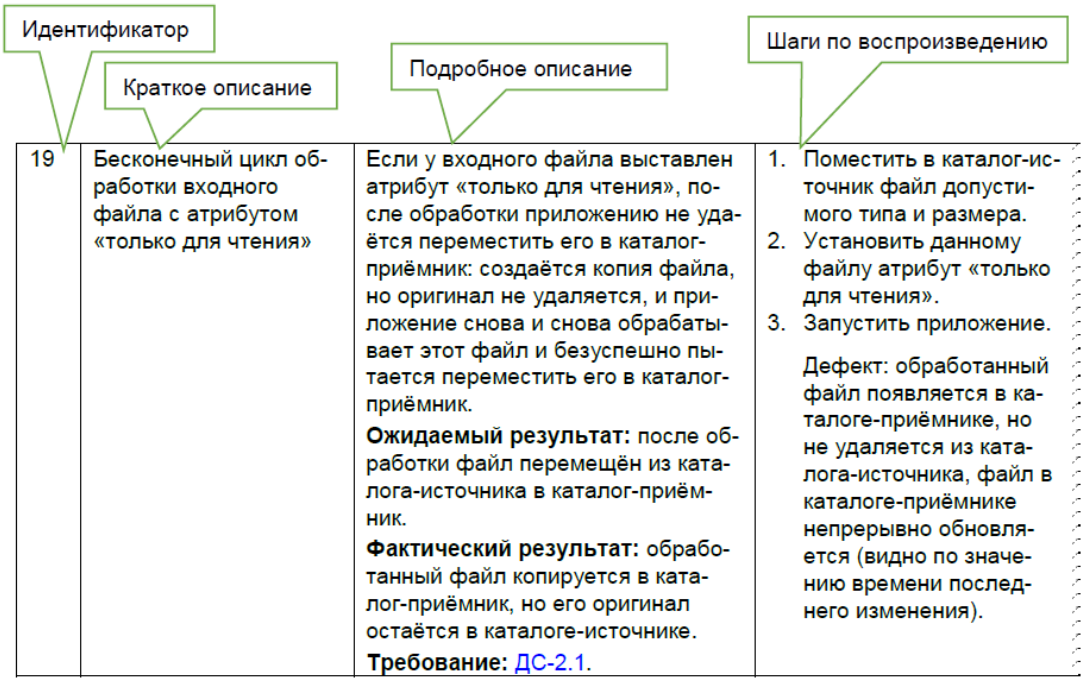


- Обнаружен (submitted) — начальное состояние отчёта (иногда называется «Новый» (new)), в котором он находится сразу после создания. Некоторые средства также позволяют сначала создавать черновик (draft) и лишь потом публиковать отчёт.
- Назначен (assigned) — в это состояние отчёт переходит с момента, когда кто-то из проектной команды назначается ответственным за исправление дефекта. Назначение ответственного производится или решением лидера команды разработки, или коллегиально, или по добровольному принципу, или 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, удостоверившийся, что дефект на самом деле был устранён. Как правило, такую проверку выполняет тестировщик, изначально написавший отчёт о дефекте.
- Закрыт (closed) — состояние отчёта, означающее, что по данному дефекту не планируется никаких дальнейших действий.
- Открыт заново (reopened) — в это состояние (как правило, из состояния «Исправлен») отчёт переводит тестировщик, удостоверившийся, что дефект по-прежнему воспроизводится на билде, в котором он уже должен быть исправ-лен.
- Рекомендован к отклонению (to be declined) — в это состояние отчёт о дефекте может быть переведён из множества других состояний с целью вынести на рассмотрение вопрос об отклонении отчёта по той или иной причине. Если рекомендация является обоснованной, отчёт переводится в состояние «Отклонён»
- Отклонён (declined) — в это состояние отчёт переводится в случаях, подробно описанных в пункте «Закрыт», если средство управления отчётами о дефектах предполагает использование этого состояния вместо состояния «Закрыт» 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 (например, выйдет новая версия библиотеки, вернётся из отпуска специалист по некоей технологии, изменятся требования заказчика и т.д.)

Атрибуты (поля) 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.

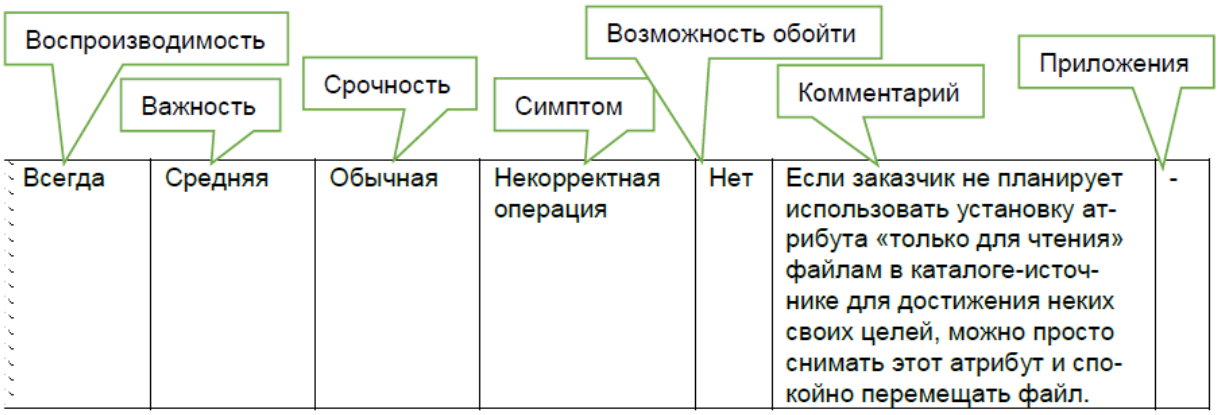


Идентификатор (identifier) представляет собой уникальное значение, позволяющее однозначно отличить один отчёт о дефекте от другого и используемое во всевозможных ссылках. В общем случае идентификатор отчёта о дефекте может представлять собой просто уникальный номер, но (если позволяет инструментальное 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?».

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).

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)

Важность (severity) показывает степень ущерба, который наносится проекту существованием дефекта. (Важность может быть Критической, высокой, средней, низкой).

Срочность (priority) показывает, как быстро дефект должен быть устранён. (Наивысшая, Высокая, Обычная, Никая)

Симптом (symptom) — позволяет классифицировать дефекты по их типичному проявлению. Не существует никакого общепринятого списка симптомов

Возможность обойти (workaround) — показывает, существует ли альтернативная последовательность действий, выполнение которой позволило бы пользователю достичь поставленной цели. В некоторых инструментальных средствах управления отчётами о дефектах это поле может просто принимать значения «Да» и «Нет», в некоторых при выборе «Да» появляется возможность описать обходной путь. Традиционно считается, что дефектам без возможности обхода стоит повысить срочность исправления.

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 прикреплённых к отчёту о дефекте приложений (копий экрана, вызывающих сбой файлов и т.д.)

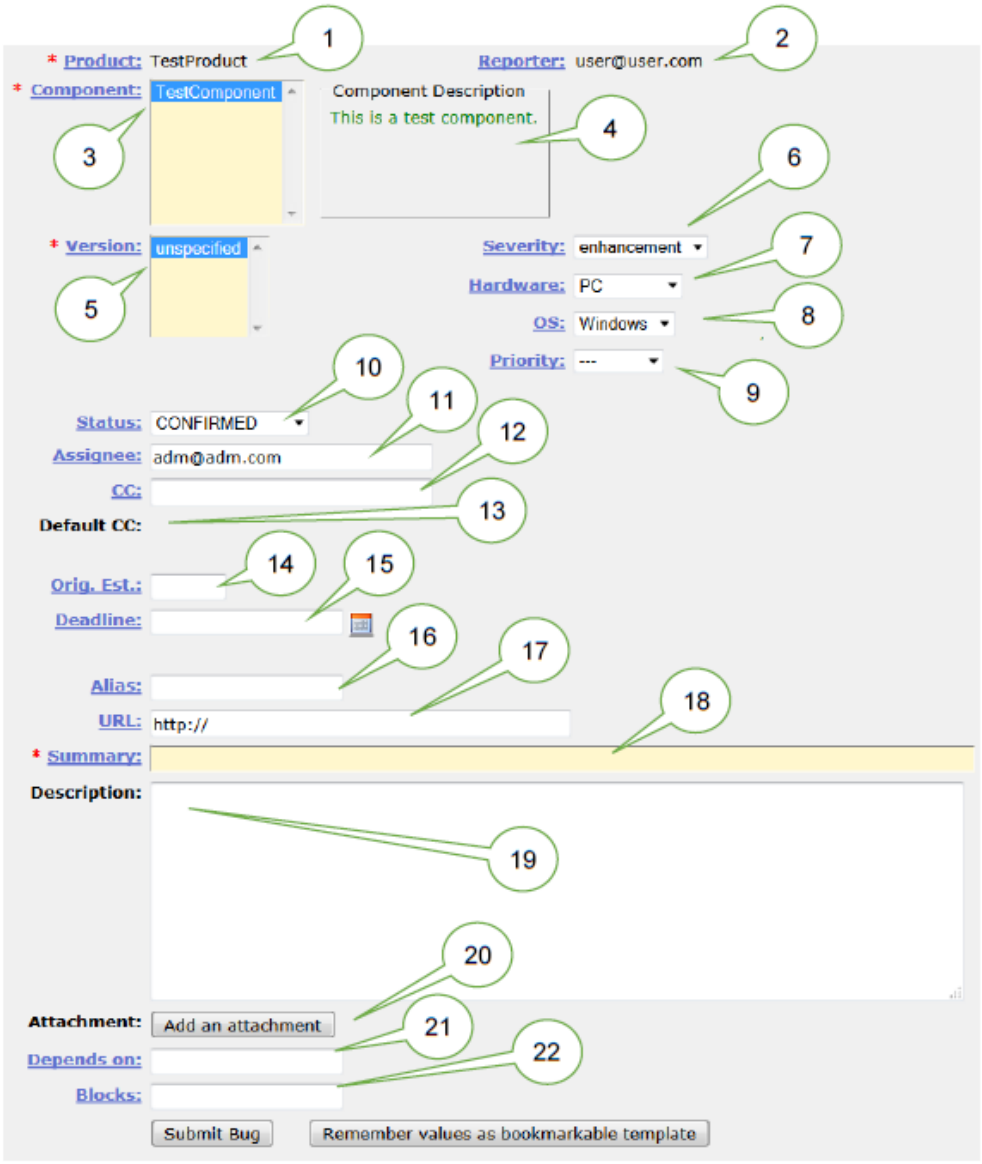
Инструментальные средства управления отчётами о дефектах

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. Как и в случае с инструментальными средствами управления тестированием, здесь не имеет смысла заучивать, как работать с отчётами о дефектах в том или ином средстве. Мы лишь рассмотрим общий набор функций, как правило, реализуемых такими средствами. Эти функции включают:

- ✓ 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.

Bugzilla318



1. Product (продукт) позволяет указать, к какому продукту или проекту относится дефект.

2. Reporter (автор отчёта) содержит e-mail автора описания дефекта.

3. Component (компонент) содержит указание компонента приложения, к которому относится описываемый дефект.

4. Component description (описание компонента) содержит описание компонента приложения, к которому относится описываемый дефект. Эта информация загружается автоматически при выборе компонента.

5. Version (версия) содержит указание версии продукта, в которой был обнаружен дефект.

6. Severity (важность) содержит указание важности дефекта.

7. Hardware (аппаратное обеспечение) позволяет выбрать профиль аппаратного окружения, в котором проявляется дефект.

8. OS (операционная система) позволяет указать операционную систему, под которой проявляется дефект.

9. Priority (срочность) позволяет указать срочность исправления дефекта.

10. Status (статус) позволяет установить статус отчёта о дефекте.

В официальной документации рекомендуется сразу же после установки Bugzilla сконфигурировать набор статусов и правила жизненного цикла отчёта о дефектах в соответствии с принятыми в какой-либо компании правилами.

11. Assignee (ответственный) указывает e-mail участника проектной команды, ответственного за изучение и исправление дефекта.

12. CC (уведомлять) содержит список e-mail адресов участников проектной команды, которые будут получать уведомления о происходящем с данным дефектом.

13. Default CC (уведомлять по умолчанию) содержит e-mail адрес(а) участников проектной команды, которые по умолчанию будут получать уведомления о происходящем с любыми дефектами (чаще всего здесь указываются e-mail адреса рассылок).

14. Original estimation (начальная оценка) позволяет указать начальную оценку того, сколько времени займёт устранение дефекта.

15. Deadline (крайний срок) позволяет указать дату, к которой дефект обязательно нужно исправить.

16. Alias (псевдоним) позволяет указать короткое запоминающееся название дефекта (возможно, в виде некоей аббревиатуры) для удобства упоминания дефекта в разнообразных документах.

17. URL (URL) позволяет указать URL, по которому проявляется дефект (особенно актуально для веб-приложений).

18. Summary (краткое описание) позволяет указать краткое описание дефекта.

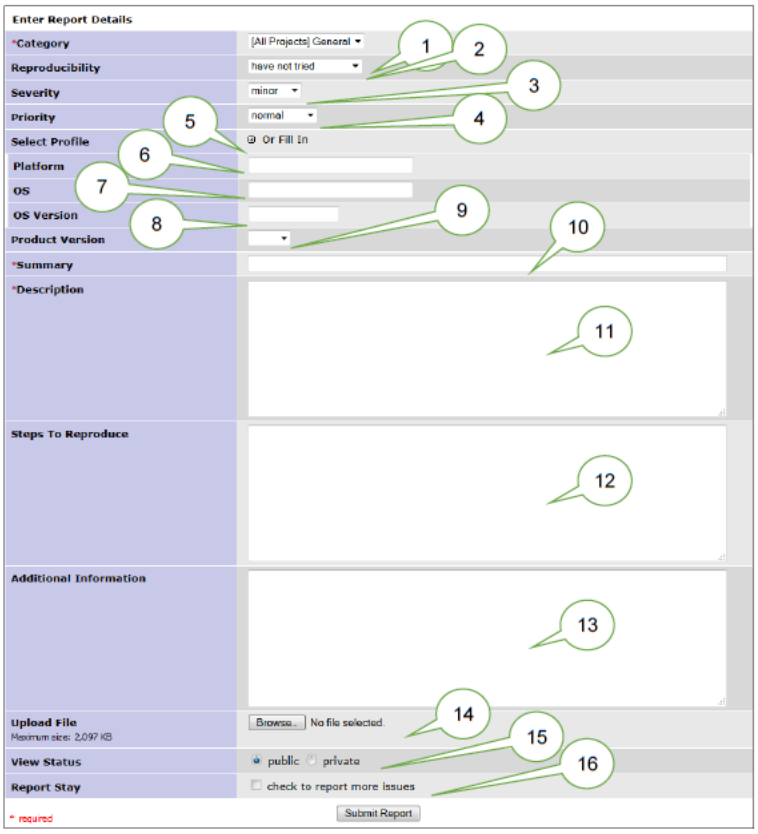
19. Description (подробное описание) позволяет указать подробное описание дефекта.

20. Attachment (вложение) позволяет добавить к отчёту о дефекте вложения в виде прикреплённых файлов.

21. Depends on (зависит от) позволяет указать перечень дефектов, которые должны быть устранены до начала работы с данным дефектом.

22. Blocks (блокирует) позволяет указать перечень дефектов, к работе с которыми можно будет приступить только после устранения данного дефекта.

Создание отчёта о дефекте в Mantis



1. Category (категория) содержит указание проекта или компонента приложения, к которому относится описываемый дефект.

2. Reproducibility (воспроизводимость) дефекта. Тут Mantis предлагает нетипично большое количество вариантов

3. Severity (важность) содержит указание важности дефекта.

4. Priority (срочность) позволяет указать срочность исправления дефекта. По умолчанию Mantis предлагает следующие варианты:

5. Select profile (выбрать профиль) позволяет выбрать из заранее подготовленного списка профиль аппаратно-программной конфигурации, под которой проявляется дефект. Если такого списка нет или он не содержит необходимых вариантов, можно вручную заполнить поля 6–7–8.

6. Platform (платформа) позволяет указать аппаратную платформу, под которой проявляется дефект.

7. OS (операционная система) позволяет указать операционную систему, под которой проявляется дефект.

8. OS Version (версия операционной системы) позволяет указать версию операционной системы, под которой проявляется дефект.

9. Product version (версия продукта) позволяет указать версию приложения, в которой был обнаружен дефект.

10. Summary (краткое описание) позволяет указать краткое описание дефекта.

11. Description (подробное описание) позволяет указать подробное описание дефекта.

12. Steps to reproduce (шаги по воспроизведению) позволяет указать шаги по воспроизведению дефекта.

13. Additional information (дополнительная информация) позволяет указать любую дополнительную информацию, которая может пригодиться при анализе и устранении дефекта.

14. Upload file (загрузить файл) позволяет загрузить копии экрана и тому подобные файлы, которые могут быть полезны при анализе и устранении дефекта.

15. View status (статус просмотра) позволяет управлять правами доступа к отчёту о дефекте и предлагает по умолчанию два варианта:

- Public (публичный).
- Private (закрытый).

16. Report stay (остаться в режиме добавления отчётов) — отметка этого поля позволяет после сохранения данного отчёта сражу же начать писать следующий.

Тест-план и отчёт о результатах тестирования

Тест-план (test plan) — документ, описывающий и регламентирующий перечень работ по тестированию, а также соответствующие техники и подходы, стратегию, области ответственности, ресурсы, расписание и ключевые даты.

Как и любой другой документ, тест-план может быть качественным или обладать недостатками. Качественный тест-план обладает большинством свойств качественных требований, а также расширяет их набор следующими пунктами:

- Реалистичность (запланированный подход реально выполним).
- Гибкость (качественный тест-план не только является модифицируемым с точки зрения работы с документом, но и построен таким образом, чтобы при возникновении непредвиденных обстоятельств допускать быстрое изменение любой из своих частей без нарушения взаимосвязи с другими частями).
- Согласованность с общим проектным планом и иными отдельными планами (например, планом разработки).

Тест-план создаётся в начале проекта и дорабатывается по мере необходимости на протяжении всего времени жизни проекта при участии наиболее квалифицированных представителей проектной команды, задействованных в обеспечении качества. Ответственным за создание тест-плана, как правило, является ведущий тестировщик

В общем случае тест-план включает следующие разделы

Цель

Области, подвергаемые тестированию

Области, не подвергаемые тестированию

Тестовая стратегия и подходы

Ресурсы

Роль о ответственность

Оценка рисков

Документация

Метрики

Метрики в тестировании являются крайне важными. Итак. **Метрика (**metric) — числовая характеристика показателя качества. Может включать описание способов оценки и анализа результата.

Метрики могут быть как прямыми (т.е. не требовать вычислений), так и расчётными (т.е. вычисляться по формуле). Типичные примеры прямых метрик — количество разработанных тест-кейсов, количество найденных дефектов и т.д. В расчётных метриках могут использоваться как совершенно тривиальные, так и довольно сложные формулы

И, наконец, стоит упомянуть про так называемые «метрики покрытия», т.к. они очень часто упоминаются в различной литературе. **Покрытие (**coverage) — процентное выражение степени, в которой исследуемый элемент (coverage item) затронут соответствующим набором тест-кейсов.

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

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.

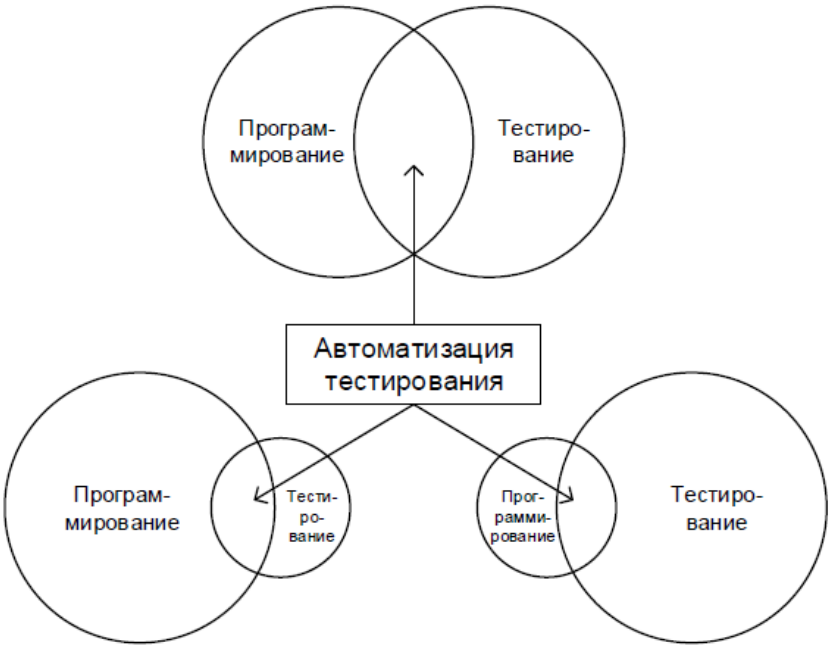
Отчёт о результатах тестирования в первую очередь нужен следующим лицам:

- 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;

Там может быть много графиков(изменения метрик и т.д.)

Автоматизация тестирования

Автоматизация тестирования представляет собой сочетание программирования и тестирования в разных масштабах (в зависимости от проекта и конкретных задач).



десятки и сотни технологий и подходов (как заимствованных из смежных дисциплин, так и уникальных) появляются и исчезают очень стремительно. Количество инструментальных средств автоматизации тестирования уже исчисляется тысячами и продолжает неуклонно расти

Преимущества и недостатки автоматизации

Итак, с использованием автоматизации мы получаем возможность увеличить тестовое покрытие за счёт:

- выполнения тест-кейсов, о которых раньше не стоило и думать;
- многократного повторения тест-кейсов с разными входными данными;
- высвобождения времени на создание новых тест-кейсов.

Но всё ли так хорошо с автоматизацией? Увы, нет.

В первую очередь следует осознать, что автоматизация не происходит сама по себе, не существует волшебной кнопки, нажатием которой решаются все проблемы. Более того, с автоматизацией тестирования связана серия серьёзных недостатков и рисков:

- Необходимость наличия высококвалифицированного персонала в силу того факта, что автоматизация — это «проект внутри проекта» (со своими требованиями, планами, кодом и т.д.). Даже если забыть на мгновение про «проект внутри проекта», техническая квалификация сотрудников, занимающихся автоматизацией, как правило, должна быть ощутимо выше, чем у их коллег, занимающихся ручным тестированием.

- Разработка и сопровождение как самих автоматизированных тест-кейсов, так и всей необходимой инфраструктуры занимает очень много времени. Ситуация усугубляется тем, что в некоторых случаях (при серьёзных изменениях в проекте или в случае ошибок в стратегии) всю соответствующую работу приходится выполнять заново с нуля: в случае ощутимого изменения требований, смены технологического домена, переработки интерфейсов (как пользовательских, так и программных) многие тест-кейсы становятся безнадёжно устаревшими и требуют создания заново.

- Автоматизация требует более тщательного планирования и управления рисками, т.к. в противном случае проекту может быть нанесён серьёзный ущерб Коммерческие средства автоматизации стоят ощутимо дорого, а имеющиеся бесплатные аналоги не всегда позволяют эффективно решать поставленные задачи. И здесь мы снова вынуждены вернуться к вопросу ошибок в планировании: если изначально набор технологий и средств автоматизации был выбран неверно, придётся не только переделывать всю работу, но и покупать новые средства автоматизации.

- Средств автоматизации крайне много, что усложняет проблему выбора того или иного средства, затрудняет планирование и определение стратегии тестирования, может повлечь за собой дополнительные временные и финансовые затраты, а также необходимость обучения персонала или найма соответствующих специалистов.

Итак, автоматизация тестирования требует ощутимых инвестиций и сильно повышает проектные риски, а потому существуют специальные подходы по оценке применимости и эффективности автоматизированного тестирования. Если выразить всю их суть очень кратко, то в первую очередь следует учесть:

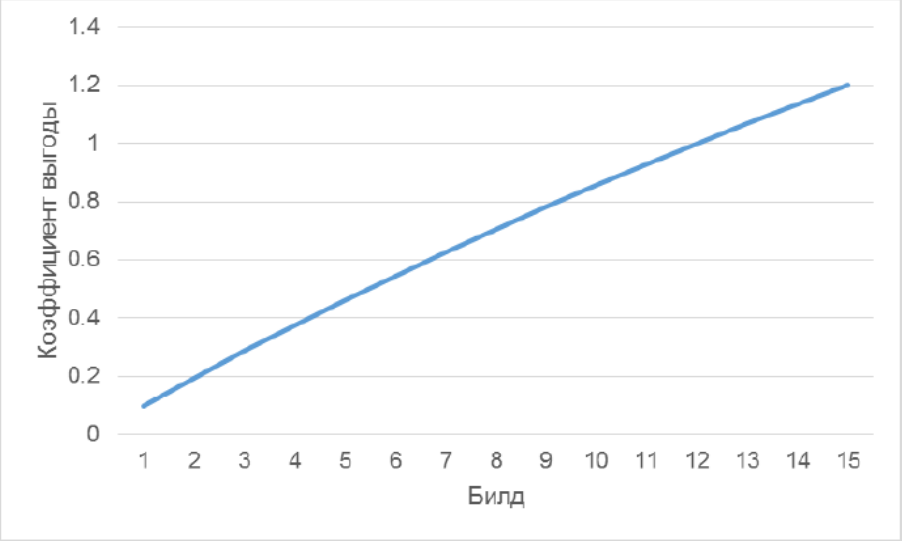
1. Затраты времени на ручное выполнение тест-кейсов и на выполнение этих же тест-кейсов, но уже автоматизированных. Чем ощутимее разница, тем более выгодной представляется автоматизация.

2. Количество повторений выполнения одних и тех же тест-кейсов. Чем оно больше, тем больше времени мы сможем сэкономить за счёт автоматизации.

3. Затраты времени на отладку, обновление и поддержку автоматизированных тест-кейсов. Этот параметр сложнее всего оценить, и именно он представляет наибольшую угрозу успеху автоматизации, потому здесь для проведения оценки следует привлекать наиболее опытных специалистов.

4. Наличие в команде соответствующих специалистов и их рабочую загрузку. Автоматизацией занимаются самые квалифицированные сотрудники, которые в это время не могут решать иные задачи.

Коэффициент выгоды автоматизации в зависимости от количества билдов



Как видно на рисунке лишь к 12-му билду автоматизация окупит вложения и с 13-го билда начнёт приносить пользу. И тем не менее существуют области, в которых автоматизация даёт ощутимый эффект почти сразу.

Сначала мы ещё раз посмотрим на список задач, решить которые помогает автоматизация:

 Выполнение тест-кейсов, непосильных человеку.

 Решение рутинных задач.

 Ускорение выполнения тестирования.

 Высвобождение человеческих ресурсов для интеллектуальной работы.

 Увеличение тестового покрытия.

 Улучшение кода за счёт увеличения тестового покрытия и применения специальных техник автоматизации.

Эти задачи чаще всего встречаются и проще всего решаются в следующих случаях

- Регрессионное тестирование
- Дымовой тест для крупных систем.
- Использование комбинаторных техник тестирования (в т.ч. доменного тестирования)
- Интеграционное тестирование
- Тестирование производительности.

С другой стороны, существуют случаи, в которых автоматизация, скорее всего, приведёт только к ухудшению ситуации. Вкратце — это все те области, где требуется человеческое мышление

Случай / задача	В чём проблема автоматизации
Планирование[195].	Компьютер пока не научился думать.
Разработка тест-кейсов[114].	
Написание отчётов о дефектах[160].	
Анализ результатов тестирования и отчётность[195].	
Функциональность, которую нужно (достаточно) проверить всего несколько раз.	Затраты на автоматизацию не окупятся.
Тест-кейсы, которые нужно выполнить всего несколько раз (если человек может их выполнить).	
Низкий уровень абстракции в имеющихся инструментах автоматизации.	Придётся писать очень много кода, что не только сложно и долго, но и приводит к появлению множества ошибок в самих тест-кейсах.
Слабые возможности средства автоматизации по протоколированию процесса тестирования и сбору технических данных о приложении и окружении.	Есть риск получить данные в виде «что-то где-то сломалось», что не помогает в диагностике проблемы.
Низкая стабильность требований.	Придётся очень многое переделывать, что в случае автоматизации обходится дороже, чем в случае ручного тестирования.
Сложные комбинации большого количества технологий.	Высокая сложность автоматизации, низкая надёжность тест-кейсов, высокая сложность оценки трудозатрат и прогнозирования рисков.
Проблемы с планированием и ручным тестированием.	Автоматизация хаоса приводит к появлению автоматизированного хаоса, но при этом ещё и требует трудозатрат. Сначала стоит решить имеющиеся проблемы, а потом включаться в автоматизацию.
Нехватка времени и угроза срыва сроков	Автоматизация не приносит мгновенных результатов. Поначалу она лишь потребляет ресурсы команды (в том числе время). Также есть универсальный афоризм: «лучше руками протестировать хоть что-то, чем автоматизированно протестировать ничего».
Области тестирования, требующие оценки	В принципе, можно разработать некие алгоритмы, оценивающие ситуацию так, как её мог бы оценить

Использование фреймворков

Фреймворки автоматизации тестирования представляют собой ничто иное, как успешно развившиеся и ставшие популярными решения, объединяющие в себе лучшие стороны тестирования под управлением данными, ключевыми словами и возможности реализации частных решений на высоком уровне абстракции.

Фреймворков автоматизации тестирования очень много, они очень разные, но их объединяет несколько общих черт:

- высокая абстракция кода (нет необходимости описывать каждое элементарное действие) с сохранением возможности выполнения низкоуровневых действий;
- универсальность и переносимость используемых подходов;
- достаточно высокое качество реализации (для популярных фреймворков).

Как правило, каждый фреймворк специализируется на своём виде тестирования, уровне тестирования, наборе технологий. Существуют фреймворки для мо-дульного тестирования (например, семейство xUnit), тестирования веб-приложений (например, семейство Selenium), тестирования мобильных приложений, тестирования производительности и т.д.

Существуют бесплатные и платные фреймворки, оформленные в виде библиотек на некотором языке программирования или в виде привычных приложений с графическим интерфейсом, узко- и широко специализированные и т.д.

Инструментарий

- JUnit — тестирование приложений для Java
- TestNG — тестирование приложений для Java
- NUnit — порт JUnit под .NET
- Selenium — тестирование приложений HTML; поддерживает браузеры Internet Explorer, Mozilla Firefox, Opera, Google Chrome, Safari.
- TOSCA Testsuite — тестирование приложений HTML, .NET, Java, SAP
- Imagrium: Фреймворк для автоматизации кросс-платформенного тестирования мобильных приложений

Еще есть такая технология, как:

Запись и воспроизведение (Record & Playback)

Технология записи и воспроизведения (Record & Playback) стала актуальной с появлением достаточно сложных средств автоматизации, обеспечивающих глубокое

взаимодействие с тестируемым приложением и операционной системой. Использование этой технологии, как правило, сводится к следующим основным шагам:

1. Тестировщик вручную выполняет тест-кейс, а средство автоматизации записывает все его действия.
2. Результаты записи представляются в виде кода на высокоуровневом языке программирования (в некоторых средствах — специально разработанном).
3. Тестировщик редактирует полученный код.
4. Готовый код автоматизированного тест-кейса выполняется для проведения тестирования в автоматизированном режиме.

Сама технология при достаточно высокой сложности внутренней реализации очень проста в использовании и по самой своей сути, потому часто применяется для обучения начинающих специалистов по автоматизации тестирования.

технология записи и воспроизведения не является универсальным средством на все случаи жизни и не может заменить ручную работу по написанию кода автоматизированных тест-кейсов, но в отдельных ситуациях может сильно ускорить решение простых рутинных задач.

Работа тестировщика

Поскольку чем выше квалификация специалиста, тем шире его выбор мест работы (даже в рамках одной крупной фирмы).

В начале карьеры любой специалист (и тестировщик не является исключением) является исполнителем и учеником. Достаточно хорошо понимать, что такое тест-кейсы, отчёты о дефектах, уметь читать требования, пользоваться парой инструментальных средств и хорошо уживаться в команде.

Постепенно тестировщик начинает погружаться во все стадии разработки проекта, понимая их всё полнее и полнее, начинает не только активно использовать, но и разрабатывать проектную документацию, принимать всё более ответственные решения.

Если выразить образно главную цель тестировщика, то она будет звучать так: «понимать, что в настоящий момент необходимо проекту, получает ли проект это необходимое в должной мере, и если нет, как изменить ситуацию к лучшему».

Навыки хорошего тестировщика:

1) Знание иностранных языков. Можете считать это аксиомой: «нет знания английского — нет карьеры в IT». Другие иностранные языки тоже приветствуются, но английский первичен.

2) Уверенное владение компьютером на уровне по-настоящему продвинутого пользователя и желание постоянно развиваться в этой области. Странно выглядит «IT'шник», неспособный набрать вменяемо отформатированный текст, скопировать файл по сети, развернуть виртуальную машину или выполнить любое иное повседневное рутинное действие.

3) Программирование. Оно на порядки упрощает жизнь любому IT'шнику — и тестировщику в первую очередь. Можно ли тестировать без знания про-граммирования? Да, можно. Можно ли это делать по-настоящему хорошо? Нет. Изучения лучше начать с языка,на котором написан ваш проект. Если проекта пока ещё нет, начинайте с JavaScript (на текущий момент — это самое универсальное решение).

4) Базы данных и язык SQL. Здесь от тестировщика тоже не требуется квалификация на уровне узких специалистов, но минимальные навыки работы с наиболее распространёнными СУБД и умение писать простые запросы можно считать обязательными.

5) Понимание принципов работы сетей и операционных систем. Хотя бы на минимальном уровне, позволяющем провести диагностику проблемы и решить её своими силами, если это возможно.

6) Понимание принципов работы веб-приложений и мобильных приложений. В наши дни почти всё пишется именно в виде таких приложений, и понимание соответствующих технологий становится обязательным для эффективного тестирования.

7) Тестирование. Что в целом логично.

В добавок к этому личностные качества, такие как:

1) повышенная ответственность и исполнительность;

2) хорошие коммуникативные навыки, способность ясно, быстро, чётко выражать свои мысли;

3) терпение, усидчивость, внимательность к деталям, наблюдательность;

4) хорошее абстрактное и аналитическое мышление;

5) способность ставить нестандартные эксперименты, склонность к исследовательской деятельности.

Понятное дело, что сложно найти человека, который бы в равной мере обладал всеми перечисленными качествами, но всегда полезно иметь некий ориентир для саморазвития.