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

**Качество (quality)** – показатель степени соответствия продукта его требованиям. Качество продукта определяется качеством процесса его разработки.

**Жизненный цикл программного обеспечения (ЖЦПО)** – это период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

**«Исчерпывающее тестирование (exhaustive testing)»** – проверки всех возможных путей выполнения кода со всеми возможными входными данными.

**Требование (requirement)** – описание того, какие функции и с соблюдением каких условий должно выполнять приложение в процессе решения полезной для пользователя задачи.

**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.

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

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

**Тест-сценарий (test scenario, test procedure specification, test script)** – документ, описывающий последовательность действий по выполнению теста (также известен как «тест-скрипт»)

**Набор тест-кейсов (test case suite, test suite, test set)** – совокупность тест-кейсов, выбранных с некоторой общей целью или по некоторому общему признаку. Иногда в такой совокупности результаты завершения одного тест-кейса становятся входным состоянием приложения для следующего тест-кейса

**Дефект** – расхождение ожидаемого и фактического результата

**Ожидаемый результат** – поведение системы, описанное в требованиях.

**Фактический результат** – поведение системы, наблюдаемое в процессе тестирования.

**Ошибка** (error, mistake) – действие человека, приводящее к некорректным результатам

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

**Планирование** (planning) – непрерывный процесс принятия управленческих решений и методической организации усилий по их реализации с целью обеспечения качества некоторого процесса на протяжении длительного периода времени.

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

**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.

**Метрика** (metric) – числовая характеристика показателя качества. Может включать описание способов оценки и анализа результата.

**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

2. Расширение рынка программного обеспечения тестирование приобрело повышенную важность в силу множества **объективных и субъективных причин**, среди которых особенно выделяются:

1)**Бизнес – причины**: намного проще конкурировать на рынке программного обеспечения той фирме, которая выпускает качественное программное обеспечение.

2)**Предпочтения пользователей**: от качества программного обеспечения зависят такие важные для пользователя факторы как безопасность личных данных, денежных операций и т.д.

3)**Объективные факторы**: наконец, есть области, в которых от качества программного обеспечения зависит жизнь людей (иногда – сотен, тысяч и даже миллионов).

3. Определение характеристик качества:

1) **Функциональные возможности** – способность программного средства обеспечивать решение задач, удовлетворяющих сформулированные потребности заказчиков и пользователей при применении комплекса программ в заданных условиях.

2)**Функциональная пригодность** – набор атрибутов, определяющих назначение, номенклатуру, основные, необходимые и достаточные функции программного средства, соответствующие техническому заданию и спецификациям требований заказчика или потенциального пользователя.

3)**Правильность (корректность)** – способность программного средства обеспечивать правильные или приемлемые для пользователя результаты и внешние эффекты.

4)**Способность к взаимодействию** – свойство программных средств и их компонентов взаимодействовать с одной или большим числом компонентов внутренней и внешней среды.

5)**Защищенность** – способность компонентов программного средства защищать программы и информацию от любых негативных воздействий.

6)**Надежность** – обеспечение комплексом программ достаточно низкой вероятности отказа в процессе функционирования программного средства в реальном времени.

7) **Эффективность** – свойства программного средства, обеспечивающие требуемую производительность решения функциональных задач, с учетом количества используемых вычислительных ресурсов в установленных условиях.

8) **Практичность (применимость)** – свойства программного средства, обусловливающие сложность его понимания, изучения и использования, а также привлекательность для квалифицированных пользователей при применении в указанных условиях.

9) **Сопровождаемость** – приспособленность программного средства к модификации и изменению конфигурации и функций.

10)**Мобильность** – подготовленность программного средства к переносу из одной аппаратно-операционной среды в другую.

4. Жизненный цикл ПО делится на десять фаз:

* выработка требований;
* разработка спецификаций;
* общее проектирование;
* проектирование архитектуры;
* детальное проектирование;
* реализация и отладка (coding and debug);
* интеграция;
* сертификация;
* внедрение (deployment);
* сопровождение (support).

5. **Упрощенная 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 – 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).

6. Водопадная модель (waterfall model) 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). 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.

V-образная модель (V-model) является логическим

развитием водопадной. Можно заметить (рисунок 2.2), 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.

Спиральная модель представляет собой процесс разработки программного обеспечения, сочетающий в себе как проектирование, так и постадийное прототипирование с целью сочетания преимуществ восходящей и нисходящей концепции.

Итерационная инкрементальная модель

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 2.3). Итогом итерации является приращение

(инкремент) функциональности продукта, выраженное в

промежуточном билде (build).

Гибкая модель (agile model) представляет собой

совокупность различных подходов к разработке ПО и базируется

на т.н. «agile-манифесте». 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.

7. Продуктивная документация (product documentation, development documentation) используется проектной командой во время разработки и поддержки продукта. Она включает:

– план проект и в том числе тестовый план;

– требования к программному продукту и функциональные спецификации;

– архитектуру и дизайн;

– тест-кейсы и наборы тест-кейсов (test cases, test suites);

– технические спецификации, такие как схемы баз данных, описание алгоритмов, интерфейсов и т.д.

8. Основные техники сбора и выявления требований

* Интервью
* Работа с фокусными группами
* Анкетирование.
* Семинары и мозговой штурм.
* Наблюдение.
* Прототипирование.
* Анализ документов.
* Моделирование процессов и взаимодействий.
* Самостоятельное описание.

9. Критерии качества

* Завершённость (completeness)
* Атомарность, единичность
* Непротиворечивость, последовательность (consistency).
* Недвусмысленность (unambiguousness, clearness).
* Выполнимость (feasibility).
* Обязательность, нужность (obligatoriness)
* Актуальность (up-to-date).
* Прослеживаемость (traceability).
* Модифицируемость (modifiability).
* Проранжированность по важности, стабильности, срочности (ranked for importance, stability, priority).
* Корректность (correctness)
* Проверяемость (verifiability).

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

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

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

– структурированным (многоуровневым) списком (вне зависимости от учёта последовательности пунктов), что позволяет отразить иерархию идей.

11. Для того чтобы чек-лист был действительно полезным инструментом, он должен обладать рядом важных свойств:

1)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.

2) 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, потом идеи простых негативных тест?кейсов, потом постепенно повышать сложность тест-кейсов, но не стоит писать эти идеи вперемешку).

3)Полнота и не избыточность. 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.

12. Жизненный цикл тест-кейса:создан, заплонирован, выполняется, пропущен, провален, пройден успешно, заблокирован,

закрыт, требует дорабатки



13.Преимущества свободных наборов:

1. Тест-кейсы можно выполнять в любом удобном порядке, а также создавать «наборы внутри наборов».

2. Если какой-то тест-кейс завершился ошибкой, это не повлияет на возможность выполнения других тест-кейсов.

14.Преимущества последовательных наборов:

1. Каждый следующий в наборе тест-кейс в качестве входного состояния приложения получает результат работы предыдущего тест-кейса, что позволяет сильно сократить количество шагов в отдельных тест-кейсах.

2. Длинные последовательности действий куда лучше имитируют работу реальных пользователей, чем отдельные «точечные» воздействия на приложение.

15. Преимущества:

– сценарии показывают реальные и понятные примеры использования продукта (в отличие от обширных чек-листов, где смысл отдельных пунктов может теряться);

– сценарии понятны конечным пользователям и хорошо подходят для обсуждения и совместного улучшения;

– сценарии и их части легче оценивать с точки зрения важности, чем отдельные пункты (особенно низкоуровневых) требований;

– сценарии отлично показывают недоработки в требованиях (если становится непонятно, что делать в том или ином пункте сценария, – с требованиями явно что-то не то).

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

1. Предоставить информацию о проблеме – уведомить проектную команду и иных заинтересованных лиц о наличии проблемы, описать суть проблемы.

2. Приоритизировать проблему – определить степень опасности проблемы для проекта и желаемые сроки её устранения.

3. 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

17. Стадии ЖЦ отчёта о дефекте: Обнаружен, назначен, исправлен, проверен, закрыт, открыт заново, рекомендован к отклонению, отклонён,

отложен



18. Атрибуты (поля) отчёта о дефекте:

1.Идентификатор (identifier)

2.Краткое описание (summary)

3.Подробное описание (description)

4.Шаги по воспроизведению

5.Воспроизводимость (reproducibility)

6.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.

7. 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 нескольких 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

19. К высокоуровневым задачам планирования относятся:

– снижение неопределённости;

– повышение эффективности;

– улучшение понимания целей;

– создание основы для управления процессами.

20. К низкоуровневым задачам планирования в тестировании (в частности задачам тест-плана) относятся:

– оценка объёма и сложности работ;

– определение необходимых ресурсов и источников их получения;

– определение расписания, сроков и ключевых точек;

– оценка рисков и подготовка превентивных контрмер;

– распределение обязанностей и ответственности;

– согласование работ по тестированию с деятельностью участников проектной команды, занимающихся другими задачами.

21. К высокоуровневым задачам отчётности относятся:

– сбор, агрегация и предоставление в удобной для восприятия форме объективной информации о результатах работы;

– формирование оценки текущего статуса и прогресса (в сравнении с планом);

– обозначение существующих и возможных проблем (если такие есть);

– формирование прогноза развития ситуации и фиксация рекомендаций по устранению проблем и повышению эффективности работы

22. К низкоуровневым задачам отчётности в тестировании (в частности к отчету о результатах тестирования) 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.

23. 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;

– руководителю команды тестировщиков («тест-лиду») – как способ структурировать собственные мысли и собрать необходимый материал для обращения к менеджеру проекта по насущным вопросам, если в этом есть необходимость;

– заказчику – как наиболее объективный источник информации о том, что происходит на проекте, за который он платит свои деньги.

24. В общем случае отчёт о результатах тестирования включает следующие разделы:

* 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).
* 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.
* Список новых дефектов (new defects list). Список обнаруженных за подотчётный период дефектов с их краткими описаниями и важностью.
* 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)