**МЕТОДОЛОГИИ РАЗРАБОТКИ ПО**

**Методология** — это система принципов, а также совокупность идей, понятий, методов, способов и средств, определяющих стиль разработки программного обеспечения.

**1. «Waterfall Model» (каскадная модель или «водопад»)**

Одна из самых старых, подразумевает последовательное прохождение стадий, каждая из которых должна завершиться полностью до начала следующей.

Нет возможности сделать шаг назад, тестирование начинается только после того, как разработка завершена или почти завершена. Продукты, разработанные по данной модели без обоснованного ее выбора, могут иметь недочеты (список требований нельзя скорректировать в любой момент), о которых становится известно лишь в конце из-за строгой последовательности действий. Стоимость внесения изменений высока, так как для ее инициализации приходится ждать завершения всего проекта.

**2. «V-Model»**

Унаследовала структуру «шаг за шагом» от каскадной модели. V-образная модель применима к системам, которым особенно важно бесперебойное функционирование. Например, прикладные программы в клиниках для наблюдения за пациентами, интегрированное ПО для механизмов управления аварийными подушками безопасности в транспортных средствах и так далее. Особенностью модели можно считать то, что она направлена на тщательную проверку и [тестирование продукта](http://www.edsd.ru/ru/uslugi/testirovanie_po), находящегося уже на первоначальных стадиях проектирования. Стадия тестирования проводится одновременно с соответствующей стадией разработки, например, во время кодирования пишутся модульные тесты.

**3. «Incremental Model» (инкрементная модель)**

В инкрементной модели полные требования к системе делятся на различные сборки. Терминология часто используется для описания поэтапной сборки ПО. Имеют место несколько циклов разработки, и вместе они составляют жизненный цикл «мульти-водопад». Цикл разделен на более мелкие легко создаваемые модули. Каждый модуль проходит через фазы определения требований, проектирования, кодирования, внедрения и тестирования. Процедура разработки по инкрементной модели предполагает выпуск на первом большом этапе продукта в базовой функциональности, а затем уже последовательное добавление новых функций, так называемых «инкрементов». Процесс продолжается до тех пор, пока не будет создана полная система.

**4. «RAD Model» (rapid application development model или быстрая разработка приложений)**

RAD-модель — разновидность инкрементной модели. В RAD-модели компоненты или функции разрабатываются несколькими высококвалифицированными командами параллельно, будто несколько мини-проектов. Временные рамки одного цикла жестко ограничены. Созданные модули затем интегрируются в один рабочий прототип. Синергия позволяет очень быстро предоставить клиенту для обозрения что-то рабочее с целью получения обратной связи и внесения изменений.

**5. «Agile Model» (гибкая методология разработки)**

В «гибкой» методологии разработки после каждой итерации заказчик может наблюдать результат и понимать, удовлетворяет он его или нет. Это одно из преимуществ гибкой модели. К ее недостаткам относят то, что из-за отсутствия конкретных формулировок результатов сложно оценить трудозатраты и стоимость, требуемые на разработку. Экстремальное программирование (XP) является одним из наиболее известных применений гибкой модели на практике.

**6. «Iterative Model» (итеративная или итерационная модель)**

Итерационная модель жизненного цикла не требует для начала полной спецификации требований. Вместо этого, создание начинается с реализации части функционала, становящейся базой для определения дальнейших требований. Этот процесс повторяется. Версия может быть неидеальна, главное, чтобы она работала. Понимая конечную цель, мы стремимся к ней так, чтобы каждый шаг был результативен, а каждая версия — работоспособна.

**7. «Spiral Model» (спиральная модель)**

«Спиральная модель» похожа на инкрементную, но с акцентом на анализ рисков. Она хорошо работает для решения критически важных бизнес-задач, когда неудача несовместима с деятельностью компании, в условиях выпуска новых продуктовых линеек, при необходимости научных исследований и практической апробации.

Эта модель не подойдет для малых проектов, она резонна для сложных и дорогих, например, таких, как разработка системы документооборота для банка, когда каждый следующий шаг требует большего анализа для оценки последствий, чем программирование.

**Качество программного обеспечения** - это совокупность характеристик ПО, относящихся к его способности удовлетворять установленные и предполагаемые потребности.

Программы должны обладать следующими характеристиками :

- **Функциональность**. Работа функций приложения, их пригодность к использованию;

- **Надежность**. Отсутствие дефектов. Устойчивость к отказам, работа под нагрузкой и способность к восстановлению;

- **Практичность**. Удобство пользователя, не забываем, что продукт мы делаем для наших клиентов, они должны с удовольствием работать с нашим приложением;

- **Эффективность**. Скорость работы приложения;

- **Сопровождаемость**. Возможность внесения изменений и дальнейшее развитие системы;

- **Мобильность**. Совместимость, адаптируемость приложения к различным конфигурациям

**(QA) Обеспечение качества** – это совокупность мероприятий, охватывающие все технологические этапы разработки, выпуска и эксплуатации программного обеспечения информационных систем, предпринимаемых на разных стадиях жизненного цикла ПО, для обеспечения качества выпускаемого продукта.

**(QC) Контроль качества** - это совокупность действий, проводимых над продуктом в процессе разработки, для получения информации о его актуальном состоянии в разрезах: "готовность продукта к выпуску", "соответствие зафиксированным требованиям", "соответствие заявленному уровню качества продукта"

**Верификация** - это процесс оценки системы или её компонентов с целью определения удовлетворяют ли результаты текущего этапа разработки условиям, сформированным в начале этого этапа. Т.е. выполняются ли наши цели, сроки, задачи по разработке проекта, определенные в начале текущей фазы.

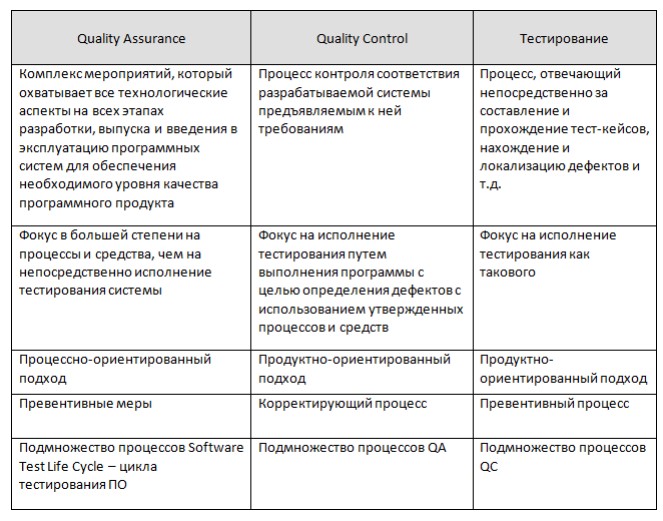
**Валидация** - это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе.

**Баг** – ошибка в программе или системе, которая выдает неожиданный или неправильный результат.

**Тестирование** - это проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. Есть и ещё одно определение.

**Тестирование** - это одна из техник контроля качества (QC), включающая в себя активности по планированию работ (Test Management), проектированию тестов (Test Designe), выполнения тестирования(Test execution) и анализа полученных результатов(Test analysis).

|  |  |  |
| --- | --- | --- |
| **QAE занимается:**  - тестированием продукта  - аудитом качества продукта  - улучшением продукта  - аудитом процессов разработки  - улучшением процессов. | **QCE выполняет задачи по:**  - тестированию  - аудиту качества продукта  - улучшению продукта  - аудиту процессов разработки (реже) | **Tester:**  - тестирует  - проводит аудит качества продукта (реже) |

****

**Цели QA:**

- Обеспечить беспроблемную приемку проекта заказчиком.

- Гарантировать успешность продукта

- Гарантировать удобство и интуитивность пользования продуктом, его стабильность, производительность и расширяемость

- Минимизировать затраты на процессы QC и разработку

- Минимизировать риски проекта.

**Фазы процесса разработки программного обеспечения**

- **Анализ требований** (requirements analysis). На этом этапе мы анализируем, что хочет заказчик, чтобы на этапе проектирования иметь чёткое представление о будущем продукте.

- **Проектирование** (design). На этом этапе команда разработчиков должна написать, как будем реализовывать поставленную заказчиком задачу, что будем делать, какие методы будем использовать, на каком язык программирования будем писать программу.

- **Реализация** (coding). После того как все требования зафиксированы и решения приняты на этапе (coding) программисты пишут код.

- **Тестирование и отладка** (testing and debug). После того как часть кода была написана, тестировщики начинают тестировать написанную программистами программу, сопоставляют требования, заводят баги, отправляют их назад на исправление (fixing) программистам. Программисты исправляют баги, тестировщики опять проверяют.

- **Внедрение** (deployment). После того как отладка завершена мы можем выпустить часть этой программы, т.е. внедрить программу, если конечно этот продукт делает, то что он должен делать.

- **Сопровождение** (support). На этом этапе пользователи могут обращаться в службу поддержки сайта или программы с просьбами, например: добавить новые фичи (функциональности), либо же исправить то, что по их мнению должно работать не так и т.д. Команда на такие обращения реагирует и сопровождает ПО.

**Важность требований заключается в том, что они:**

- 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.

**Требования к программному обеспечению** - совокупность утверждений относительно атрибутов, свойств или качеств программной системы, подлежащей реализации. Создаются в процессе разработки требований к программному обеспечению, в результате анализа требований.

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

**Виды требований по уровням.**

- Бизнес-требования - определяют назначение ПО, описываются в документе о видении (vision) и границах проекта.

- Пользовательские требования - определяют набор пользовательских задач, которые должна решать программа, а также способы (сценарии) их решения в системе. Пользовательские требования могут выражаться в виде фраз утверждений, в виде способов применения (use case), пользовательских историй (user story), сценариев взаимодействия (scenario).

- Функциональные требования — охватывают предполагаемое поведение системы, определяя действия, которые система способна выполнять. Описывается в системной спецификации.

**Виды требований по характеру.**

Функциональный характер— требования к поведению системы

- Бизнес-требования

- Пользовательские требования

- Функциональные требования

Нефункциональный характер— требования к характеру поведения системы

- Бизнес-правила — определяют ограничения, проистекающие из предметной области и свойств автоматизируемого объекта (предприятия)

- Системные требования и ограничения — определения элементарных операций, которые должна иметь система, а также различных условий, которым она может удовлетворять. К системным требованиям и ограничениям относятся:

- Ограничения на программные интерфейсы, в том числе к внешним системам

- Требования к атрибутам качества

- Требования к применяемому оборудованию и ПО

- Требования к документированию

- Требования к дизайну и удобству использования

- Требования к безопасности и надёжности

- Требования к показателям назначения (производительность, устойчивость к сбоям и т. п.)

- Требования к эксплуатации и персоналу

- Прочие требования и ограничения (внешние воздействия, мобильность, автономность и т. п.).

**Источники и пути 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 фигу-рами 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 (как правило, представляющих собой целевую аудиторию, и/или обладающих важной для проекта информацией, и/или уполномоченных принимать важные для проекта решения).

**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» (например: «договор на закупку формируется отделом закупок, визируется бухгалтерией и юридическим отделом…»), так и к «техническим процессам и взаимодействиям» (например: «платёжное поручение генерируется модулем “Бухгалтерия”, шифруется модулем “Безопасность” и передаётся на сохранение в модуль “Хранилище”»). 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.

**Характеристики качественных требований**

**Единичность** - Требование описывает одну и только одну вещь.

**Завершенность** - Требование полностью определено в одном месте и вся необходимая информация присутствует.

**Последовательность** - Требование не противоречит другим требованиям и полностью соответствует внешней документации.

**Атомарность** - Требование «атомарно». То есть оно не может быть разбито на ряд более детальных требований без потери завершенности

**Отслеживаемость** - Требование полностью или частично соответствует деловым нуждам как заявлено заинтересованными лицами и документировано

**Актуальность** - Требование не стало устаревшим с течением времени.

**Выполнимость** - Требование может быть реализовано в пределах проекта

**Недвусмысленность** - Требование кратко определено без обращения к техническому жаргону, акронимам и другим скрытым формулировкам. Оно выражает объективные факты, не субъективные мнения. Возможна одна и только одна интерпретация. Определение не содержит нечётких фраз. Использование отрицательных утверждений и составных утверждений запрещено

**Обязательность** - Требование представляет определенную заинтересованным лицом характеристику, отсутствие которой приведёт к неполноценности решения, которая не может быть проигнорирована. Необязательное требование — противоречие самому понятию требования.

**Проверяемость** - Реализованность требования может быть определена через один из четырёх возможных методов: осмотр, демонстрация, тест или анализ

**Цели тестирования (Testing goals).**

1. Показать разработчику и клиенту, что программное обеспечение отвечает заявленным требованиям.

2. Найти ситуации, когда программное обеспечение ведет себя ошибочно, нежелательно или не соответствует спецификации.

3. Создать предпосылки для предупреждения возникновения ошибок в программном обеспечении.

**Основные принципы тестирования**

Исчерпывающее тестирование невозможно ни одним из тестировщиков

Скопление багов

Парадокс пестицида

Эффективность раннего тестирования

Тестирование зависит от продукта

Тестирование показывает наличие багов в продукте, но никак не их отсутствие

Продукт хорошо протестирован и вроде бы нет очевидных багов, то это хороший продукт

**Цикл тестирования ПО состоит из четырех этапов:**

1. Изучение и анализ предмета тестирования.

- Какие функциональности предстоит протестировать?

- Как эти функциональности работают?

2. Планирование тестирования.

- Как тестировать продукт?

3. Исполнение тестирования.

- это практический поиск багов в созданной программе или на сайте с использованием тест-кейсов и чек листов написанных ранее.

4. Анализ прошедшего тестирования и написание отчета о проведенном тестировании

**Стратегия тестирования**

На этапе стратегии предопределяются вся дальнейшая политика, все дальнейшие действия, связанные с тестированием. Определяются стадии тестирования и типы тестов. Оформляется и разрабатывается тест план, определяется состав участников и их роли.

- какие инструментальные средства и методы будут применяться;

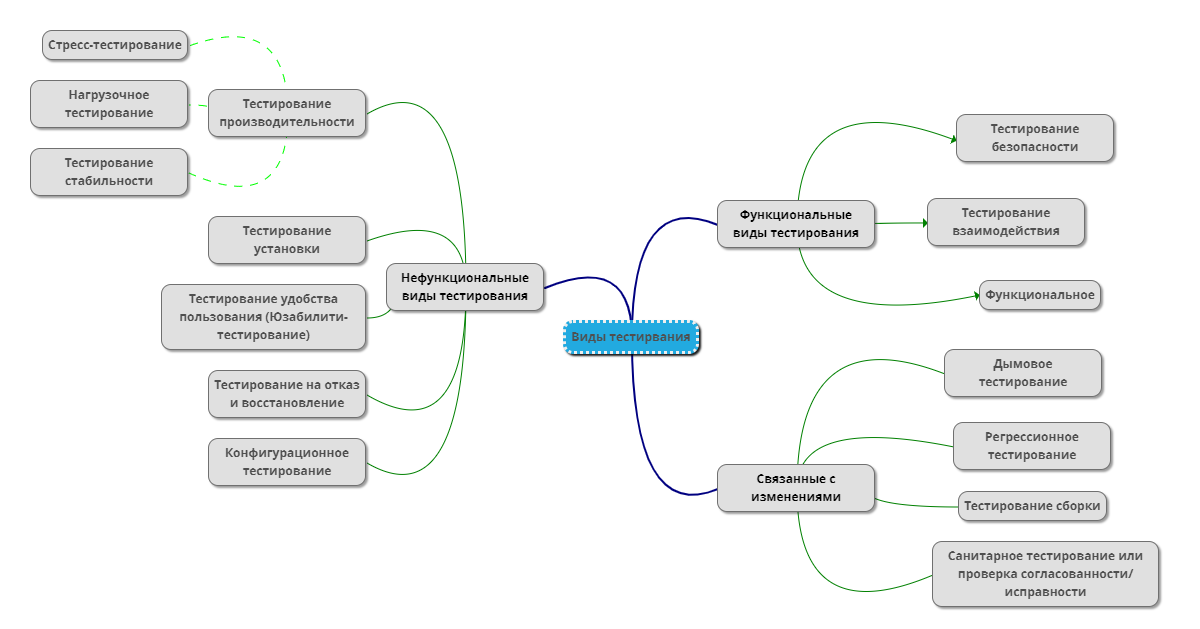
- критерии завершения итераций;

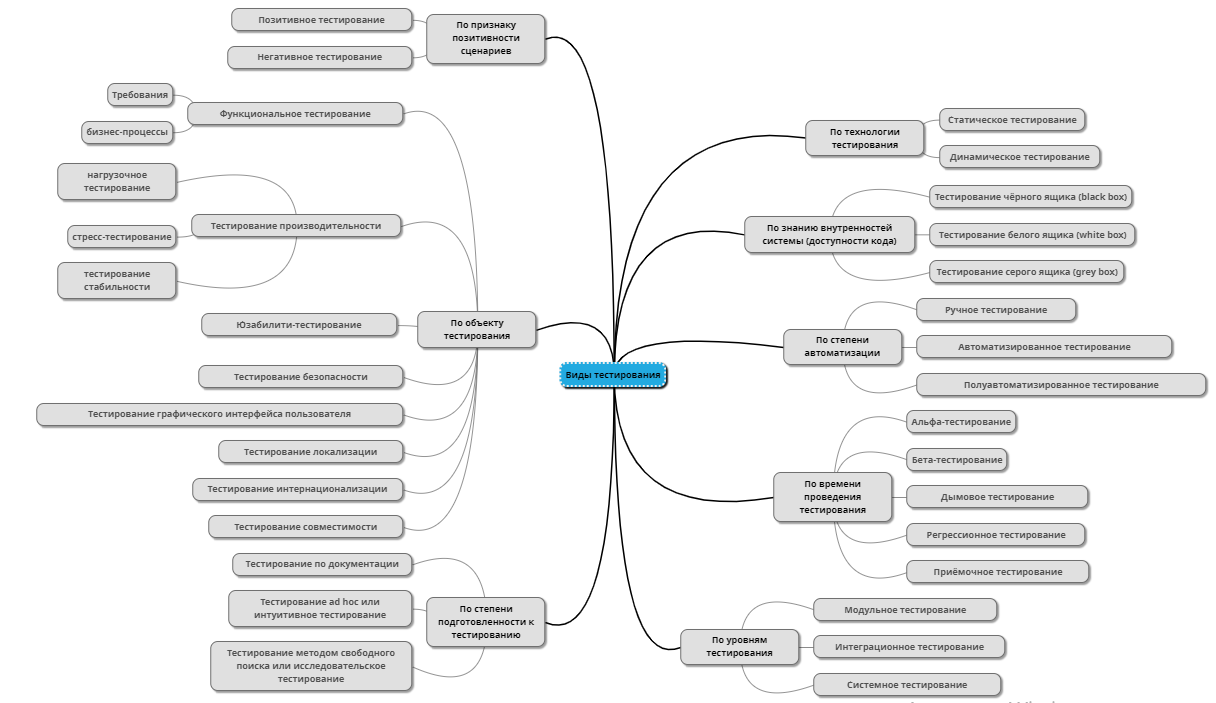
- процентную установку области охвата кода (то есть, что считать полностью протестированным приложением? 100% кода, или 95%);

- типы отчетов и способы автоматизированного их хранения;

- виды применяемых тестов: «черный ящик» или «белый ящик», либо их комбинации на различных стадиях

**Виды тестирования**





|  |  |
| --- | --- |
| **По объекту тестирования**  - **Функциональное тестирование** (functional testing). Проверка функций и характеристик разрабатываемого ПО на основе проектной документации;  - **Тестирование производительности** (performance testing). Проверка работоспособности системы под нагрузкой. Может служить для проверки и подтверждения других атрибутов качества системы, таких как масштабируемость, надежность и потребление ресурсов. Выделяются следующие подвиды: нагрузочное тестирование (load testing), стресс-тестирование (stress testing), тестирование стабильности (stability testing);  - **Юзабилити-тестирование** (usability testing ). Цель данного вида тестирования заключается в определении степени удобства и практичности пользовательского интерфейса;  - **Тестирование безопасности** (security testing). Проверка надежности системы от возможных рисков и угроз (потеря, конфиденциальность, целостность и доступность данных);  - **Тестирование графического интерфейса пользователя** (GUI) - это тип тестирования программного обеспечения, который проверяет графический интерфейс пользователя тестируемого приложения;  - **Тестирование локализации** (localization testing). Корректность работы отдельных компонентов системы (форматы дат, единицы измерения, не меняется часовой пояс при переезде и т.д.);  - **Тестирование интернационализации** (internationalization). Позволяет убедиться в поддержке культурных особенностей других стран (главным образом, в языковой поддержке).  - **Тестирование совместимости** (compatibility testing) Проверка возможности приложения взаимодействовать с различными программными продуктами, операционными системами и окружением. | **По времени проведения тестирования**  - **Альфа-тестирование** (alpha testing). Внутреннее тестирование (имитация реальной работы с системой штатными разработчиками).  - **Бета-тестирование** (beta testing). Внешнее пробное использование с привлечением отобранных пользователей.  - **Дымовое тестирование** (smoke testing). Минимальный набор тестов на явные ошибки;  - **Тестирование новой функциональности** (new feature testing). Тестируется новые функции системы;  - **Регрессионное тестирование** (regression testing). Проверка изменений сделанных в системе для подтверждения того факта, что существующая ранее функциональность работает как и прежде;  - **Приемочное тестирование** (acceptance testing).  Целью приемочного тестирования является оценка готовности системы для его выпуска на рынок или передачи клиенту. Может включать в себя альфа-тестирование (alpha testing) и бета-тестирование (beta testing). |
| **По технологии тестирования**  **Статическое тестирование** – это единственный способ тестирования без запуска программного кода приложения. Статическим тестированием пользуются для верификации практически любого артефакта разработки: программного кода компонент, требований, системных спецификаций, функциональных спецификаций, документов и т.д. Статические методы тестирования – один из наиболее эффективных способов обнаружения дефектов на ранних стадиях разработки ПО.  - вычитка документации, документации по дизайну, спецификации, требования и.т.д.  - вычитка кода (поверхностная проверка кода, архитектура кода)  - автоматизированное (проверка требований и написание скриптов)  **Динамическое тестирование** – процесс тестирования, производимый над работающей системой или подсистемой. Оно не может быть осуществлено без запуска программного кода приложения. Если быть более точным, динамическое тестирование состоит из:  1. Запуска системы или подсистемы  2. Вызова необходимых функциональных элементов или модулей  3. Сравнения через графический интерфейс пользователя поведения системы с ожидаемым результатом поведения. |
| **По степени подготовленности к тестированию**  -**Тестирование по документации** (formal testing). Тестирование приложения проводится по заранее подготовленным данным (тест-кейсы, чек-листы, читлисты, спецификация и т.д.)  - **Тестирование ad hoc или интуитивное тестирование** (ad hoc testing) — тестирование проводится при полном отсутствии документации, без плана и цели;  - **Тестирование методом свободного поиска или исследовательское тестирование** (exploratory testing) — предполагается наличие минимально необходимой для тестирования документации, либо тестирование без документации. |

**Вечный круг тестирования**

|  |  |
| --- | --- |
| **Понять логику.**  Когда мы получаем новый продукт, первая наша цель это понят логику продукта (для чего этот сайт или для чего это приложение) и разобраться в логике, во всех функциях, почему эти функции работают так, а не иначе. | **Найти несоответствия.**  На этом этапе мы открываем сайт или мобильное приложение (т.е. тестируемый продукт) и начинаем сравнивать с тем что мы ожидаем от продукта, после того как  прочли техническую документацию и поняли логику. Вы замечаете эти несоответствия(дефекты) |
| **Перепроверить.**  Разработчик дефекты исправляет и перенаправляет тестировщикам. После чего мы их перепроверяем. | **Донести информацию**.  Далее оформляете баги и доносите их разработчикам, соответственно через багтрекинговую систему. |

**Всего принято выделять 7 этапов тестирования:**

1. Работа с требованиями. Знакомство с требованиями заказчика, что должен из себя представлять итоговый продукт, обсуждение.
2. Разработка стратегии тестирования. Оценка сроков тестирования, выявление среды тестирования, объединение всей информации, полученной при работе с требованиями.
3. Создание тестовой документации. Написание сценариев, которые позволят проверить функционал.
4. Тестирование прототипа. Тестирование основного функционала продукта, корректировка целей, добавление фичей.
5. Основное тестирование. Выполнение общей проверки продукта.
6. Стабилизация. На данном этапе происходит работа над устранением багов.
7. Эксплуатация. Проводится регресс-тестирование, устранение ошибок, которые нашел конечный пользователь.

**Тестовые артефакты**

**Спецификация программного обеспечения** (Software Specification) - законченное описание поведения программы, которую требуется разработать.

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

**Тестовый случай** (Test Case) - это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части. Другими словами – это сценарий, по которому мы будем, что-либо тестировать.

**Тестовый набор** (Test suite) - это набор тест кейсов объединённых по какой-либо роли или функциональности.

**Баг Репорты** (Bug Reports) - это документы, описывающие ситуацию или последовательность действий, приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.

**Чек лист** (контрольный список) - список параметров, который необходимо проверить

**Техники тест дизайна**

Тест дизайн – это этап процесса тестирования ПО, на котором проектируются и создаются тестовые случаи (тест кейсы), в соответствии с определёнными ранее критериями качества и целями тестирования.

**Эквивалентное разделение** (Equivalence Partitioning - EP), классы эквивалентности (equivalent classes-EC)

**Анализ граничных Значений** (Boundary Value Analysis – BVA)

**Причина/ Следствие (Cause/Effect - CE)** Простая проверка базовых действий и их результата.

**Предугадывание ошибки (Error Guessing - EG)** Используя свои знания о системе, QA-специалист может «предугадать», при каких входных условиях есть риск ошибок

**Попарное тестирование (Pairwise testing)** Суть этого метода, также известного как pairwise testing, в том, что каждое значение каждого проверяемого параметра должно быть протестировано на взаимодействие с каждым значением всех остальных параметров

**Таблица принятия решений** -описывает поведение системы при разных значениях нескольких входных данных. Этот метод хорош тем, что он показывает сразу все возможные сценарии в форме, понятной даже неспециалисту.

**Исчерпывающее тестирование (Exhaustive Testing - ET)** В пределах этой техники вы должны проверить все возможные комбинации входных значений, и в результате, это должно найти все проблемы

**Градация Серьезности дефекта (Severity)**

**S1 Блокирующая** (Blocker) Блокирующая ошибка, приводящая приложение в нерабочее состояние, в результате которого дальнейшая работа с тестируемой системой или ее ключевыми функциями становится невозможна. Решение проблемы необходимо для дальнейшего функционирования системы.

**S2 Критическая** (Critical) Критическая ошибка, неправильно работающая ключевая бизнес логика, дыра в системе безопасности, проблема, приведшая к временному падению сервера или приводящая в нерабочее состояние некоторую часть системы, без возможности решения проблемы, используя другие входные точки. Решение проблемы необходимо для дальнейшей работы с ключевыми функциями тестируемой системой.

**S3 Значительная** (Major) Значительная ошибка, часть основной бизнес логики работает некорректно. Ошибка не критична или есть возможность для работы с тестируемой функцией, используя другие входные точки.

**S4 Незначительная** (Minor) Незначительная ошибка, не нарушающая бизнес логику тестируемой части приложения, очевидная проблема пользовательского интерфейса.

**S5 Тривиальная** (Trivial) Тривиальная ошибка, не касающаяся бизнес логики приложения, плохо воспроизводимая проблема, малозаметная посредствам пользовательского интерфейса, проблема сторонних библиотек или сервисов, проблема, не оказывающая никакого влияния на общее качество продукта.

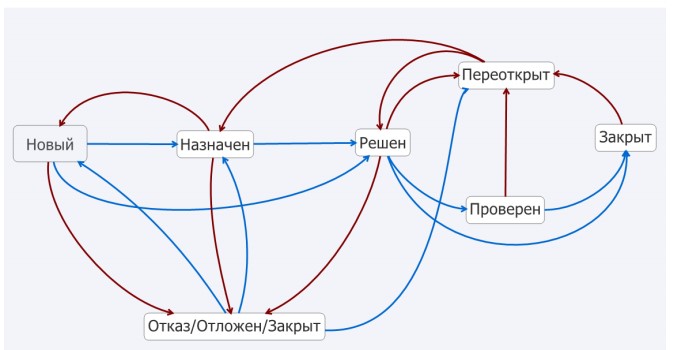
**Градация Приоритета дефекта (Priority)**

**P1 Высокий (High)** Ошибка должна быть исправлена как можно быстрее, т.к. ее наличие является критической для проекта.

**P2 Средний (Medium)** Ошибка должна быть исправлена, ее наличие не является критичной, но требует обязательного решения.

**P3 Низкий (Low)** Ошибка должна быть исправлена, ее наличие не является критичной, и не требует срочного решения.

**Жизненный цикл дефекта**



**Тест дизайн (Test Design)** – это этап процесса тестирования ПО, на котором проектируются и создаются тестовые случаи (Test Case), в соответствии с определёнными ранее критериями качества и целями тестирования.

**Тестовый случай (Test Case)** - это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части.

**Тестовый набор (Test Suite)** - это набор тестов, реализующих бизнес-задачу, выполняемую тестируемой системой. Тестовый набор включает в себя, кроме тестовых сценариев, ещё и тестовые данные и правила их генерации.

Выполняемое действие (Action) – Ожидаемый результат (Expected result) – Результат теста (Test result).

Каждый тест кейс имеет 3 основные составляющие:

- **PreConditions** (Предусловия) – список действий, которые приводят систему к состоянию пригодному для проведения основной проверки. Либо список условий, выполнение которых говорит о том, что система находится в пригодном для проведения основного теста состояния.

- **Test Case Description** (Описание тестового случая) – список действий, переводящих систему из одного состояния в другое, для получения результата, на основании которого можно сделать вывод о удовлетворении реализации, поставленным требованиям.

- **PostConditions** (Постусловия) – список действий, которые возвращают систему в первоначальное состояние.

**Модульное тестирование** (компонентное или юнит-тестирование) проводится для проверки соответствия программного продукта функциональным требованиям и определения багов в наименьших доступных частях приложения, которые могут быть протестированы по отдельности (модули программ, объекты, классы и т.д.)

Основная идея юнит-тестирования – это изолированная проверка каждого отдельного элемента путем запуска тестов в искусственной среде. Главная задача - это показать, что отдельные части приложения работают корректно.

**Интеграционное тестирование** – это совместное выполнение двух или более классов, пакетов, компонентов или подсистем, созданных несколькими программистами или группами.

**Системное тестирование**– это выполнение ПО в его окончательной конфигурации, интегрированного с другими программными и аппаратными системами.

Системное тестирование - это тестирование программного обеспечения, которое выполняется на полной, интегрированной системе, с целью проверки соответствия системы функциональным и прочим требованиям.

Системное тестирование выполняет основную задачу - определить разницу между функционалом готовой системы и функционалом, заявленным исходными требованиями.

Системное тестирование не требует знаний о внутреннем устройстве системы, так как относится к методам тестирования черного ящика.

**Регрессонным тестированием** называют повторное выполнение тестов, направленное на обнаружение дефектов в программе, уже прошедшей этот набор тестов.

**Фазы тестирования:**

Реализация тестирования делится на три этапа:

- Создание тестового набора (test suite) путем ручной разработки или автоматической генерации для конкретной среды тестирования (testing environment).

- Прогон программы на тестах, управляемый тестовым монитором (test monitor, test driver) с получением протокола тестирования (test log).

- Оценка результатов выполнения программы на наборе тестов с целью принятия решения о продолжении или остановке тестирования.

**Уровни интеграционного тестирования:**

- ***Компонентный интеграционный уровень*** (Component Integration testing)

Проверяется взаимодействие между компонентами системы после проведения компонентного тестирования.

- ***Системный интеграционный уровень*** (System Integration Testing)

Проверяется взаимодействие между разными системами после проведения системного тестирования.

**Подходы к интеграционному тестированию:**

- ***Снизу вверх*** (Bottom Up Integration)

Все низкоуровневые модули, процедуры или функции собираются воедино и затем тестируются. После чего собирается следующий уровень модулей для проведения интеграционного тестирования. Данный подход считается полезным, если все или практически все модули, разрабатываемого уровня, готовы. Также данный подход помогает определить по результатам тестирования уровень готовности приложения.

- ***Сверху вниз*** (Top Down Integration)

Вначале тестируются все высокоуровневые модули, и постепенно один за другим добавляются низкоуровневые. Все модули более низкого уровня симулируются заглушками с аналогичной функциональностью, затем по мере готовности они заменяются реальными активными компонентами. Таким образом, проводится тестирование сверху вниз.

- ***Большой взрыв*** ("Big Bang" Integration)

Все или практически все разработанные модули собираются вместе в виде законченной системы или ее основной части, и затем проводится интеграционное тестирование. Такой подход очень хорош для сохранения времени. Однако если тест кейсы и их результаты записаны не верно, то сам процесс интеграции сильно осложнится, что станет преградой для команды тестирования при достижении основной цели интеграционного тестирования.

**Этапы интеграционного тестирования:**

1. Создание тест плана.

2. Создание тестовых данных и тест кейсов.

3. Создание сценариев для запуска тест кейсов (при необходимости).

4. Выполнения тест кейсов, как только компоненты будут интегрированы.

5. Исправление багов и перепроверка кода.

6. Повторение цикла тестирования до тех пор, пока компоненты не будут успешно интегрированы.

**Основные задачи**

Главной задачей интеграционного тестирования является поиск ошибок, связанных с взаимодействием модулей системы или нескольких систем. В результате все смежные системы и модули одной системы должны работать согласованно.

Способы проведения интеграционного тестирования подбираются в зависимости от интеграционных решений.

**Критерии тестирования комплексных систем.**

Можно выделить требования к идеальному критерию тестирования:

- Критерий должен быть достаточным, т.е. показывать, когда некоторое конечное множество тестов достаточно для тестирования данной программы.

- Критерий должен быть полным, т.е. в случае ошибки должен существовать тест из множества тестов, удовлетворяющих критерию, который раскрывает ошибку.

- Критерий должен быть надежным, т.е. любые два множества тестов, удовлетворяющих ему, одновременно должны раскрывать или не раскрывать ошибки программы.

- Критерий должен быть легко проверяемым, например, вычисляемым на тестах.

**Классы критериев:**

- Структурные критерии используют информацию о структуре программы (критерии так называемого «белого ящика»).

- Функциональные критерии формулируются в описании требований к программному изделию (критерии так называемого «черного ящика»).

- Критерии стохастического тестирования формулируются в терминах проверки наличия заданных свойств у тестируемого приложения, средствами проверки некоторой статистической теории.

- Мутационные критерии ориентированы на проверку свойств программного изделия на основе подхода Монте-Карло.

**Структурные критерии (класс I).**

Структурные критерии используют модель программы в виде «белого ящика», что предполагает знание исходного текста программы или спецификации программы в виде потокового графа управления. Структурная информация понятна и доступна разработчикам подсистем и модулей приложения, поэтому данный класс критериев часто используется на этапах модульного и интеграционного тестирования.

Структурные критерии базируются на операторах, ветвях и путях.

Условие критерия тестирования команд (критерий С0) – набор тестов в совокупности должен обеспечить прохождение каждой команды не менее одного раза. Это слабый критерий, используется в больших программных системах, где другие критерии применить невозможно.

Условие критерия тестирования ветвей (критерий С1) – набор тестов в совокупности должен обеспечить прохождение каждой ветви не менее одного раза. Это достаточно сильный и при этом экономичный критерий. Данный критерий часто используется в системах автоматизации тестирования.

Условие критерия тестирования путей (критерий С2) – набор тестов в совокупности должен обеспечить прохождение каждого пути не менее одного раза. Если программа содержит цикл (в особенности с неявно заданным числом итераций), то число итераций ограничивается константой (часто – 2, или числом классов выходных путей).

Структурные критерии не проверяют соответствие спецификации, если оно не отражено в структуре программы.

**Функциональные критерии (класс II).**

Функциональный критерий – важнейший для программной индустрии критерий тестирования. Он обеспечивает, прежде всего, контроль степени выполнения требований заказчика в программном продукте. Поскольку требования формулируются к продукту в целом, они отражают взаимодействие тестируемого приложения с окружением. При функциональном тестировании преимущественно используется модель «черного ящика». Проблема функционального тестирования – это, прежде всего, трудоемкость; дело в том, что документы, фиксирующие требования к программному изделию (Software requirement specification, Functional specification и т.п.), как правило, достаточно объемны, тем не менее, соответствующая проверка должна быть всеобъемлющей.

Ниже приведены частные виды функциональных критериев.

- *Тестирование пунктов спецификации* - набор тестов в совокупности должен обеспечить проверку каждого тестируемого пункта не менее одного раза. Спецификация требований может содержать сотни и тысячи пунктов требований к программному продукту и каждое из этих требований при тестировании должно быть проверено в соответствии с критерием не менее чем одним тестом.

- *Тестирование классов входных данных* – набор тестов в совокупности должен обеспечить проверку представителя каждого класса входных данных не менее одного раза. При создании тестов классы входных данных сопоставляются с режимами использования тестируемого компонента или подсистемы приложения, что заметно сокращает варианты перебора, учитываемые при разработке тестовых наборов. Перебирая в соответствии с критерием величины входных переменных (например, различные файлы – источники входных данных), мы вынуждены применять мощные тестовые наборы. Действительно, наряду с ограничениями на величины входных данных, существуют ограничения на величины входных данных во всевозможных комбинациях, в том числе проверка реакций системы на появление ошибок в значениях или структурах входных данных. Учет этого многообразия – процесс трудоемкий, что создает сложности для применения критерия.

- *Тестирование правил* – набор тестов в совокупности должен обеспечить проверку каждого правила, если входные и выходные значения описываются набором правил некоторой грамматики. Следует заметить, что грамматика должна быть достаточно простой, чтобы трудоемкость разработки соответствующего набора тестов была реальной (вписывалась в сроки и штат специалистов, выделенных для реализации фазы тестирования).

- *Тестирование классов выходных данных* – набор тестов в совокупности должен обеспечить проверку представителя каждого выходного класса, при условии, что выходные результаты заранее расклассифицированы, причем отдельные классы результатов указывают, в том числе ограничения на ресурсы или на время (time out). При создании тестов классы выходных данных сопоставляются с режимами использования тестируемого компонента или подсистемы, что заметно сокращает варианты перебора, учитываемые при разработке тестовых наборов.

- *Тестирование функций* – набор тестов в совокупности должен обеспечить проверку каждого действия, реализуемого тестируемым модулем, не менее одного раза. Очень популярный на практике критерий, который, однако, не обеспечивает покрытия части функциональности тестируемого компонента, связанной со структурными и поведенческими свойствами, описание которых не сосредоточено в отдельных функциях (т.е. описание рассредоточено по компоненту).

Критерий тестирования функций объединяет отчасти особенности структурных и функциональных критериев. Он базируется на модели «полупрозрачного ящика», где явно указаны не только входы и выходы тестируемого компонента, но также состав и структура используемых методов (функций, процедур) и классов.

- *Комбинированные критерии для программ и спецификаций –* набор тестов в совокупности должен обеспечить проверку всех комбинаций непротиворечивых условий программ и спецификаций не менее одного раза. При этом все комбинации непротиворечивых условий надо подтвердить, а условия противоречий следует обнаружить и ликвидировать.

**Стохастические критерии (класс III).**

Стохастическое тестирование применяется при тестировании сложных программных комплексов – когда набор детерминированных тестов (X, Y) имеет громадную мощность. В случаях, когда подобный набор невозможно разработать и исполнить на фазе тестирования, можно применить следующую методику.

- Разработать программы-имитаторы случайных последовательных входных сигналов {x}.

- Вычислить независимым способом значения {y} для соответствующих входных сигналов {y} и получить тестовый набор {X,Y}.

- Протестировать приложение на тестовом наборе {X,Y}, используя два способа контроля результатов:

1.Детерминированный контроль – проверка соответствия вычисленного значения значению y, полученному в результате прогона теста на наборе {x} – случайной последовательности входных сигналов, сгенерированной имитатором.

2.Стохастический контроль – проверка соответствия множества { }, полученного в результате прогона тестов на наборе значений {x}, заранее известному распределению результатов F(Y). В этом случае множество y неизвестно (его вычисление невозможно), но известен закон распределения данного множества. в y

Критерии стохастического тестирования:

Статистические методы окончания тестирования – стохастические методы принятия решений о совпадении гипотез о распределении случайных величин. К ним принадлежат широко известные: метод Стьюдента (St), метод Хи-квадрат (x2) и т.п.

Метод оценки скорости выявления ошибок – основан на модели скорости выявления ошибок, согласно которой тестирование прекращается, если оцененный интервал времени между текущей ошибкой и следующей слишком велик для фазы тестирования приложения.

**Мутационный критерий (класс IV).**

Считается, что профессиональные программисты пишут сразу почти правильные программы, отличающиеся от правильных мелкими ошибками или описками типа – перестановка местами максимальных значений индексов в описании массивов, ошибки в знаках арифметических операций, занижение или завышение границы цикла на 1 и т.п. Предлагается подход, позволяющий на основе мелких ошибок оценить общее число ошибок, оставшихся в программе.

Подход базируется на следующих понятиях:

Мутации – мелкие ошибки в программе.

Мутанты – программы, отличающиеся друг от друга мутациями.

Метод мутационного тестирования – в разрабатываемую программу P вносят мутации, т.е. искусственно создают программы-мутанты P1, P2…Затем программа P и ее мутанты тестируются на одном и том же наборе тестов {X,Y}.

Если на наборе {X,Y} подтверждается правильность программы P и, кроме того, выделяются все внесенные в программы-мутанты ошибки, то набор тестов (X,Y) соответствует мутационному критерию, а тестируемая программа объявляется правильной.

Если некоторые мутанты не выявили всех мутаций, то надо расширять набор тестов (X,Y) и продолжать тестирование.

**Методы тестирования комплексных систем.**

1 Тестирование «белого ящика».

Известна: внутренняя структура программы.

Исследуются: внутренние элементы программы и связи между ними.

2 Тестирование «черного ящика».

Известны: функции программы.

Исследуется: работа каждой функции на всей области определения.

**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.

**Планирование** - совокупность процессов по:

- созданию стратегии тестирования;

- оценке трудозатрат;

- прогнозированию сроков;

- назначению и оптимизации ресурсов;

- контролю выполнения задач.

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

К высокоуровневым задачам отчётности относятся:

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

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

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

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

**Задачи планирования**

К высокоуровневым задачам планирования относятся:

- снижение неопределённости;

- повышение эффективности;

- улучшение понимания целей;

- создание основы для управления процессами.

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

Тест план включает в себя:

- описание объекта тестирования;

- цели тестирования и объем работ;

- стратегию тестирования (методика, инструменты);

- приоритеты проекта и роль тестирования в общем процессе;

- процесс тестирования (график работ, критерии начала и окончания тестирования, специальные знания, оценка рисков);

- тестовую среду;

- тестовые сценарии (тест кейсы).

Хороший тест план должен отвечать на следующие вопросы:

**Что надо тестировать?**

- описание объекта тестирования: системы, приложения, оборудования

**Что будете тестировать?**

- список функций и описание тестируемой системы и её компонент в отдельности

**Как будете тестировать?**

- стратегия тестирования, а именно: виды тестирования и их применение по отношению к объекту тестирования

**Когда будете тестировать?**

- последовательность проведения работ: подготовка, тестирование, анализ результатов.

**Критерии начала тестирования:**

- готовность тестовой платформы

- законченность разработки требуемого функционала

- наличие всей необходимой документации

**Критерии окончания тестирования:**

- требования к количеству открытых багов выполнены

- выдержка определенного периода без изменения исходного кода приложения Code Freeze

- выдержка определенного периода без открытия новых багов Zero Bug Bounce

**Кто ответственный или будет тестировать?**

Тест-план включает в себя следующие разделы.

- **Цель** (purpose). Предельно краткое описание цели разработки приложения (в этом разделе информация подаётся в сжатом виде и в контексте того, на что следует обращать первостепенное внимание при организации тестирования и повышения качества).

- **Области, подвергаемые тестированию** (features to be tested). Перечень функций и/или нефункциональных особенностей приложения, которые будут подвергнуты тестированию

- **Области, не подвергаемые тестированию** (features not to be tested). Перечень функций и/или нефункциональных особенностей приложения, которые не будут подвергнуты тестированию. Причины исключения той или иной области из списка тестируемых могут быть самыми различными - от предельно низкой их важности для заказчика до нехватки времени или иных ресурсов.

- **Тестовая стратегия** (test strategy) и подходы (test approach). Описание процесса тестирования с точки зрения применяемых методов, подходов, видов тестирования, технологий, инструментальных средств.

- **Критерии** (criteria). Этот раздел включает следующие подразделы:

- **Приёмочные критерии, критерии качества** (acceptance criteria) — любые объективные показатели качества, которым разрабатываемый продукт должен соответствовать с точки зрения заказчика или пользователя, чтобы считаться готовым к эксплуатации.

- **Критерии начала тестирования** (entry criteria) — перечень условий, при выполнении которых команда приступает к тестированию.

- **Критерии приостановки тестирования** (suspension criteria) — перечень условий, при выполнении которых тестирование приостанавливается. Наличие этого критерия страхует команду от бессмысленной траты усилий в условиях, когда тестирование не принесёт ожидаемой пользы.

**- Критерии возобновления тестирования** (resumption criteria) — перечень условий, при выполнении которых тестирование возобновляется.

**- Критерии завершения тестирования** (exit criteria) — перечень условий, при выполнении которых тестирование завершается. Наличие этого критерия страхует команду как от преждевременного прекращения тестирования, так и от продолжения тестирования в условиях, когда оно уже перестаёт приносить ощутимый эффект.

- **Ресурсы** (resources). В данном разделе тест плана перечисляются все необходимые для успешной реализации стратегии тестирования ресурсы, которые в общем случае можно разделить на:

- программные ресурсы (например,какое ПО необходимо команде тестировщиков);

- аппаратные ресурсы (какое аппаратное обеспечение, в каком количестве и к какому моменту необходимо команде тестировщиков);

- человеческие ресурсы (сколько специалистов какого уровня и со знаниями в каких областях должно подключиться к команде тестировщиков в тот или иной момент времени);

- временные ресурсы (сколько по времени займёт выполнение тех или иных работ);

- **Расписание** (test schedule). Фактически это календарь, в котором указано, что и к какому моменту должно быть сделано. Особое внимание уделяется т.н. «ключевым точкам» (milestones), к моменту наступления которых должен быть получен некий значимый результат.

- **Роли и ответственность** (roles and responsibility). Перечень необходимых ролей (например, «ведущий тестировщик», «эксперт по оптимизации производительности») и область ответственности специалистов, выполняющих эти роли.

- **Оценка рисков** (risk evaluation). Перечень рисков, которые с высокой вероятностью могут возникнуть в процессе работы над проектом. По каждому риску даётся оценка представляемой им угрозы и приводятся варианты выхода из ситуации.

- **Документация** (documentation). Перечень используемой тестовой документации с указанием, кто и когда должен её готовить и кому передавать.

- **Метрики** (metrics). Числовые характеристики показателей качества, способы их оценки, формулы.

**Покрытие** (coverage) - процентное выражение степени, в которой исследуемый элемент (coverage) затронут соответствующим набором тест-кейсов.

**Инициализация** (от англ. initialization, инициирование) — создание, активация, подготовка к работе, определение параметров. Приведение программы или устройства в состояние готовности к использованию.

Выделяют 2 основных направления подготовки к процессу разработки ПО в деятельности отдела QA: снижение вероятности появления ошибок и гарантирование обработки всех ранее найденных ошибок.

Так для **снижения вероятности появления ошибок** мы:

- разрабатываем и применяем стандарты кодирования для разных языков программирования;

- разрабатываем и применяем метрики для оценки качества наших программных решений;

- проводим предварительную подготовку новых сотрудников.

Для **гарантирования обработки всех дефектов**:

- существует система уведомлений и напоминаний для программистов, тестировщиков и команд сопровождения ПО о новых и открытых проблемах;

- осуществляется регулярный мониторинг проблем, которые должны быть решены или находятся на разных этапах решения;

- применяем жесткий процесс обработки дефектов.

**Этапы процесса обеспечения качества:**

— проверка требований к продукту;

— оценка рисков;

— планирование идей по улучшению качества продукта;

— планирование тестирования;

— анализ результатов тестирования.