Список вопросов

1. Что такое проект?
   ИТ-**проект** – это краткосрочное усилие по созданию уникального продукта, сервиса или среды, например, замещение старых сервисов новыми, разработка коммерческого сайта, создание новых видов настольных компьютеров или слияние баз данных.
2. Перечислите известные вам модели разработки ПО, их особенности, достоинства и недостатки.

Модель разработки ПО (Soft ware 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, редметной области, доступных ресурсов и множества других факторов.
Моделей разработки ПО много, но в общем случае классическими можно считать водопадную, v-образную, итерационную инкрементальную, спиральную и гибкую.
Водопадная модель (waterfall model19) сейчас представляет скорее исторический интерес, т.к. в современных проектах практически неприменима. Она предполагает однократное выполнение каждой из фаз проекта, которые, в свою очередь, строго следуют друг за другом (рисунок 2.1.a). Очень упрощённо можно сказать, что в рамках этой модели в любой момент времени команде «видна» лишь предыдущая и следующая фаза. В реальной же разработке ПО приходится «видеть весь проект целиком» и возвращаться к предыдущим фазам, чтобы исправить недоработки или что-то уточнить.
К недостаткам водопадной модели принято относить тот факт, что участие пользователей ПО в ней либо не предусмотрено вообще, либо предусмотрено лишь косвенно на стадии однократного сбора требований. С точки зрения же тестирования эта модель плоха тем, что тестирование в явном виде появляется здесь лишь с середины развития проекта, достигая своего максимума в самом конце.

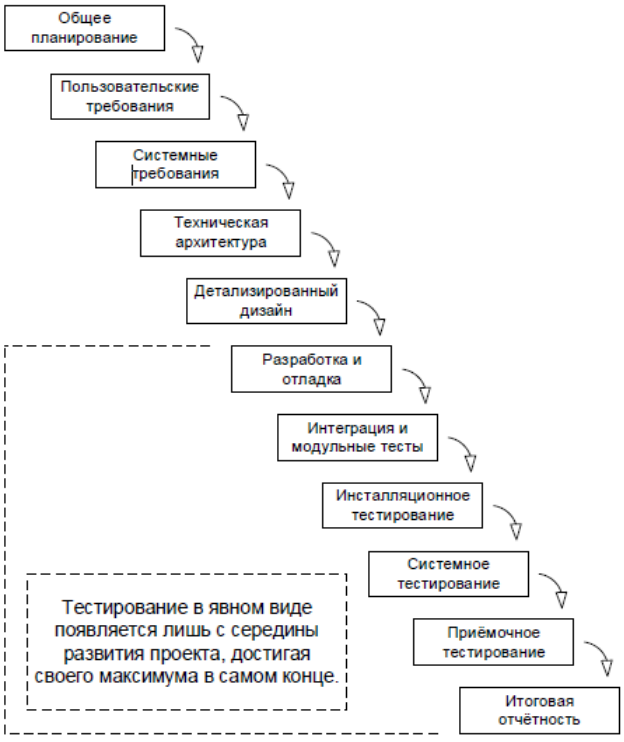


*Рисунок 2.1.a — **Водопадная модель разработки ПО***

V-образная модель (V-model22) является логическим развитием водопадной. Можно заметить (рисунок 2.1.b), что в общем случае как водопадная, так и v-образная модели жизненного цикла ПО могут содержать один и тот же набор стадий, но принципиальное отличие заключается в том, как эта информация используется в процессе реализации проекта.

Очень упрощённо можно сказать, что при использовании v-образной модели на каждой стадии «на спуске» нужно думать о том, что и как будет происходить на соответствующей стадии «на подъёме». Тестирование здесь появляется уже на самых ранних стадиях развития проекта, что позволяет минимизировать риски, а также обнаружить и устранить множество потенциальных проблем до того, как они станут проблемами реальными.

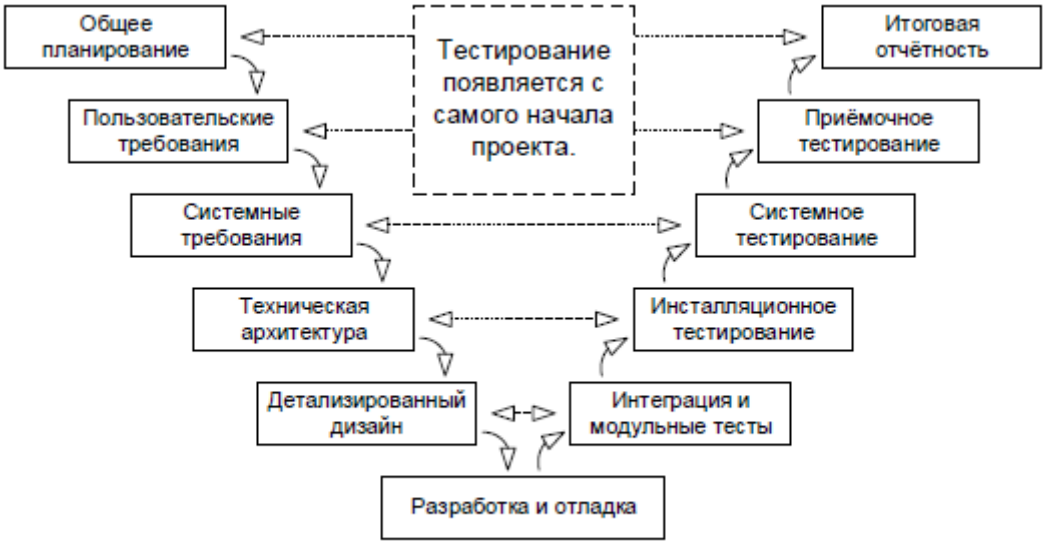


*Рисунок 2.1.b — **V-образная модель разработки ПО***

Итерационная инкрементальная модель (iterative model25, incremental model26) является фундаментальной основой современного подхода к разработке ПО. Как следует из названия модели, ей свойственна определённая двойственность (а ISTQB-глоссарий даже не приводит единого определения, разбивая его на отдельные части):

• с точки зрения жизненного цикла модель является **итерационной**, т.к. подразумевает многократное повторение одних и тех же стадий;

• с точки зрения развития продукта (приращения его полезных функций) модель является **инкрементальной**.

Ключевой особенностью данной модели является разбиение проекта на относительно небольшие промежутки (итерации), каждый из которых в общем случае может включать в себя все классические стадии, присущие водопадной и v-образной моделям (рисунок 2.1.c). Итогом итерации является приращение (инкремент) функциональности продукта, выраженное в промежуточном билде (build27).

Длина итераций может меняться в зависимости от множества факторов, однако сам принцип многократного повторения позволяет гарантировать, что и тестирование, и демонстрация продукта конечному заказчику (с получением обратной связи) будет активно применяться с самого начала и на протяжении всего времени разработки проекта.

Во многих случаях допускается распараллеливание отдельных стадий внутри итерации и активная доработка с целью устранения недостатков, обнаруженных на любой из (предыдущих) стадий.

Итерационная инкрементальная модель очень хорошо зарекомендовала себя на объёмных и сложных проектах, выполняемых большими командами на протяжении длительных сроков. Однако к основным недостаткам этой модели часто относят высокие накладные расходы, вызванные высокой «бюрократизированностью» и общей громоздкостью модели.

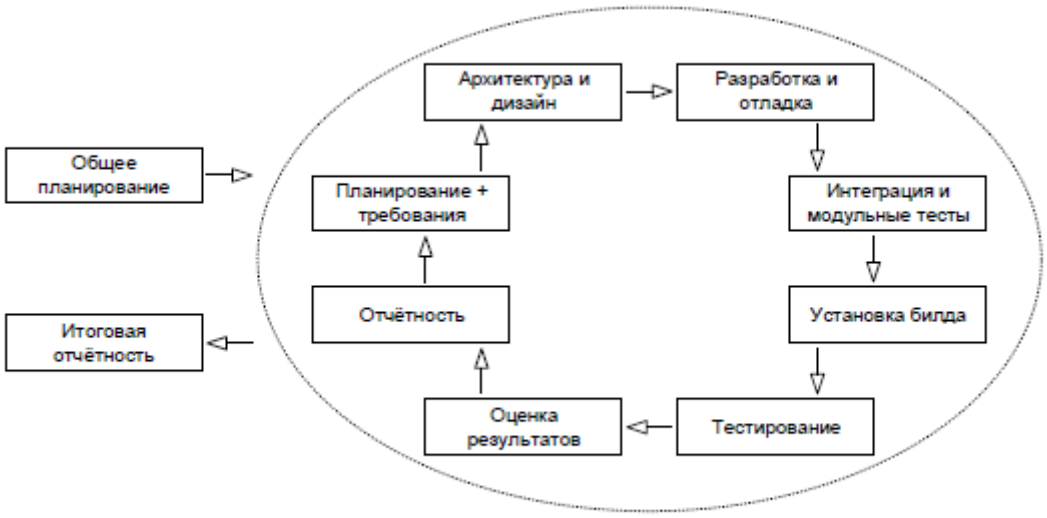


*Рисунок 2.1.c — **Итерационная инкрементальная модель разработки ПО***

Спиральная модель (spiral model30) представляет собой частный случай итерационной инкрементальной модели, в котором особое внимание уделяется управлению рисками, в особенности влияющими на организацию процесса разработки проекта и контрольные точки.
Схематично суть спиральной модели представлена на рисунке 2.1.d. Обратите внимание на то, что здесь явно выделены четыре ключевые фазы:

- проработка целей, альтернатив и ограничений;
- анализ рисков и прототипирование;
- разработка (промежуточной версии) продукта;
- планирование следующего цикла.

С точки зрения тестирования и управления качеством повышенное внимание рискам является ощутимым преимуществом при использовании спиральной модели для разработки концептуальных проектов, в которых требования естественным образом являются сложными и нестабильными (могут многократно меняться по ходу выполнения проекта).

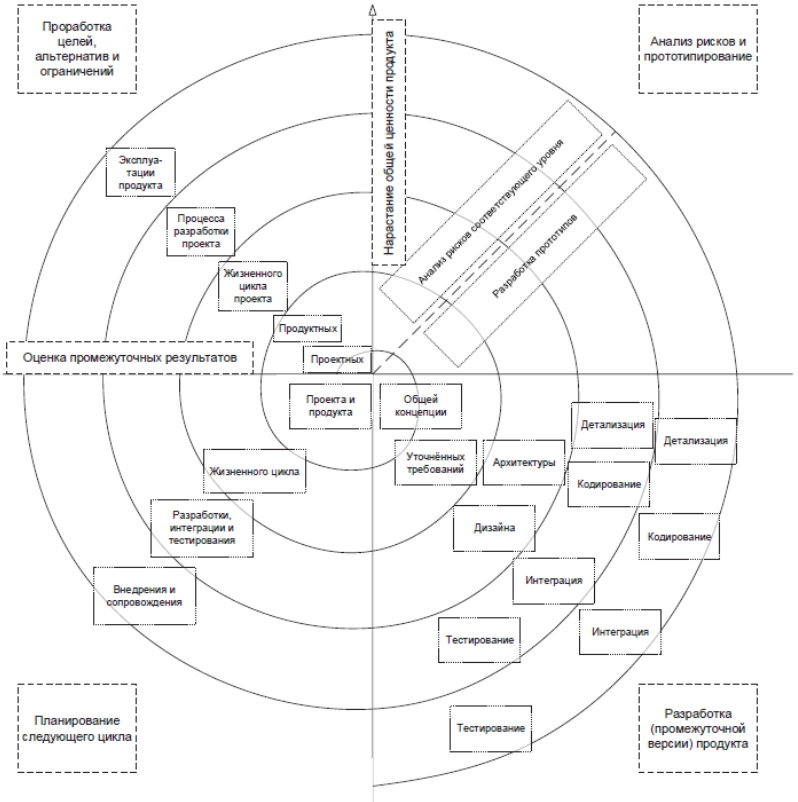
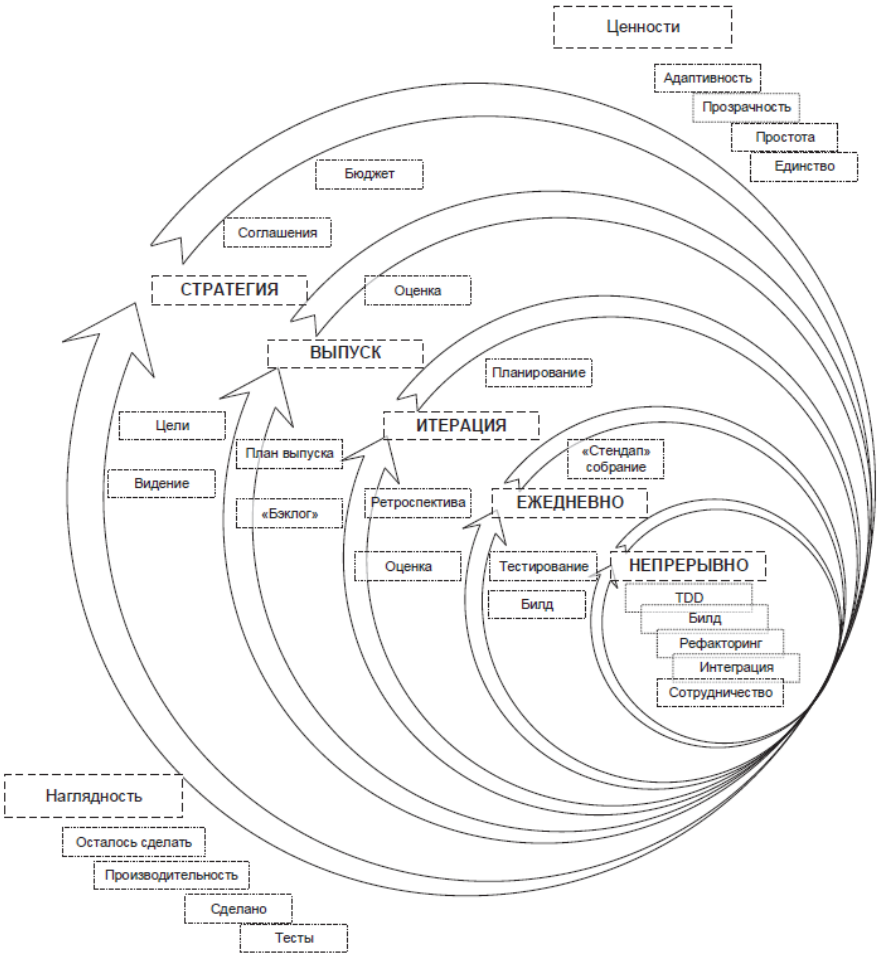


*Рисунок 2.1.d — **Спиральная модель разработки ПО***

Гибкая модель (agile model35) представляет собой совокупность различных подходов к разработке ПО и базируется на т.н. «agile-манифесте»36:
• Люди и взаимодействие важнее процессов и инструментов.
• Работающий продукт важнее исчерпывающей документации.
• Сотрудничество с заказчиком важнее согласования условий контракта.
• Готовность к изменениям важнее следования первоначальному плану.

Как несложно догадаться, положенные в основу гибкой модели подходы являются логическим развитием и продолжением всего того, что было за десятилетия создано и опробовано в водопадной, v-образной, итерационной инкрементальной, спиральной и иных моделях. Причём здесь впервые был достигнут ощутимый результат в снижении бюрократической составляющей и максимальной адаптации процесса разработки ПО к мгновенным изменениям рынка и требований заказчика.
Очень упрощённо (почти на грани допустимого) можно сказать, что гибкая модель представляет собой облегчённую с точки зрения документации смесь итерационной инкрементальной и спиральной моделей (рисунки 2.1.e37, 2.1.f); при этом следует помнить об «agile-манифесте» и всех вытекающих из него преимуществах и недостатках.
Главным недостатком гибкой модели считается сложность её применения к крупным проектам, а также частое ошибочное внедрение её подходов, вызванное недопониманием фундаментальных принципов модели.
Тем не менее можно утверждать, что всё больше и больше проектов начинают использовать гибкую модель разработки.



Вкратце можно выразить суть моделей разработки ПО таблицей 2.1.а.

*Таблица 2.1.а — **Сравнение моделей разработки ПО***

Модель	Преимущества	Недостатки	Тестирование
Водопадная	• У каждой стадии есть чёткий проверяемый результат. • В каждый момент времени команда выполняет один вид работы. • Хорошо работает для небольших задач.	• Полная неспособность адаптировать проект к изменениям в требованиях. • Крайне позднее создание работающего продукта.	• С середины проекта.
V-образная	• У каждой стадии есть чёткий проверяемый результат. • Внимание тестированию уделяется с первой же стадии. • Хорошо работает для проектов со стабильными требованиями.	• Недостаточная гибкость и адаптируемость. • Отсутствует раннее прототипирование. • Сложность устранения проблем, пропущенных на ранних стадиях развития проекта.	• На переходах между стадиями.

Таблица 2.1.а [окончание]

Модель	Преимущества	Недостатки	Тестирование
Итерационная инкрементальная	• Достаточно раннее прототипирование. • Простота управления итерациями. • Декомпозиция проекта на управляемые итерации.	• Недостаточная гибкость внутри итераций. • Сложность устранения проблем, пропущенных на ранних стадиях развития проекта.	• В определённые моменты итераций. • Повторное тестирование (после доработки) уже проверенного ранее.
Спиральная	• Глубокий анализ рисков. • Подходит для крупных проектов. • Достаточно раннее прототипирование.	• Высокие накладные расходы. • Сложность применения для небольших проектов. • Высокая зависимость успеха от качества анализа рисков.	
Гибкая	• Максимальное вовлечение заказчика. • Много работы с требованиями. • Тесная интеграция тестирования и разработки. • Минимизация документации.	• Сложность реализации для больших проектов. • Сложность построения стабильных процессов.	• В определённые моменты итераций **и в любой необходимый момент**.

3. Что такое тестирование?
   Тестирование программного обеспечения — процесс анализа программного средства сопутствующей документации с целью выявления дефектов и повышения качества продукта.
4. Перечислите виды тестирования и дайте им определения.

*Рисунок 2.3.а — **Упрощённая классификация тестирования***

По запуску кода на исполнение:
. Статическое тестирование — без запуска.
. Динамическое тестирование — с запуском.
• По доступу к коду и архитектуре приложения:
. Метод белого ящика — доступ к коду есть.
. Метод чёрного ящика — доступа к коду нет.
. Метод серого ящика — к части кода доступ есть, к части — нет.
• По степени автоматизации:
. Ручное тестирование — тест-кейсы выполняет человек.
. Автоматизированное тестирование — тест-кейсы частично или полностью выполняет специальное инструментальное средство.
По уровню детализации приложения (по уровню тестирования):
. Модульное (компонентное) тестирование — проверяются отдельные небольшие части приложения.
. Интеграционное тестирование — проверяется взаимодействие между несколькими частями приложения.
. Системное тестирование — приложение проверяется как единое целое.
• По (убыванию) степени важности тестируемых функций (по уровню функционального тестирования):
. Дымовое тестирование (обязательно изучите этимологию термина — хотя бы в Википедии109) — проверка самой важной, самой ключевой функциональности, неработоспособность которой делает бессмысленной саму идею использования приложения.
. Тестирование критического пути — проверка функциональности, используемой типичными пользователями в типичной повседневной деятельности.
. Расширенное тестирование — проверка всей (остальной) функциональности, заявленной в требованиях.
• По принципам работы с приложением:
. Позитивное тестирование — все действия с приложением выполняются строго по инструкции без никаких недопустимых действий, некорректных данных и т.д. Можно образно сказать, что приложение исследуется в «тепличных условиях».
. Негативное тестирование — в работе с приложением выполняются (некорректные) операции и используются данные, потенциально приводящие к ошибкам (классика жанра — деление на ноль).

Альфа-тестирование (alpha testing150) выполняется внутри организации-разработчика с возможным частичным привлечением конечных пользователей. Может являться формой внутреннего приёмочного тестирования{89}. 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 testing151) выполняется вне организации-разработчика с активным привлечением конечных пользователей/заказчиков. Может являться формой внешнего приёмочного тестирования{89}. 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 testing152) — финальная стадия тестирования перед вы пуском 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{89}. Суть этого вида вкратце: продукт уже почти готов, и сейчас обратная связь от реальных пользователей используется для устранения последних недоработок.

5. Перечислите методы тестирования и дайте им определения

Метод белого ящика (white box testing119, 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 testing120). Для более глубокого изучения сути метода белого ящика рекомендуется ознакомиться с техниками исследования потока управления{98} или потока данных{98}, использования диаграмм состояний{98}. 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{77} и вовсе предполагает запуск кода на исполнение и при этом работу именно с кодом, а не с «приложением целиком»).
• **Метод чёрного ящика** (black box testing121, closed box testing, specifi cation-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 перечисленных на рисунках 2.3.b и 2.3.c 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 — требования (requirements-based testing122)) и общий здравый смысл (для случаев, когда поведение приложения в некоторой ситуации не регламентировано явно; иногда это называют «тестированием на основе неявных требований», но канонического определения у этого подхода нет).
• **Метод серого ящика** (gray box testing123) — комбинация методов белого ящика и чёрного ящика, состоящая в том, что к части кода и архитектуры у тестировщика доступ есть, а к части — нет. На рисунках 2.3.b и 2.3.c этот метод обозначен особым пунктиром и серым цветом потому, что 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.

6. Перечислите уровни тестирования и дайте им определения

*Рисунок 2.3.е — **Самый полный вариант классификации тестирования по уровню тестирования***

Модульное (компонентное) тестирование (unit testing, module testing, component testing128) 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{75}, значительно упрощающих и ускоряющих разработку соответствующих тест-кейсов.

Интеграционное тестирование (integration testing129, component integration testing130, pairwise integration testing131, system integration testing132, incremental testing133, interface testing134, thread testing135) 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. (См. также техники восходящего, нисходящего и гибридного тестирования в хронологической классификации по иерархии компонентов{103}.)

Системное тестирование (system testing136) 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, перечисленных в данной главе.

7. Что такое качество ПО?

Ка́чество програ́ммного обеспечения — способность программного продукта при заданных условиях удовлетворять установленным или предполагаемым потребностям (ISO/IEC 25000:2014)[1].

Другие определения из стандартов:

- весь объём признаков и характеристик программ, который относится к их способности удовлетворять установленным или предполагаемым потребностям (ГОСТ Р ИСО/МЭК 9126-93, ISO 8402:94)[2][3];
- степень, в которой система, компонент или процесс удовлетворяют потребностям или ожиданиям заказчика или пользователя (IEEE Std 610.12-1990)[4].

Рис.1 – Модель качества программного обеспечения (ISO 9126-1)

8. Что такое требования?
   Требование (requirement49) — описание того, какие функции и с соблюдением каких условий должно выполнять приложение в процессе решения полезной для пользователя задачи.

*Рисунок 2.2.d — **Основные техники сбора и выявления требований***

9. Перечислите свойства требований.

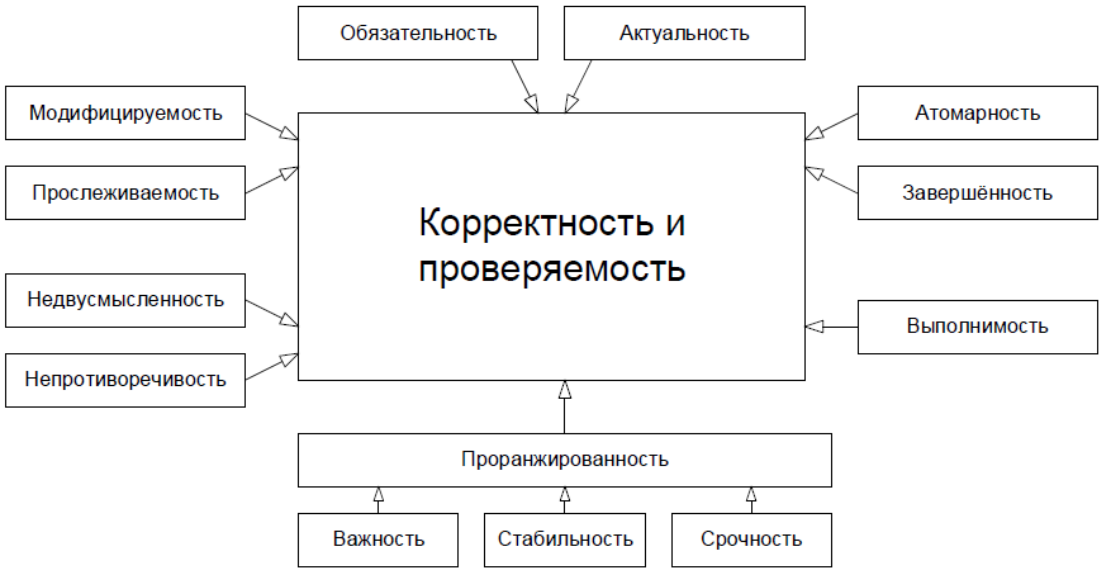


*Рисунок 2.2.f — **Свойства качественного требования***

10. Какие существуют 3 уровня требований, дайте им определения

*Рисунок 2.2.е — **Уровни и типы требований***

Бизнес-требования (business requirements69) выражают цель, ради которой разрабатывается продукт (зачем вообще он нужен, какая от него ожидается польза, как заказчик с его помощью будет получать прибыль). Результатом выявления требований на этом уровне является общее видение (vision and scope70) — документ, который, как правило, представлен простым текстом и таблицами. Здесь нет детализации поведения системы и иных технических характеристик, но вполне могут быть определены приоритеты решаемых бизнес-задач, риски и т.п.
Несколько простых, изолированных от контекста и друг от друга примеров бизнес-требований:
• *Нужен инструмент, в реальном времени отображающий наиболее выгодный курс покупки и продажи валюты.*
• *Необходимо в два-три раза повысить количество заявок, обрабатываемых одним оператором за смену.*

• *Нужно автоматизировать процесс выписки товарно-транспортных накладных на основе договоров.*
Пользовательские требования (user requirements71) описывают задачи, которые пользователь может выполнять с помощью разрабатываемой системы (реакцию системы на действия пользователя, сценарии работы пользователя). Поскольку здесь уже появляется описание поведения системы, требования этого уровня могут быть использованы для оценки объёма работ, стоимости проекта, времени разработки и т.д. Пользовательские требования оформляются в виде вариантов использования (use cases72), пользовательских историй (user stories73), пользовательских сценариев (user scenarios74). (Также см. создание пользовательских сценариев{148} в процессе выполнения тестирования.)
Несколько простых, изолированных от контекста и друг от друга примеров пользовательских требований:
• *При первом входе пользователя в систему должно отображаться лицензионное соглашение.*
• *Администратор должен иметь возможность просматривать список всех пользователей, работающих в данный момент в системе.*
• *При первом сохранении новой статьи система должна выдавать запрос на сохранение в виде черновика или публикацию.*
Бизнес-правила (business rules75) описывают особенности принятых в предметной области (и/или непосредственно у заказчика) процессов, ограничений и иных правил. Эти правила могут относиться к бизнес-процессам, правилам работы сотрудников, нюансам работы ПО и т.д.
Несколько простых, изолированных от контекста и друг от друга примеров бизнес-правил:

• Никакой документ, просмотренный посетителями сайта хотя бы один раз, не может быть отредактирован или удалён.
• Публикация статьи возможна только после утверждения главным редактором.
• Подключение к системе извне офиса запрещено в нерабочее время.

Атрибуты качества (quality attributes[76]) расширяют собой нефункциональные требования и на уровне пользовательских требований могут быть представлены в виде описания ключевых для проекта показателей качества (свойств продукта, не связанных с функциональностью, но являющихся важными для достижения целей создания продукта — производительность, масштабируемость, восстанавливаемость). Атрибутов качества очень много[77], но для любого проекта реально важными является лишь некоторое их подмножество.
Несколько простых, изолированных от контекста и друг от друга примеров атрибутов качества:
• Максимальное время готовности системы к выполнению новой команды после отмены предыдущей не может превышать одну секунду.

Внесённые в текст статьи изменения не должны быть утеряны при нарушении соединения между клиентом и сервером.
• Приложение должно поддерживать добавление произвольного количества неиероглифических
языков интерфейса.
Функциональные требования (functional requirements[78]) описывают поведение системы, т.е. её действия (вычисления, преобразования, проверки, обработку и т.д.). В контексте проектирования функциональные требования в основном влияют на дизайн системы.
Стоит помнить, что к поведению системы относится не только то, что система должна делать, но и то, что она **не** должна делать (например: «приложение **не** должно выгружать из оперативной памяти фоновые документы в течение 30 минут с момента выполнения с ними последней операции»).
Несколько простых, изолированных от контекста и друг от друга примеров функциональных требований:
• В процессе инсталляции приложение должно проверять остаток свободного места на целевом носителе.
• Система должна автоматически выполнять резервное копирование данных ежедневно в указанный момент времени.
• Электронный адрес пользователя, вводимый при регистрации, должен быть проверен на соответствие требованиям RFC822.
Нефункциональные требования (non-functional requirements[79]) описывают свойства системы (удобство использования, безопасность, надёжность, расширяемость и т.д.), которыми она должна обладать при реализации своего поведения. Здесь приводится более техническое и детальное описание атрибутов качества. В контексте проектирования нефункциональные требования в основном влияют на архитектуру системы.
Несколько простых, изолированных от контекста и друг от друга примеров нефункциональных требований:
• При одновременной непрерывной работе с системой 1000 пользователей, минимальное время
между возникновением сбоев должно быть более или равно 100 часов.
• Ни при каких условиях общий объём используемой приложением памяти не может превышать 2 ГБ.
• Размер шрифта для любой надписи на экране должен поддерживать настройку в диапазоне от 5 до 15 пунктов.
Следующие требования в общем случае могут быть отнесены к нефункциональным, однако их часто выделяют в отдельные подгруппы (здесь для простоты рассмотрены лишь три таких подгруппы, но их может быть и гораздо больше; как правило, они проистекают из атрибутов качества, но высокая степень детализации позволяет отнести их к уровню требований к продукту).

Ограничения (limitations, constraints[80]) представляют собой факторы, ограничивающие выбор способов и средств (в том числе инструментов) реализации продукта.
Несколько простых, изолированных от контекста и друг от друга примеров ограничений:
• Все элементы интерфейса должны отображаться без прокрутки при разрешениях экрана от 800x600 до 1920x1080.
• Не допускается использование Flash при реализации клиентской части приложения.

Приложение должно сохранять способность реализовывать функции с уровнем важности «критический» при отсутствии у клиента поддержки JavaScript.
Требования к интерфейсам (external interfaces requirements[81]) описывают особенности взаимодействия разрабатываемой системы с другими системами и операционной средой.
Несколько простых, изолированных от контекста и друг от друга примеров требований к интерфейсам:
• Обмен данными между клиентской и серверной частями приложения при осуществлении фоновых AJAX-запросов должен быть реализован в формате JSON.
• Протоколирование событий должно вестись в журнале событий операционной системы.
• Соединение с почтовым сервером должно выполняться согласно RFC3207 («SMTP over TLS»).
Требования к данным (data requirements[82]) описывают структуры данных (и сами данные), являющиеся неотъемлемой частью разрабатываемой системы. Часто сюда относят описание базы данных и особенностей её использования.
Несколько простых, изолированных от контекста и друг от друга примеров требований к данным:
• Все данные системы, за исключением пользовательских документов, должны храниться в БД под управлением СУБД MySQL, пользовательские документы должны храниться в БД под управлением СУБД MongoDB.
• Информация о кассовых транзакциях за текущий месяц должна храниться в операционной таблице, а по завершении месяца переноситься в архивную.
• Для ускорения операций поиска по тексту статей и обзоров должны быть предусмотрены полнотекстовые индексы на соответствующих полях таблиц.
Спецификация требований (soft ware requirements specifi cation, SRS[83]) объединяет в себе описание всех требований уровня продукта и может представлять собой весьма объёмный документ (сотни и тысячи страниц).
Поскольку требований может быть очень много, а их приходится не только единожды написать и согласовать между собой, но и постоянно обновлять, работу проектной команды по управлению требованиями значительно облегчают соответствующие инструментальные средства (requirements management tools[84, 85]).

11. Что такое баг?

Ошибка (error[309], mistake) — действие человека, приводящее к некорректным результатам.

Дефект (defect[310], bug, problem, fault) — недостаток в компоненте или системе, способный привести к ситуации сбоя или отказа.

Программная ошибка (*жарг.* **баг**) — означает ошибку в программе или в системе, из-за которой программа выдает неожиданное поведение и, как следствие, результат.

Сбой (interruption[311]) или **отказ** (failure[312]) — отклонение поведения системы от ожидаемого.
В ГОСТ 27.002-89 даны хорошие и краткие определения сбоя и отказа:
Сбой — самоустраняющийся отказ или однократный отказ, устраняемый незначительным вмешательством оператора.
Отказ — событие, заключающееся в нарушении работоспособного состояния объекта.

12. Что такое тест кейс?

Тест-кейс[122]	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.

Тест-кейс (test case286) — 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).

13. Перечислите основные атрибуты тест кейсов.

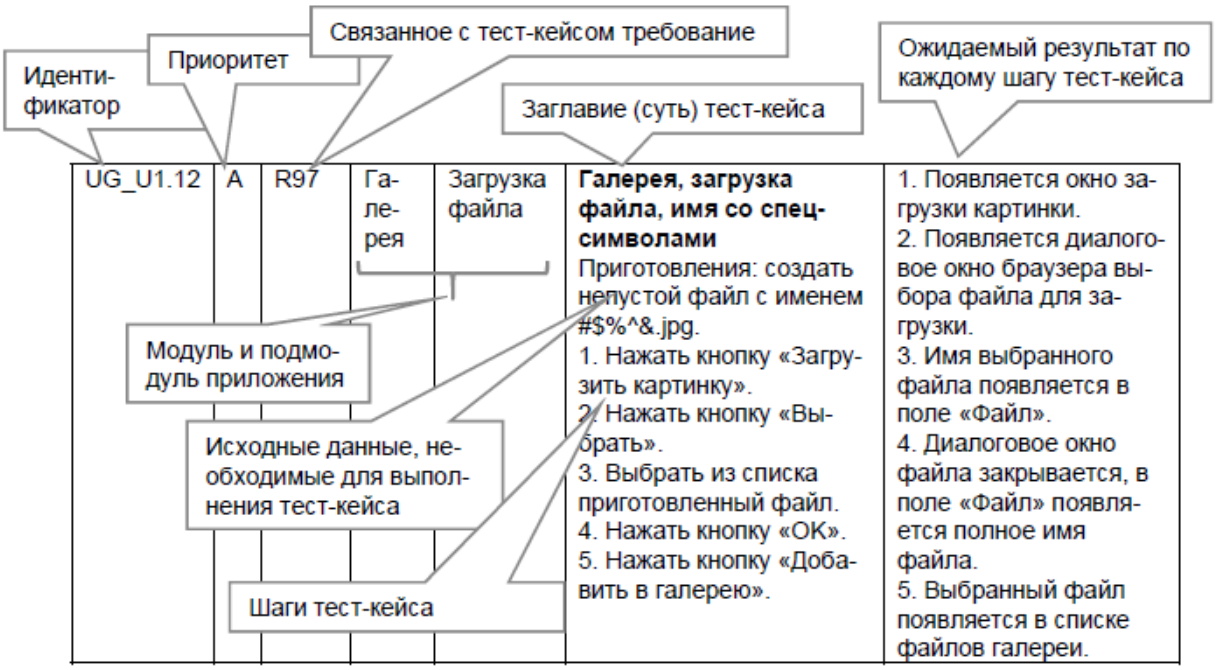


*Рисунок 2.4.b — **Общий вид тест-кейса***

Идентификатор (identifi er) 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{148} или функции, с которыми связан тест-кейс;
• потенциальной важностью дефекта{180}, 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 — потому, что один тест-кейс может затрагивать несколько требований). Наличие этого поля улучшает такое свойство тест-кейса, как прослеживаемость{145}.
Частые вопросы, связанные с заполнением этого поля, таковы:
• Можно ли его оставить пустым? Да. Тест-кейс вполне мог разрабатываться вне прямой привязки к требованиям, и (пока?) значение этого поля определить сложно. Хоть такой вариант и не считается хорошим, он достаточно распространён.
• Можно ли в этом поле указывать несколько требований? Да, но чаще всего стараются выбрать одно самое главное или «более высокоуровневое» (например, вместо того, чтобы перечислять R56.1, R56.2, R56.3 и т.д., можно просто написать R56). Чаще всего в инструментах управления тестами это поле представляет собой выпадающий список, где можно выбрать только одно значение, и этот вопрос становится неактуальным. К тому же многие тест-кейсы всё же направлены на проверку строго одного требования, и для них этот вопрос также неактуален.
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 (суть) тест-кейса (title) призвано упростить и ускорить понимание основной идеи (цели) 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, 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
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.

14. Что такое чек-лист?

Чек-лист (checklist281) — набор идей [тест-кейсов]. Последнее слово не зря взято в

скобки282, 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:

• в котором последовательность пунктов не имеет значения (например, список значений некоего поля);

• в котором последовательность пунктов важна (например, шаги в краткой инструкции);

• структурированный (многоуровневый) список, который позволяет отразить иерархию идей.

Очень частым является вопрос о том, нужно ли в чек-листах писать ожидаемые результаты. В классическом понимании чек-листа – нет (хоть это и не запрещено), т.к. чек-лист – это набор идей, а их детализация в виде шагов и ожидаемых результатов будет в тест-кейсах. Но ожидаемые результаты могут добавляться, например, в сле дующих случаях:

• в некоем пункте чек-листа рассматривается особое, нетривиальное поведение приложение или сложная проверка, результат которой важно отметить уже сейчас, чтобы не забыть;

• в силу сжатых сроков и/или нехватки иных ресурсов тестирование проводится напрямую по чек-листам без тест-кейсов.

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 до 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{83}, потом идеи простых негативных тест-кейсов, потом постепенно повышать сложность тест-кейсов, но не стоит писать эти идеи вперемешку).

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.

15. Что такое баг репорт?

Отчёт о дефекте (defect report317) — 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;

• приоритизировать проблему — определить степень опасности проблемы для проекта и желаемые сроки её устранения;

• содействовать устранению проблемы — качественный отчёт о дефекте не только предоставляет все необходимые подробности для понимания сути случившегося, но также может содержать анализ причин возникновения проблемы и рекомендации по исправлению ситуации.

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 (fi xed) — в это состояние отчёт переводит ответственный за исправление дефекта член команды после выполнения соответствующих действий по исправлению.

• Проверен (verifi ed) — 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 исправлен.

• 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» (см. следующий пункт).

• 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 (выйдет новая версия библиотеки, вернёт

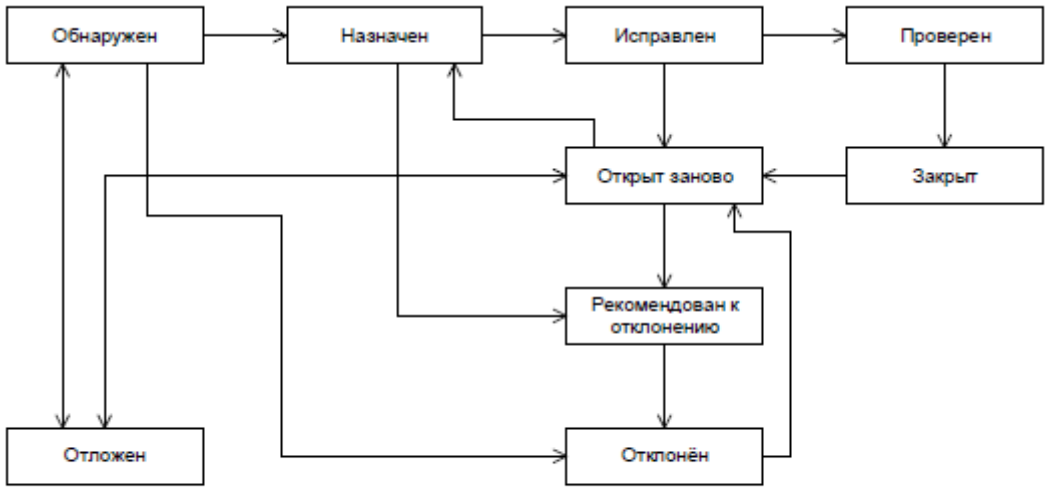


*Рисунок 2.5.c — **Жизненный цикл отчёта о дефекте с наиболее типичными переходами между состояниями***

16. Перечислите основные атрибуты баг-репорта?

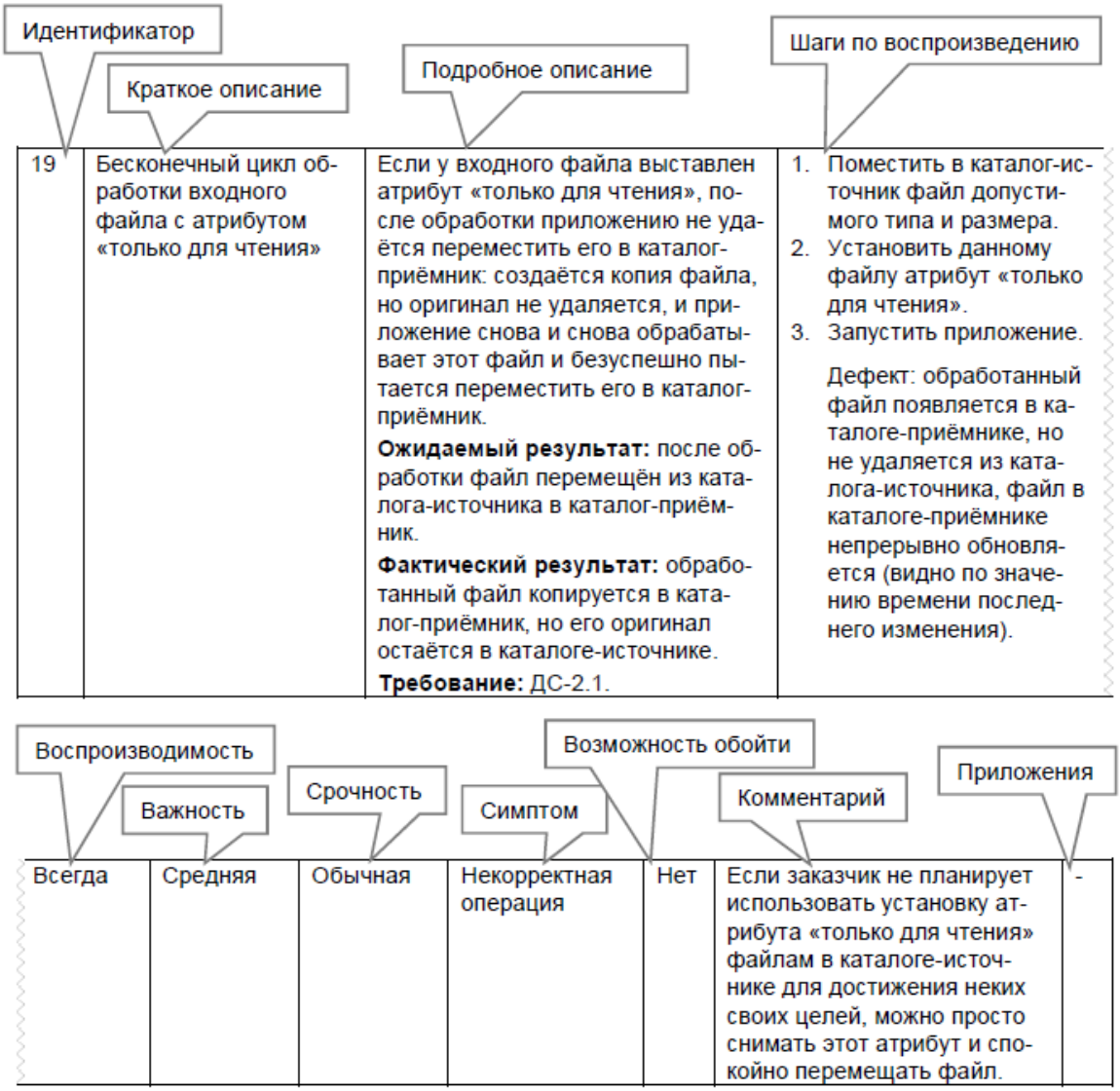


Рисунок 2.5.е — Общий вид отчёта о дефекте

Идентификатор (identifi er) представляет собой уникальное значение, позволяющее однозначно отличить один отчёт о дефекте от другого и используемое во всевозможных ссылках. В общем случае идентификатор отчёта о дефекте может представлять собой просто уникальный номер, но (если позволяет инструментальное средство управления отчётами) может быть и куда сложнее: включать префиксы, суффиксы и иные осмысленные компоненты, позволяющие быстро определить суть дефекта и часть приложения (или требований), к которой он относится.

Краткое описание (summary) должно в предельно лаконичной форме давать исчерпывающий ответ на вопросы «Что произошло?» «Где это произошло»? «При каких условиях это произошло?». Например: «Отсутствует логотип на странице приветствия, если пользователь является администратором»:

• Что произошло? Отсутствует логотип.

• Где это произошло? На странице приветствия.

• При каких условиях это произошло? Если пользователь является администратором.

Подробное описание (description) представляет в развёрнутом виде необходимую информацию о дефекте, а также (обязательно!) описание фактического результата, ожидаемого результата и ссылку на требование (если это возможно).

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).

Симптом (symptom) — позволяет классифицировать дефекты по их типичному проявлению. Не существует никакого общепринятого списка симптомов. Более того, далеко не в каждом инструментальном средстве управления отчётами о дефектах есть такое поле, а там, где оно есть, его можно настроить. В качестве примера рассмотрим следующие значения симптомов дефекта.

• Косметический дефект (cosmetic fl aw) — визуально заметный недостаток интерфейса, не влияющий на функциональность приложения (например, надпись на кнопке выполнена шрифтом не той гарнитуры).

• Повреждение/потеря данных (data corruption/loss) — в результате возникновения дефекта искажаются, уничтожаются (или не сохраняются) некоторые данные (например, при копировании файлов копии оказываются повреждёнными).

• Проблема в документации (documentation issue) — дефект относится не к приложению, а к документации (например, отсутствует раздел руководства по эксплуатации).

• Некорректная операция (incorrect operation) — некоторая операция выполняется некорректно (например, калькулятор показывает ответ 17 при умножении 2 на 3).

• Проблема инсталляции (installation problem) — дефект проявляется на стадии установки и/или конфигурирования приложения (см. инсталляционное тестирование{88}).

• Ошибка локализации (localization issue) — что-то в приложении не переведено или переведено неверно на выбранный язык интерфейса (см. локализационное тестирование{91}).

• Нереализованная функциональность (missing feature) — некая функция приложения не выполняется или не может быть вызвана (например, в списке форматов для экспорта документа отсутствует несколько пунктов, которые там должны быть).

• Проблема масштабируемости (scalability) — при увеличении количества доступных приложению ресурсов не происходит ожидаемого прироста производительности приложения (см. тестирование производительности{93} и тестирование масштабируемости{94}).

• Низкая производительность (low performance) — выполнение неких операций занимает недопустимо большое время (см. тестирование производительности{93}).

Крах системы (system crash) — приложение прекращает работу или теряет способность выполнять свои ключевые функции (также может сопровождаться крахом операционной системы, веб-сервера и т.д.).

• Неожиданное поведение (unexpected behavior) — в процессе выполнения некоторой типичной операции приложение ведёт себя необычным (отличным от общепринятого) образом (например, после добавления в список новой записи активной становится не новая запись, а первая в списке).

• Недружественное поведение (unfriendly behavior) — поведение приложения создаёт пользователю неудобства в работе (например, на разных диалоговых окнах в разном порядке расположены кнопки «OK» и «Cancel»).

• Расхождение с требованиями (variance from specs) — этот симптом указывают, если дефект сложно соотнести с другими симптомами, но тем не менее приложение ведёт себя не так, как описано в требованиях.

• Предложение по улучшению (enhancement) — во многих инструментальных средствах управления отчётами о дефектах для этого случая есть отдельный вид отчёта, т.к. предложение по улучшению формально нельзя считать дефектом: приложение ведёт себя согласно требованиям, но у тестировщика есть обоснованное мнение о том, как ту или иную функциональность можно улучшить.

Симптом	Важность	Воспроизводимо	Срочность
Incorrect operation	Critical	Always	ASAP
Data loss	Major	Sometimes	High
Cosmetic Flaw	Medium		Normal
Documentation issue	Minor		Low
Installation problem			
Localization issue			
Missing feature			
Slow performance			
System crash			
Unexpected behavior			
Unfriendly behavior			
Variance from specs			

Возможность обойти (workaround) — показывает, существует ли альтернативная последовательность действий, выполнение которой позволило бы пользователю достичь поставленной цели (например, клавиатурная комбинация Ctrl+P не работает, но распечатать документ можно, выбрав соответствующие пункты в меню). 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.
Вложения (attachments) — представляет собой не столько поле, сколько список прикреплённых к отчёту о дефекте приложений (копий экрана, вызывающих сбой файлов и т.д.).

17. Что такое Важность? Срочность? Чем они отличаются?

Важность (severity) показывает степень ущерба, который наносится проекту существованием дефекта.
В общем случае выделяют следующие градации важности:
• Критическая (critical) — существование дефекта приводит к масштабным последствиям катастрофического характера, например: потеря данных, раскрытие конфиденциальной информации, нарушение ключевой функциональности приложения и т.д.
• Высокая (major) — существование дефекта приносит ощутимые неудобства многим пользователям в рамках их типичной деятельности, например: недоступность вставки из буфера обмена, неработоспособность общепринятых клавиатурных комбинаций, необходимость перезапуска приложения при выполнении типичных сценариев работы.
• Средняя (medium) — существование дефекта слабо влияет на типичные сценарии работы пользователей, и/или существует обходной путь достижения цели, например: диалоговое окно не закрывается автоматически после нажатия кнопок «OK»/«Cancel», при распечатке нескольких документов подряд не сохраняется значение поля «Двусторонняя печать», перепутаны направления сортировок по некоему полю таблицы.
• Низкая (minor) — существование дефекта редко обнаруживается незначительным процентом пользователей и (почти) не влияет на их работу, например: опечатка в глубоко вложенном пункте меню настроек, некое окно сразу при отображении расположено неудобно (нужно перетянуть его в удобное место), неточно отображается время до завершения операции копирования файлов.
Срочность (priority) показывает, как быстро дефект должен быть устранён.
В общем случае выделяют следующие градации срочности:
• Наивысшая (ASAP, as soon as possible) срочность указывает на необходимость устранить дефект настолько быстро, насколько это возможно. В зависимости от контекста «настолько быстро, насколько возможно» может варьироваться от «в ближайшем билде» до единиц минут.

• Высокая (high) срочность означает, что дефект следует исправить вне очереди, т.к. его существование или уже объективно мешает работе, или начнёт создавать такие помехи в самом ближайшем будущем.
• Обычная (normal) срочность означает, что дефект следует исправить в порядке общей очерёдности. Такое значение срочности получает большинство дефектов.
• Низкая (low) срочность означает, что в обозримом будущем исправление данного дефекта не окажет существенного влияния на повышение качества продукта.__

*Таблица 2.5.b — **Примеры сочетания важности и срочности дефектов***

		Важность	
		Критическая	**Низкая**
Срочность	**Наивысшая**	Проблемы с безопасностью во введённом в эксплуатацию банковском ПО.	На корпоративном сайте повредилась картинка с фирменным логотипом.
	Низкая	В самом начале разработки проекта обнаружена ситуация, при которой могут быть повреждены или вовсе утеряны пользовательские данные.	В руководстве пользователя обнаружено несколько опечаток, не влияющих на смысл текста.

18. Опишите принципы построения краткого описания баг-репорта.

19. Чем отличается краткое описание от подробного?
20. Что входит в подробное описание баг-репорта?

21. Что такое тест план?

Тест-план
Тест-план является частью проектной документации и основным документом в тестировании, описывающим весь объем работ по тестированию.
Цели создания тест-плана:
 Согласование объёмов и стратегии тестирования различных составляющих тестируемого ПО с другими участниками проектной команды
 Приоритезация задач по тестированию
 Своевременное планирование ресурсозатрат на тестирование
 Учёт требуемых ресурсов (ПО, оборудование), необходимых для тестирования

Заблаговременный учёт рисков, которые могут возникнуть в процессе реализации плана, и внедрение предупреждающей стратегии
Содержание тест-плана
Документ должен как минимум отвечать на следующие вопросы:
 Что надо тестировать (объект тестирования: система, приложение, оборудование)
 Что будете тестировать (список функций и компонентов тестируемой системы)
 Как будете тестировать (стратегия тестирования – виды тестирования и их применение по отношению к тестируемому объекту)
 Тестовые окружения, на которых необходимо проверять программный продукт
 Когда будете тестировать (последовательность проведения работ: подготовка, тестирование, анализ результатов, учёт зависимостей тестовых активностей от задач разработки и смежных групп)

 Риски и стратегии по их разрешению.

22. Что такое отчет о тестировании?

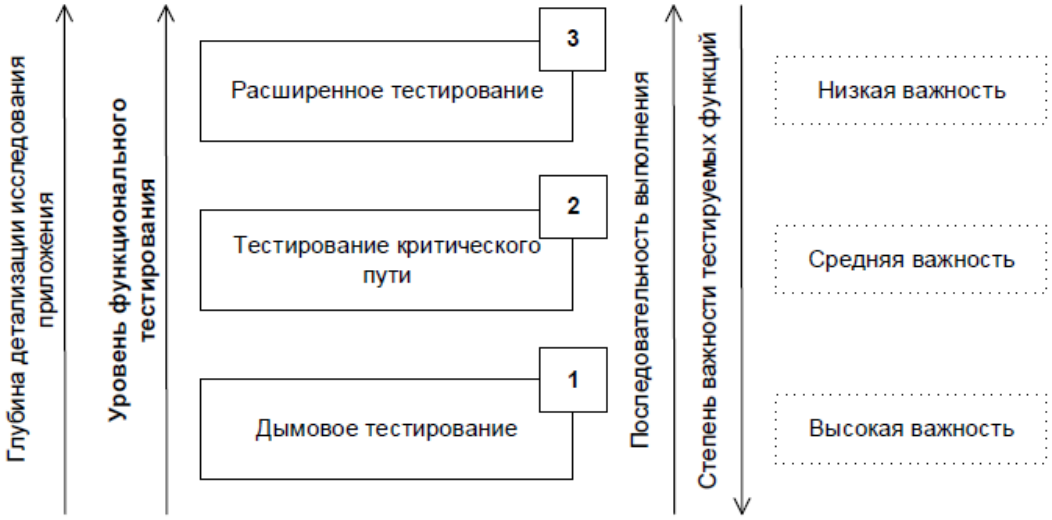


Рисунок 2.3.h — Классификация тестирования по (убыванию) степени важности тестируемых функций (по уровню функционального тестирования)

23. Что такое smoke test?

Дымовое тестирование (smoke test[139], intake test[140], build verifi cation test[141]) 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 (не)целесообразности 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 %.

24. Что такое critical path test?

Тестирование критического пути (critical path[144] test) направлено на исследование функциональности, используемой типичными пользователями в типичной повседневной деятельности. Как видно из определения в сноске к англоязычной версии термина, сама идея позаимствована из управления проектами и трансформирована в контексте тестирования в следующую: существует большинство пользователей, которые чаще всего используют некое подмножество функций приложения (см. рисунок 2.3.g). Именно эти функции и нужно проверить, как только мы убедились, что приложение «в принципе работает» (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).

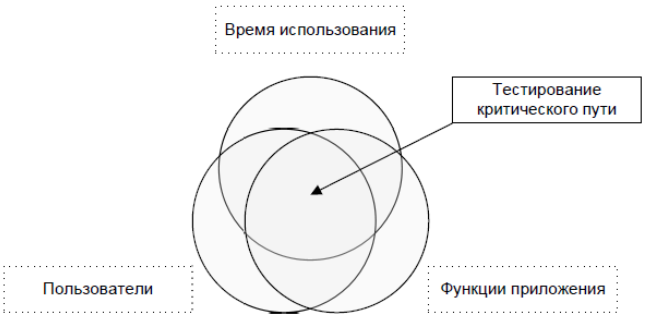


Рисунок 2.3.g — Суть тестирования критического пути

25. Что такое extended test?

Расширенное тестирование (extended test145) 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).

26. Что такое динамическое тестирование?

Динамическое тестирование (dynamic testing118) — тестирование с запуском кода на исполнение. Запускаться на исполнение может как код всего приложения целиком (системное тестирование{78}), так и код нескольких взаимосвязанных частей (интеграционное тестирование{77}), отдельных частей (модульное или компонентное тестирование{77}) и даже отдельные участки кода. Основная идея этого вида тестирования состоит в том, что проверяется реальное поведение (части) приложения.

27. Что такое статическое тестирование?

Статическое тестирование (static testing116) — тестирование без запуска кода на исполнение. В рамках этого подхода тестированию могут подвергаться:

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

. Графические прототипы (например, эскизы пользовательского интерфейса).

. Код приложения (что часто выполняется самими программистами в рамках аудита кода (code review117), являющегося специфической вариацией взаимного просмотра{50} в применении к исходному коду). Код приложения также можно проверять с использованием техник тестирования на основе структур кода{98}.

. Параметры (настройки) среды исполнения приложения.

. Подготовленные тестовые данные.

28. Что такое граничные значения?

Граничное условие (border condition, boundary condition350) — значение, находящееся на границе классов эквивалентности.

29. Что такое классы эквивалентности?

Класс эквивалентности (equivalence class349) — набор данных, обрабатываемых одинаковым образом и приводящих к одинаковому результату.

Класс эквивалентности (equivalence class) — одно или несколько значений ввода, к которым программное обеспечение применяет одинаковую логику.

Техника анализа классов эквивалентности

это техника, при которой мы разделяем функционал (часто диапазон возможных вводимых значений) на группы эквивалентных по своему влиянию на систему значений.

Тестирование на основе классов эквивалентности (equivalence partitioning229) — техника тестирования, направленная на сокращение количества разрабатываемых и выполняемых тест-кейсов при сохранении достаточного тестового покрытия. Суть техники состоит в выявлении наборов эквивалентных тест-кейсов (каждый из которых проверяет одно и то же поведение приложения) и выборе из таких наборов небольшого подмножества тест-кейсов, с наибольшей вероятностью обнаруживающих проблему.

30. Что такое позитивные тест?

Позитивное тестирование (positive testing146) 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.

31. Что такое негативный тест?

Негативное тестирование (negative testing147, invalid testing148) — 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.

32. Что такое ожидаемый результат?

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…».

33. Что такое отчет о результатах тестирования?

Отчёт о результатах тестирования (test progress report[342], test summary report[343]) — 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.
Краткое описание (summary). 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** (test team). Список участников проектной команды, задействованных в обеспечении качества, с указанием их должностей и ролей в подотчётный период.
• **Описание процесса тестирования** (testing process description). Последовательное описание того, какие работы были выполнены за подотчётный период.
• **Расписание** (timetable). Детализированное расписание работы команды тестировщиков и/или личные расписания участников команды.
• **Статистика по новым дефектам** (new defects statistics). Таблица, в которой представлены данные по обнаруженным за подотчётный период дефектам (с классификацией по стадии жизненного цикла и важности).
• **Список новых дефектов** (new defects list). Список обнаруженных за подотчётный период дефектов с их краткими описаниями и важностью.
• **Статистика по всем дефектам** (overall defects statistics). 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 (recommendations). 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** (appendixes). Фактические данные (как правило, значения метрик и графическое представление их изменения во времени).

34. Что такое автоматизированное тестирование, его особенности?

*Таблица 2.3.b — **Преимущества и недостатки автоматизированного тестирования***

Преимущества	Недостатки
• Скорость выполнения тест-кейсов может в разы и на порядки превосходить возможности человека. • Отсутствие влияния человеческого фактора в процессе выполнения тест-кейсов (усталости, невнимательности и т.д.). • Минимизация затрат при многократном выполнении тест-кейсов (участие человека здесь требуется лишь эпизодически). • Способность средств автоматизации выполнить тест-кейсы, в принципе непосильные для человека в силу своей сложности, скорости или иных факторов. • Способность средств автоматизации собирать, сохранять, анализировать, агрегировать и представлять в удобной для восприятия человеком форме колоссальные объёмы данных. • Способность средств автоматизации выполнять низкоуровневые действия с приложением, операционной системой, каналами передачи данных и т.д.	• Необходим высококвалифицированный персонал в силу того факта, что автоматизация — это «проект внутри проекта» (со своими требованиями, планами, кодом и т.д.) • Высокие затраты на сложные средства автоматизации, разработку и сопровождение кода тест-кейсов. • Автоматизация требует более тщательного планирования и управления рисками, т.к. в противном случае проекту может быть нанесён серьёзный ущерб. • Средств автоматизации крайне много, что усложняет проблему выбора того или иного средства и может повлечь за собой финансовые затраты (и риски), необходимость обучения персонала (или поиска специалистов). • 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.

Если же выразить все преимущества и недостатки автоматизации тестирования одной фразой, то получается, что автоматизация позволяет ощутимо увеличить тестовое покрытие (test coverage[127]), но при этом столь же ощутимо увеличивает риски.

Автоматизированное тестирование (automated testing, test automation126) — 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.

35. Что такое тестирование безопасности, его особенности?

Тестирование безопасности (security testing185) — тестирование, направленное на проверку способности приложения противостоять злонамеренным попыткам получения доступа к данным или функциям, права на доступ к которым у злоумышленника нет.

36. Что такое модульное тестирование, его особенности?

Модульное (компонентное) тестирование (unit testing, module testing, component testing128) 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{75}, значительно упрощающих и ускоряющих разработку соответствующих тест-кейсов.

37. Что такое инсталляционное тестирование, его особенности?

Инсталляционное тестирование (installation testing, installability testing165) — тестирование, направленное на выявление дефектов, влияющих на протекание стадии инсталляции (уста-

новки) приложения. В общем случае такое тестирование проверяет множество сценариев и аспектов работы инсталлятора в таких ситуациях, как:
. новая среда исполнения, в которой приложение ранее не было инсталлировано;
. обновление существующей версии («апгрейд»);
. изменение текущей версии на более старую («даунгрейд»);
. повторная установка приложения с целью устранения возникших проблем («переинсталляция»);
. повторный запуск инсталляции после ошибки, приведшей к невозможности продолжения инсталляции;
. удаление приложения;
. установка нового приложения из семейства приложений;
. автоматическая инсталляция без участия пользователя.

38. Что такое юзабилити тестирование, его особенности?

Тестирование удобства использования (usability[175] testing) — тестирование, направленное на исследование того, насколько конечному пользователю понятно, как работать с продуктом (understandability[176], learnability[177], operability[178]), а также на то, насколько ему нравится использовать продукт (attractiveness[179]). 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, проведением маркетинговых исследований и т.д.