# **История тестирования программного обеспечения**

Первые программные системы разрабатывались в рамках программ научных исследований или программ для нужд министерств обороны. Тестирование таких продуктов проводилось строго формализовано с записью всех тестовых процедур, тестовых данных, полученных результатов. Тестирование выделялось в отдельный процесс, который начинался после завершения кодирования, но при этом, как правило, выполнялось тем же персоналом.

В 1960-х много внимания уделялось «исчерпывающему» тестированию, которое должно проводиться с использованием всех путей в коде или всех возможных входных данных. Было отмечено, что в этих условиях полное тестирование программного обеспечения невозможно, потому что, во-первых, количество возможных входных данных очень велико, во-вторых, существует множество путей, в-третьих, сложно найти проблемы в архитектуре и спецификациях. По этим причинам «исчерпывающее» тестирование было отклонено и признано теоретически невозможным.

В начале 1970-х тестирование программного обеспечения обозначалось как «процесс, направленный на демонстрацию корректности продукта» или как «деятельность по подтверждению правильности работы программного обеспечения». В зарождавшейся программной инженерии верификация программного обеспечения значилась как «доказательство правильности». Хотя концепция была теоретически перспективной, на практике она требовала много времени и была недостаточно всеобъемлющей. Было решено, что доказательство правильности – неэффективный метод тестирования программного обеспечения. Однако, в некоторых случаях демонстрация правильной работы используется и в наши дни, например, приемо-сдаточные испытания. Во второй половине 1970-х тестирование представлялось как выполнение программы с намерением найти ошибки, а не доказать, что она работает. Успешный тест – это тест, который обнаруживает ранее неизвестные проблемы. Данный подход прямо противоположен предыдущему. Указанные два определения представляют собой «парадокс тестирования», в основе которого лежат два противоположных утверждения: с одной стороны, тестирование позволяет убедиться, что продукт работает хорошо, а с другой – выявляет ошибки в программном обеспечении, показывая, что продукт не работает. Вторая цель тестирования является более продуктивной с точки зрения улучшения качества, так как не позволяет игнорировать недостатки программного обеспечения.

В 1980-х тестирование расширилось таким понятием, как предупреждение дефектов. Проектирование тестов – наиболее эффективный из известных методов предупреждения ошибок. В это же время стали высказываться мысли, что необходима методология тестирования, в частности, что тестирование должно включать проверки на всем протяжении цикла разработки, и это должен быть управляемый процесс. В ходе тестирования надо проверить не только собранную программу, но и требования, код, архитектуру, сами тесты. «Традиционное» тестирование, существовавшее до начала 1980-х, относилось только к скомпилированной, готовой системе (сейчас это обычно называется системное тестирование), но в дальнейшем тестировщики стали вовлекаться во все аспекты жизненного цикла разработки. Это позволяло раньше находить проблемы в требованиях и архитектуре и тем самым сокращать сроки и бюджет разработки. В середине 1980-х появились первые инструменты для автоматизированного тестирования. Предполагалось, что компьютер сможет выполнить больше тестов, чем человек, и сделает это более надежно. Поначалу эти инструменты были крайне простыми и не имели возможности написания сценариев на скриптовых языках.

В начале 1990-х в понятие «тестирование» стали включать планирование, проектирование, создание, поддержку и выполнение тестов и тестовых окружений, и это означало переход от тестирования к обеспечению качества, охватывающего весь цикл разработки программного обеспечения. В это время начинают появляться различные программные инструменты для поддержки процесса тестирования: более продвинутые среды для автоматизации с возможностью создания скриптов и генерации отчетов, системы управления тестами, программного обеспечения для проведения нагрузочного тестирования. В середине 1990-х с развитием Интернета и разработкой большого количества веб-приложений особую популярность стало получать «гибкое тестирование» (по аналогии с гибкими методологиями программирования).

В 2000-х появилось еще более широкое определение тестирования, когда в него было добавлено понятие «оптимизация бизнес-технологий» (BTO). BTO направляет развитие информационных технологий в соответствии с целями бизнеса. Основной подход заключается в оценке и максимизации значимости всех этапов жизненного цикла разработки программного обеспечения для достижения необходимого уровня качества, производительности, доступности.

# **Модель качества продукта по стандарту ISO/IEC 25010:2011**

*Качество в использовании* - степень применимости продукта или системы заданными пользователями для удовлетворения их потребностей в достижении заданных целей с результативностью, эффективностью, свободой от риска и удовлетворенностью в заданных контекстах использования.

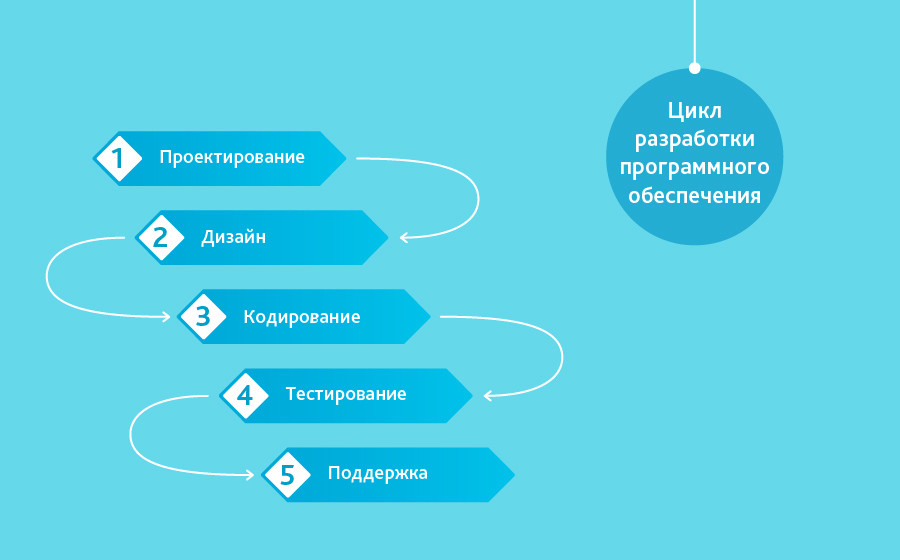
Качество в использовании характеризует влияние, которое продукт оказывает на правообладателей. Оно определяется качеством программного обеспечения, аппаратных средств и эксплуатационной среды, а также характеристиками пользователей, задач и социального окружения

Данная модель состоит из 5 характеристик:

* Результативность - точность и полнота, с которой пользователи достигают заданных целей.
* Эффективность - ресурсы, затрачиваемые в зависимости от точности и полноты, с которыми пользователь достигает целей.
* Удовлетворенность - степень удовлетворения потребностей пользователя при применении продукта или системы в заданном контексте использования. Подхарактеристики: применимость, доверие, удовольствие, комфорт.
* Свобода от риска - степень уменьшения продуктом или системой потенциального риска для экономического статуса, человеческой жизни, здоровья или окружающей среды. Подхарактеристики: уменьшение экономического риска, уменьшение риска для здоровья и безопасности, уменьшение риска для окружающей среды.
* Покрытие контекста - степень применимости продукта или системы с результативностью, эффективностью, свободой от риска и удовлетворенностью как в заданных контекстах использования, так и вне них. Подхарактеристики: покрытия контекста являются полнота контекста, гибкость.

# **Процесс и методологии разработки программного обеспечения (Водопадная, V-образная, Итерационная)**

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

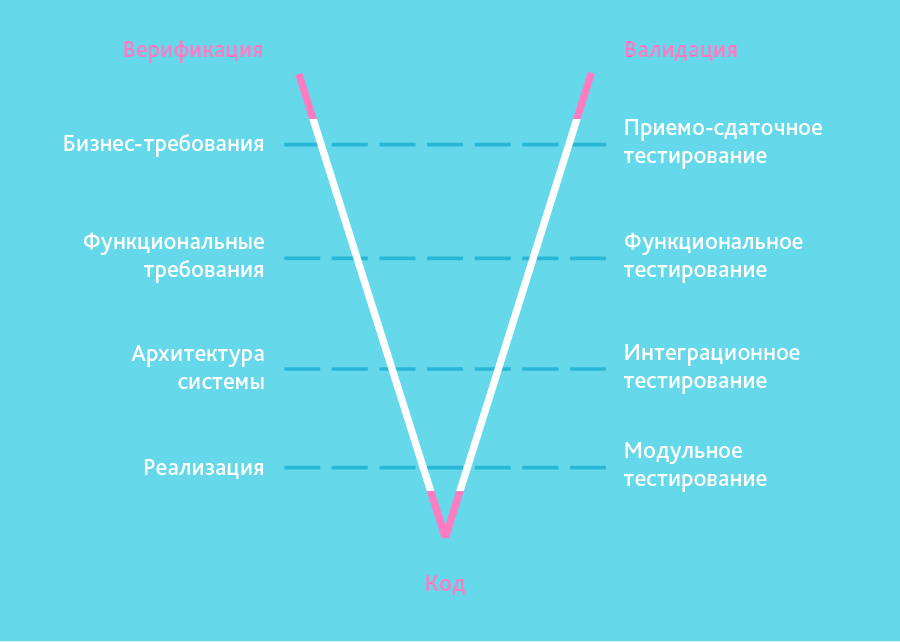
**

Одна из самых старых, подразумевает последовательное прохождение стадий, каждая из которых должна завершиться полностью до начала следующей. В модели Waterfall легко управлять проектом. Благодаря её жесткости, разработка проходит быстро, стоимость и срок заранее определены. Но это палка о двух концах. Каскадная модель будет давать отличный результат только в проектах с четко и заранее определенными требованиями и способами их реализации. 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. Тем не менее, фиксированная стоимость часто перевешивает минусы подхода. Исправление осознанных в процессе создания недостатков возможно, и, по опыту, требует от одного до трех дополнительных соглашений к контракту с небольшим ТЗ.

Когда использовать каскадную методологию?

* Только тогда, когда требования известны, понятны и зафиксированы. Противоречивых требований не имеется.
* Нет проблем с доступностью программистов нужной квалификации.
* В относительно небольших проектах.

*«V-Model»*

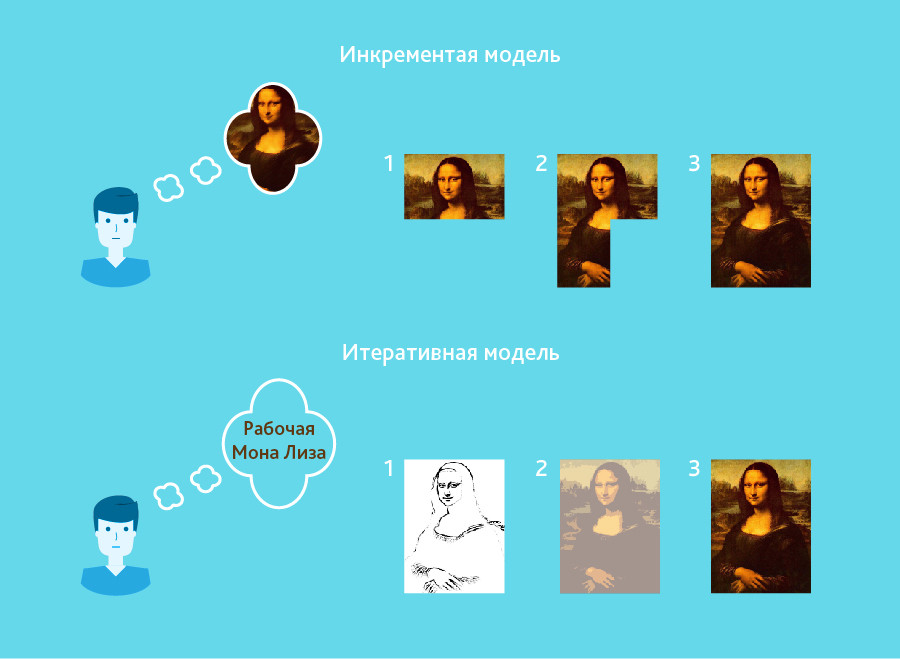


Унаследовала структуру «шаг за шагом» от каскадной модели. V-образная модель применима к системам, которым особенно важно бесперебойное функционирование. Например, прикладные программы в клиниках для наблюдения за пациентами, интегрированное ПО для механизмов управления аварийными подушками безопасности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

Когда использовать V-модель?

* Если требуется тщательное тестирование продукта, то V-модель оправдает заложенную в себя идею: validation and verification.
* Для малых и средних проектов, где требования четко определены и фиксированы.
* В условиях доступности инженеров необходимой квалификации, особенно тестировщиков.

«Iterative Model»



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 неидеальной, главное, чтобы она работала. Понимая конечную цель, мы стремимся к ней так, чтобы каждый шаг был результативен, а каждая версия — работоспособна.

На диаграмме показана итерационная «разработка» Мона Лизы. Как видно, в первой итерации есть лишь набросок Джоконды, во второй — появляются цвета, а третья итерация добавляет деталей, насыщенности и завершает процесс. В инкрементной же модели функционал продукта наращивается по кусочкам, продукт составляется из частей. В отличие от итерационной модели, каждый кусочек представляет собой целостный элемент.

Примером итерационной разработки может служить распознавание голоса. Первые исследования и подготовка научного аппарата начались давно, в начале — в мыслях, затем — на бумаге. С каждой новой итерацией качество распознавания улучшалось. Тем не менее, идеальное распознавание еще не достигнуто, следовательно, задача еще не решена полностью.

Когда оптимально использовать итеративную модель?

* Требования к конечной системе заранее четко определены и понятны.
* Проект большой или очень большой.
* Основная задача должна быть определена, но детали реализации могут эволюционировать с течением времени.

# **Процесс и методологии разработки программного обеспечения (Спиральная, Гибкая, SCRUM)**

«Spiral Model»



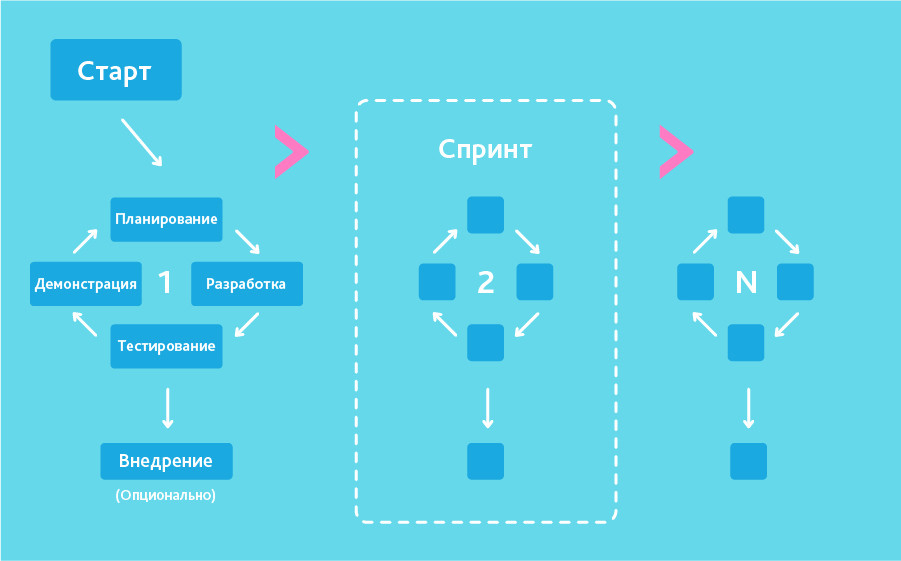
Спиральная модель предполагает 4 этапа для каждого витка:

* Планирование
* Анализ рисков
* Конструирование
* оценка результата и при удовлетворительном качестве переход к новому витку

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. На проекте по разработке СЭД для ОДУ Сибири СО ЕЭС два совещания об изменении кодификации разделов электронного архива занимают в 10 раз больше времени, чем объединение двух папок программистом. Государственные проекты, в которых мы участвовали, начинались с подготовки экспертным сообществом дорогостоящей концепции, которая отнюдь не всегда бесполезна, поскольку окупается в масштабах страны.

«Agile Model» & «Scrum»



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.

В основе такого типа — непродолжительные ежедневные встречи — «Scrum» и регулярно повторяющиеся собрания (раз в неделю, раз в две недели или раз в месяц), которые называются «Sprint». На ежедневных совещаниях участники команды обсуждают:

* отчёт о проделанной работе с момента последнего Scrum’a;
* список задач, которые сотрудник должен выполнить до следующего собрания;
* затруднения, возникшие в ходе работы.

Методология подходит для больших или нацеленных на длительный жизненный цикл проектов, постоянно адаптируемых к условиям рынка. Соответственно, в процессе реализации требования изменяются. Стоит вспомнить класс творческих людей, которым свойственно генерировать, выдавать и опробовать новые идеи еженедельно или даже ежедневно. Гибкая разработка лучше всего подходит для этого психотипа руководителей. Внутренние стартапы компании мы разрабатываем по Agile. Примером клиентских проектов является Электронная Система Медицинских Осмотров, созданная для проведения массовых медосмотров в считанные минуты. Во втором абзаце этого отзыва, наши американские партнеры описали очень важную вещь, принципиальную для успеха на Agile.

Когда использовать Agile:

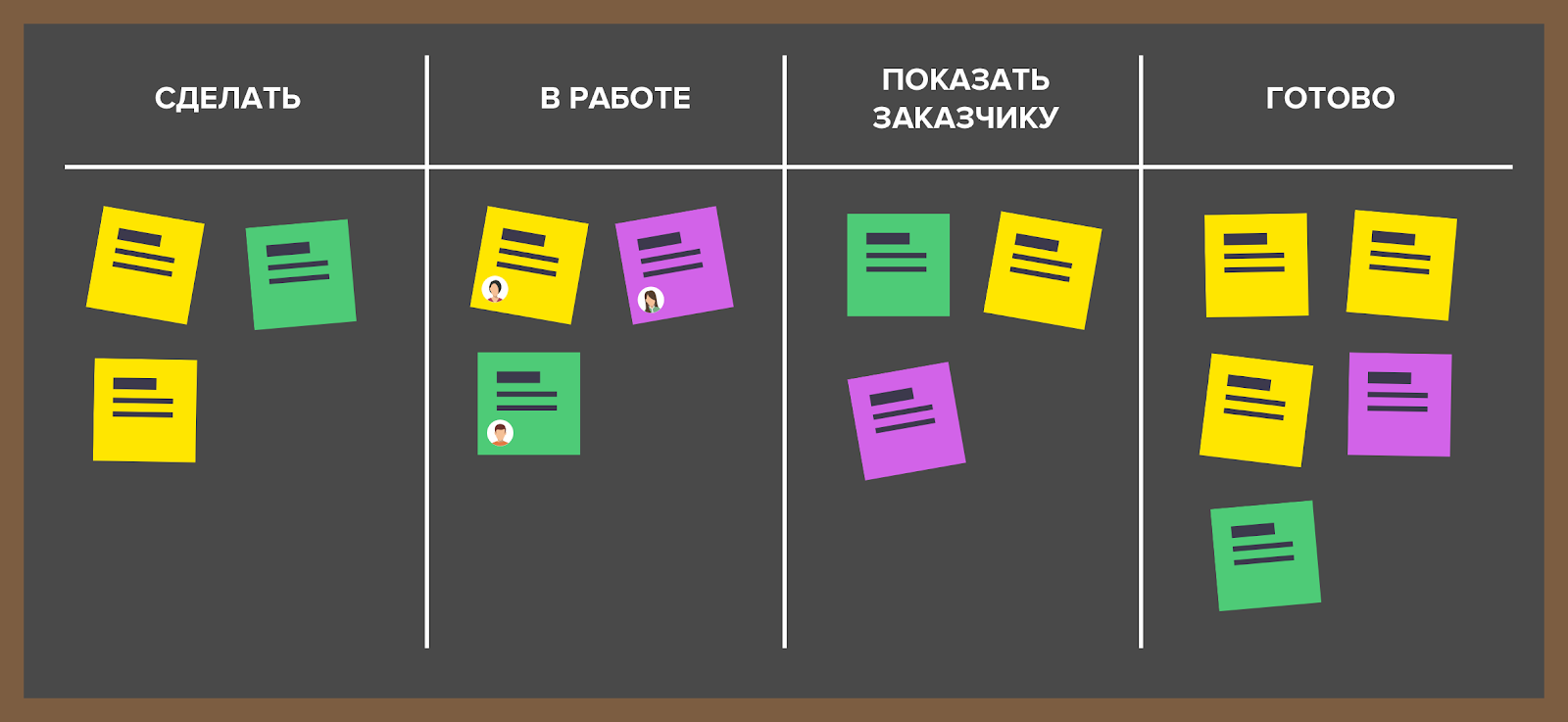
* Когда потребности пользователей постоянно меняются в динамическом бизнесе.
* Изменения на Agile реализуются за меньшую цену из-за частых инкрементов.
* В отличие от модели водопада, в гибкой модели для старта проекта достаточно лишь небольшого планирования.

# **Процесс и методологии разработки программного обеспечения (Канбан, Экстремальное программирование)**

Канбан-доска позволяет вывести процесс выполнения задач в визуальное восприятие. Такой подход помогает видеть весь рабочий процесс, четко распределять задачи и вовремя направлять усилия в «слабые» зоны.

Это работает так: столбики представляют собой разные этапы, на которые разбивают рабочий процесс. Карточки в столбцах — это конкретные задачи-шаги. За каждый этап несет ответственность отдел/сотрудник. Карточки перемещаются по столбцам в соответствии со своим статусом.

При этом принцип формирования каждого столбца должен быть один. Например, это могут быть этапы производственного процесса («прототипирование», «дизайн», «разработка», «тестирование») или статусы выполнения задач («предстоит сделать», «в работе», «на проверке», «завершено»). По каждой колонке должно быть определено ограничение объема незавершенной работы — это позволяет предупредить перегрузы и простои. Этот принцип берет свое начало в законе американского ученого Джона Литтла, согласно которому при увеличении количества одновременно выполняемых задач, снижается скорость выполнения каждой из них. Поэтому команды постоянно балансируют между ограничением на невыполненную работу и скоростью пропускной системы. Лучшие практики ведения канбан-доски основаны на простых компонентах — обсуждение, баланс и взаимодействие.



«XP»

Цель методики XP — справиться с постоянно меняющимися требованиями к программному продукту и повысить качество разработки. Поэтому XP хорошо подходит для сложных и неопределенных проектов

Методология XP строится вокруг четырех процессов: кодирования, тестирования, дизайна и слушания. Кроме того, экстремальное программирование имеет ценности: простоту, коммуникацию, обратную связь, смелость и уважение.

Далее идет ряд перечень основных тезисов-компонентов, каждый из которых может быть отдельным вопросом, поэтому ОЧЕНЬ сокращу:

* *Вся команда*. Все участники проекта с применением XP работают как одна команда. В нее обязательно входит представитель заказчика, лучше, если это будет реальный конечный пользователь продукта, разбирающийся в бизнесе.
* *Игра в планирование.* В ходе такой игры можно быстро собрать известные требования к системе, оценить и запланировать их разработку в соответствии с приоритетностью. Как и любая другая игра, планирование имеет своих участников и свою цель. Ключевой фигурой является, конечно же, заказчик. Именно он сообщает о необходимости той или иной функциональности. Программисты же дают ориентировочную оценку каждой функциональности. Прелесть игры в планирование заключается в единстве цели и солидарности разработчика и заказчика: в случае победы побеждают все, в случае поражения все проигрывают. Но при этом каждый участник идет к победе своей дорогой: заказчик выбирает наиболее важные задачи в соответствии с бюджетом, а программист оценивает задачи в соответствии со своими возможностями по их реализации.
* *План релизов.* План релизов определяет даты релизов и формулировки пользователей, которые будут воплощены в каждом из них.
* *Планирование итераций.* Планирование итераций начинается со встречи в начале каждой итерации с целью выработки плана шагов для решения программных задач. Каждая итерация должна длиться от одной до трех недель. Формулировки внутри итерации сортируются в по-рядке их значимости для заказчика. Кроме того, добавляются задачи, которые не смогли пройти тесты приемки и требуют доработки. Формулировки и результаты тестов переводятся в программные задачи. Задачи записываются на карточках, которые образуют детальный план итерации. Для решения к каждой из задач требуется от одного до трех дней. Задачи, для которых нужно менее одного дня, можно сгруппировать вместе, а большие задачи разделить на несколько мелких. Разработчики оценивают задачи и сроки, для их выполнения.
* *Stand Up.* Каждое утро проводится собрание для обсуждения проблем, их решений и для усиления концентрации команды. Собрание проводится стоя для избежания длительных дискуссий не интересных всем членам команды. Ну тут дефолт, подробно расписывать не буду.
* *Простота.* Простой дизайн всегда занимает меньше времени, чем сложный. Поэтому всегда делайте самые простые вещи, которые только смогут работать.
* *Система метафор.* Выбор системы метафор нужен, чтобы удержать команду в одних и тех же рамках при именовании классов и методов. То, как вы называете свои объекты, очень важно для понимания общего дизайна системы и повторного использования кодов.
* *Заказчик на рабочей площадке.* Основной проблемой разработки программного обеспечения является недостаток знаний программистов в разрабатываемой предметной области. Экстремальное программирование нашло выход и из этой ситуации. Это участие заказчика в процессе разработки. Если привлечь заказчика или его представителя не удается, иногда оказывается целесообразным временный наем специалиста в разрабатываемой области.
* *Тестирование до начала разработки.* В экстремальном программировании роль тестирования интереснее: теперь вначале идет тест, а потом код. Как же тестировать то, чего еще нет? Ответ прост и банален: тестируйте свои мысли — чего следует ожидать от будущего куска программы или функциональности. Это позволит лучше понять, что требуется сделать программистам, и проверить работоспособность кода сразу, как только он будет написан.
* *Парное программирование.*
* *Смена позиций. Во время очередной итерации всех работников следует перемещать на новые участки работы.*
* *Коллективное владение кодом.* Коллективное владение кодом стимулирует разработчиков подавать идеи для всех частей проекта, а не только для своих модулей. Любой разработчик может изменять любой код для расширения функциональности и исправления ошибок.
* Code Convention. Договариваемся о стиле написания.
* *Частая интеграция.* Разработчики, по-возможности, должны интегрировать и выпускать свой код каждые несколько часов. В любом случае никогда нельзя держать изменения дольше одного дня. Частая интеграция позволяет избежать отчуждения и фрагментирования в разработке, когда разработчики не могут общаться в смысле обмена идеями или повторного использования кода. Каждый должен работать с самой последней версией.
* *Сорокачасовая рабочая неделя.* Восьмичасовой рабочий день и пятидневная рабочая неделя. Так совпадают биоритмы и остальные практики реализовать проще с одинаковым графиком для каждого сотрудника.

# **Процесс тестирования программного обеспечения (Планирование и управление тестированием, Анализ и проектирование тестов, Внедрение и реализация тестов, Оценка критериев выхода и создание отчетов)**

**Управление тестированием** — это процесс управления тестовой деятельностью с целью обеспечения высококачественного и высококлассного тестирования программного приложения. Метод заключается в организации, контроле, обеспечении отслеживания и видимости процесса тестирования с целью создания высококачественного программного приложения. Это обеспечивает выполнение процесса тестирования программного обеспечения в соответствии с ожиданиями.

**План тестирования** можно определить как документ, описывающий объем, подход, ресурсы и график предполагаемых мероприятий по тестированию.

При тесте программного обеспечения план предоставляет подробную информацию о предстоящем тестировании, включая:

* Стратегия тестирования
* Цель тестирования
* Критерии выхода/приостановки
* Планирование ресурсов
* Результаты тестирования

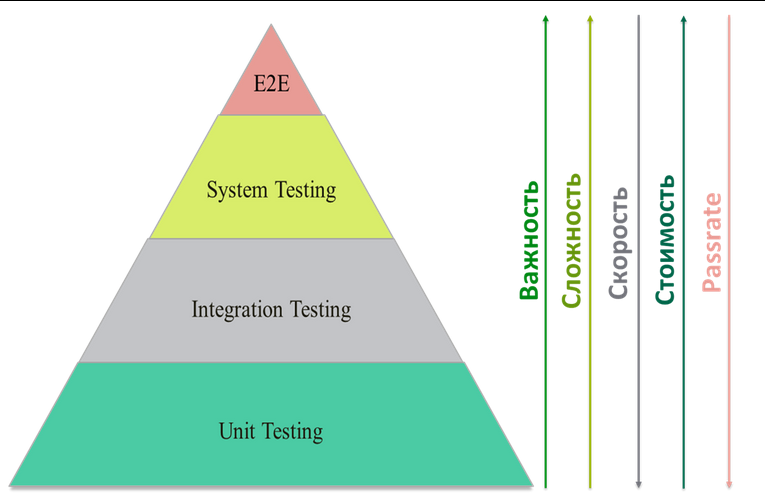
В рамках **анализа тестирования**, нам нужно определить следующие вещи:

* Уровни тестирования — на сколько глубоко нам нужно протестировать каждое требование (Компонентное, интеграционное и системное тестирование);
* Уровень детализации и качества наших требований;
* Связи между требованиями;
* Сложность нашего приложения;
* Продуктовые и проектные риски;
* Жизненный цикл разработки ПО и его длительность;
* Инструменты по управлению тестированием (Test management system);
* Опыт и постоянность команды;
* Доступность для консультации других участников проекта.

# **Уровни тестирования программного обеспечения, Верификация и валидация**

Уровни:

* Модульное тестирование (юнит)
* Интеграционное тестирование
* Системное тестирования
* Приемочное тестирование



**Верификация.** При верификации команда разработчиков изучает документы для создания программного обеспечения или приложения. Цель состоит в том, чтобы убедиться, что разработчик, которому поручен проект, соблюдает все изложенные требования. Логика кода должна соответствовать проектной документации независимо от языка программирования. Включает: *проверку требований, дизайна и логики кода.*

**Валидация.** Валидация включает в себя проверку приложения на разных этапах разработки, чтобы убедиться, что оно соответствует требованиям. Если в документе требуется веб-страница с функцией живого чата, то разработчик должен создать именно ее. Если что-то отсутствует или не соответствует запросу клиента, это следует выявить и исправить, чтобы получить ожидаемый продукт. Включает: *проверку дизайна, установки, функциональности, производительности и готовности.*

*Верификация (verification) — это процесс оценки системы, чтобы понять, удовлетворяют ли результаты текущего этапа разработки условиям, которые были сформулированы в его начале.*

*Валидация (validation) — это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, его требованиям к системе.*

# **Типы тестирования программного обеспечения (Статическое и динамическое, Ручное и автоматизированное)**

# **Статическое тестирование** — процесс тестирования, который проводится для верификации практически любого артефакта разработки: программного кода компонент, требований, системных спецификаций, функциональных спецификаций, документов проектирования и архитектуры программных систем и их компонентов.

# **Динамическое тестирование** — тестирование проводится на работающей системе, не может быть осуществлено без запуска программного кода приложения.

Если тест подразумевает выполнение большого числа повторяющихся задач, то его имеет смысл **автоматизировать**. Классический пример — регрессионное тестирование, позволяющее обнаружить серьёзные баги, поэтому его важно регулярно проводить в полном объеме до запуска конечного продукта.

Рассмотрим направления для применения автоматизации:

* Тестирование графического интерфейса
* Регрессионное тестирование (при добавлении новых функций)
* Функциональное тестирование

Когда для оценки работы программы ключевую роль начинают играть поведенческие особенности человека и интуиция, то стоит обратить внимание на **ручное тестирование** — здесь важнее поставить себя на место пользователя. Тестировщик может подходить творчески к процессу проверки функциональности: придумывать неожиданные идеи, имитировать необычные варианты использования — это помогает раскрыть критические ошибки, которые «не заметит» автоматизированное тестирование.

Применение:

* Юзабилити-тестирование
* Ad hoc тестирование (разовые тесты для проверки одного аспекта)

# 

# **Методы тестирования программного обеспечения (Белого, черного, серого ящиков)**

**Тестирование белого ящика** — метод тестирования ПО, который предполагает полный доступ к коду проекта.

**Тестирование серого ящика** — метод тестирования ПО, который предполагает частичный доступ к коду проекта (комбинация White Box и Black Box методов).

**Тестирование чёрного ящика** — метод тестирования ПО, который не предполагает доступа (полного или частичного) к системе. Основывается на работе исключительно с внешним интерфейсом тестируемой системы.

# **Виды тестирования программного обеспечения (функциональное, нефункциональное, структурное, тестирование изменений, тестирование по приоритетам)**

**Функциональное тестирование (functional testing)** — направлено на проверку корректности работы функциональности приложения.

**Нефункциональное тестирование (non-functional testing)** — тестирование атрибутов компонента или системы, не относящихся к функциональности.

**Структурное** **тестирование** — это тип тестирования программного обеспечения, который использует внутренний дизайн программного обеспечения для тестирования или, другими словами, тестирование программного обеспечения, которое выполняется командой, знающей фазу разработки программного обеспечения, известно как структурное тестирование.

**Регрессионное** **тестирование** — это вид тестирования направленный на *проверку изменений*, сделанных в приложении или окружающей среде (починка дефекта, слияние кода, миграция на другую операционную систему, базу данных, веб сервер или сервер приложения), для подтверждения того факта, что существующая ранее функциональность работает как и прежде.

**Срочность (priority)** показывает, как быстро дефект должен быть устранён. Priority выставляется менеджером, тимлидом или заказчиком

**Градация Приоритета дефекта (Priority):**

* P1 Высокий (High). Критическая для проекта ошибка. Должна быть исправлена как можно быстрее.
* P2 Средний (Medium). Не критичная для проекта ошибка, однако требует обязательного решения.
* P3 Низкий (Low). Наличие данной ошибки не является критичным и не требует срочного решения. Может быть исправлена, когда у команды появится время на ее устранение.

# **Функциональное тестирование, Модели поведения пользователя, Позитивное и негативное тестирование**

**Функциональное тестирование (functional testing)** — направлено на проверку корректности работы функциональности приложения.

Классификация по уровню функционального тестирования:

* Дымовое тестирование (smoke test) — тестирование, выполняемое на новой сборке, с целью подтверждения того, что программное обеспечение стартует и выполняет основные для бизнеса функции.
* Тестирование критического пути (critical path) — направлено для проверки функциональности, используемой обычными пользователями во время их повседневной деятельности.
* Расширенное тестирование (extended) — направлено на

**Позитивное тестирование** – это процесс проверки на корректное поведение согласно техническим требованиям и документации. Позитивное тестирование выполняется для обеспечения того, что система делает именно то, что ожидается.

**Негативное тестирование** – это процесс проверки на некорректное поведение. В ходе такого тестирования мы можем узнать, что система справится с непредвиденными ситуациями.

# **Исследовательское тестирование ПО (что это, отличие от сценарного тестирования, когда стоит и когда не стоит применять)**

**Ad-hoc тестирование**

Под ad-hoc тестированием будем понимать тестирование без использования спецификаций, планов и разработанных тест-кейсов: чистая импровизация.

**Исследовательское тестирование**

Более формальная версия ad-hoc: тестирование, не требующее написания тест-кейсов, но подразумевающее, что каждый последующий тест выбирается на основании результата предыдущего теста. А по Сэму Канеру, «Testing Computer Software», «исследовательское тестирование» — вдумчивый подход к ad-hoc тестированию.

**Сценарное тестирование**

Классическое тестирование по предварительно написанным и задокументированным сценариям.

**Исследовательское тестирование используется, когда:**

* Мало времени
* Требований нет, они не полны или устарели
* Небольшой проект
* Многократное прохождение тех же сценариев (глаз тестеров замыливается)
* Пришел внезапный запрос на изменения (чтобы не терять время на доки)
* Перестраховка (если после сценариев остались сомнения)

**Исследовательским тестированием НЕ обойтись, когда:**

* Приложение работает по ГОСТам и отклонения критичны
* Проводится интеграционное тестирование
* Тестовые сценарии отдаются на аутсорс
* Длительный проект

# **Исследовательское тестирование ПО (Туры, Туры по бизнес-центру)**

Это места, где «делается бизнес». Они начинаются с запуска кода и до завершения его работы. В них расположены функции, ради которых пользователи и используют данное приложение. Это «задняя стенка коробки» (читай – приложения), которая является главной при демонстрации коммерческой ценности продукта, а также код, который это поддерживает.

### *Тур по путеводителю.*

Путеводители часто содержат много информации о местах, которые можно посмотреть. Они рассказывают о лучших отелях, торговых комплексах и привлекательных местах без подробных деталей. Эксперты, посетившие эти места, любезно скажут туристам, как именно нужно наслаждаться этими местами. Здесь должно быть красиво, чисто, безопасно, чтобы туристы могли спокойно тратить свои деньги. Это очень важные места, и они должны быть включены в тестовую стратегию исследовательского тестирования. Мы должны заботиться о том, чтобы пользователи получали пользу и удовольствие от работы здесь.

Разновидностями тура также являются тур блоггера, который содержит советы третьей стороны, а так

же тур эксперта, который рассматривает отзывы неудовлетворенных пользователей. Этому могут помочь различные форумы, новости, сообщества или даже книги (в случае, если приложение большое). Еще один полезный тур – тур конкурента, который позволяет рассмотреть приложение со стороны конкурирующей системы.

### *Денежный тур*

Каждое приложение имеет функциональность, которая, в глазах пользователя (заказчика), принесет им деньги. «Это самое привлекательно для меня» — скажет в этом случае пользователь. «Денежная» функциональность несет с собой «денежную силу». Сюда входят различные нюансы использования продукта, поскольку именно они позволяют зарабатывать деньги. Специалисты по продажам могут направлять демонстрацию продукта в сторону указанных сценариев, затрагивая то, что не было четко зафиксировано в документации к продукту. Такое «продажное видение» – отличная основа для денежного тура.

### *Тур по отметкам*

Мы выбираем какие-либо ориентиры и осуществляем скачок к одному из них через все приложение. Например, мы выбираем ключевую функцию в качестве отметки, которую мы определили в туре по путеводителю или в денежном туре. Далее мы составляем набор из таких отметок, определяем их порядок и исследуем приложение от отметки к отметке, пока не достигнем каждой из них. Далее можно создать карту покрытия отметками. Разный порядок, разные отметки – вот поле для вариативности подобного тура.

### *Интеллектуальный тур*

Задавайте приложению сложные, интеллектуальные вопросы. Как можно заставить работать приложение так тяжело, как это возможно? Какие фичи приблизят его к собственному лимиту? Какой ввод и входные данные заставят его напрягаться? Какие данные могут обмануть процедуры проверки ошибок? Очевидно, это напрямую зависит от особенностей конкретного приложения.

### *Тур FedEx*

Думайте о данных, которые должны проходить через приложение. Данные начинают свой путь с ввода, далее хранятся во внутренних хранилищах и переменных, где часто происходит их обработка и изменение, а далее они могут подвергаться вычислениям. В конце большинство из таких данных предоставляются пользователю или куда-либо еще. Попробуйте найти все области, с которыми соприкасаются данные. Определите весь их жизненный цикл.

### *Тур «после работы»*

После денежной активности приложения, оно все еще продолжает работу. Это поддержка задач, архивация данных и файлов. Это может быть автоматическим процессом, но его также можно выполнять принудительно. Данный тур напоминает нам делать это.

### *Тур уборщика*

Указанный тур включает выбор цели (все меню, ошибки, диалоговые окна, к примеру) и посещение каждой из них наиболее коротким путем.

# **Исследовательское тестирование ПО (Туры, Туры историческому району, Туры по району развлечений)**

### **Исторический район**

Такие районы представляют legacy код, функции, представленные в старых версиях и исправленные ошибки, которые все также нужно проверять.

### *Тур «плохие соседи»*

Плохие соседи – те секции, в которых обнаруживается большое количество ошибок. Как можно определить эти секции? Заранее никак. А вот на опыте это делается на основе уже обнаруженных багов и их количества. Появляется возможность сделать некое предположение о местах их появления. Поскольку ошибки могут скапливаться локально, этот тур стоит того, чтобы его проходить. После работы с подобными ошибками рекомендуется пройти тур уборщика через близлежащие функции, чтобы убедиться, что исправления не задели их.

### *Музейный тур*

Музейные древности здесь — legacy код. Его можно определить по датам, указанной в репозитории. Такой код, использованный в новом окружении, склонен к плохонькой работе. Исследовать этот код сложно: программист может быть уже не в компании, а документация может быть скудной. Тестировщик должен выявить такой код и обратить внимание на его работу.

### *Тур прежней версии*

После апдейта предыдущей версии желательно провести тесты, которые были написаны для то, прежней версии. Это позволит убедиться в том, что функциональность, которую пользователи использовали ранее, все еще может быть использована полезно в новой версии продукта. В случае, если в новой версии произошло удаление или обновление функциональности, тестирование должно происходить соответствующим этим обновлениям образом.

### **Район развлечений**

Это возможность немного развлечься с приложением: сделать его более привлекательным, поиграть с шаблонами, цветами. Доп фичи и тд

### *Тур актера из команды поддержки*

Пользователь склонен к искушению фичами, которые лежат рядом с теми, которые освещаются. Тур актера из команды поддержки сосредотачивается на тех фичах, которые разделяют экран с основными. Их близость к основным увеличивает их видимость, и мы должны уделять им соответствующее внимание. Покрути головой – вправо и влево – и посмотри, уделили ли должное внимание актерам, которые поддерживают основных.

### *Тур по задней аллее*

Все то, что удостаивается наименьшего внимания, то, что используется наименее часто, что используется наиболее редко. Самое донышко, самые дальние полки, то, что находится за фасадом.

### *Тур «All night long»*

Клубный тур – никогда не останавливайся, еще один клуб, еще один коктейль. Сможешь ли ты выжить всю ночь? Накопление данных в памяти, запись (перезапись), многократное чтение переменных – это может привести к сбоям приложения: утечки памяти, нарушение данных, условия гонки и другое. Главное условие – никогда не закрывайте приложение. Открывайте файлы и не закрывайте их. Не сохраняйте файлы. Подсоединитесь к удаленным ресурсам и не отсоединитесь. И в то время, как действуют вышеописанные условия, можно пройти другие туры для работы приложения и обработки данных в нем. При перезагрузке приложения происходит очистка некоторых его участков – не допускайте этого.

# **Исследовательское тестирование ПО (Туры, Туры по туристическому району, Туры по району отелей)**

### **Туристический район**

Во многих городах есть места, которые интересны только туристам и пользователям-новичкам. Местные ребята избегают таких мест, поскольку им там тесно. То есть (прим. для ПО) пользователи, которые уже имеют опыт взаимодействия с приложением, подобные функции почти не используют. Их и тестим.

### *Тур коллекционера*

Коллекционируйте выходные данные, и чем больше, тем лучше. Идея в том, чтобы пройти туда, куда вы только можете и задокументировать все то, что вы увидите. Убедитесь, что вы увидите все выходные данные, которые может сгенерировать приложение. Для текстового редактора: печать, проверка орфографии, форматирование, различные структуры, таблицы, графика. Для магазина: возможность покупки из любого допустимого отдела, удачные и неудачные транзакции по картам. Нужно преследовать все возможные выходные данные, пока вы не сможете утвердить, что вы были везде, видели все и завершили свою коллекцию.

*Тур одинокого бизнесмена*

Идея тура в том, чтобы проверить самые дальние уголки приложения, какие только возможно. Какая фича требует наибольшего количества кликов? Выберите такую, «прокликайте» к ней путь и протестируйте. Какая фича требует наибольшего количества экранов, чтобы ей можно было воспользоваться? Выберите и протестируйте. Смысл в том, чтобы пропутешествовать как можно дольше до конечного пункта.

*Тур супермодели*

Этот тур не о функциях или чем-то существенном, а о том, как приложение выглядит и какое первое впечатление производит. Сосредоточьтесь на интерфейсе. Он хорош? Хорошо выполнен? Если я делаю изменения, отображается ли это на нем?

### *Тур «тестируй одно, другое – бесплатно»*

Этот тур для того, чтобы протестировать множественный запуск приложения одновременно. Запустите свое приложение, а потом еще раз, и еще. А теперь используйте фичи, которые относятся к памяти или дисковому пространству. Заставьте все копии приложения сделать что-то с тем же файлом или передачей каких-либо данных.

### *Тур шотландского паба*

Это особенно актуально для крупных приложений. В них есть такие места, которые достаточно сложно найти без чьей-либо помощи. Их не сложно использовать, их просто сложно отыскать. Таких мест может быть действительно много. Проблема в том, чтобы узнать о них. Как? Общаться с пользователями, просматривать какие-либо блоги и просто погружаться в приложение.

### **Район отелей**

Это место, где можно отойти от основных функций приложения, популярных фич, и протестировать второстепенные, поддерживающие функции, которые могут быть не отражены в тест планах.

### *Тур под дождем*

Идея в том, чтобы начать операцию и затем резко ее остановить, словно в этот момент пошел дождь и вам не хочется ходить куда-либо в это время. Начните вводить информацию о покупке с тем, чтобы сразу закончить эту процедуру и выйти; начните печать и отмените ее до того, как она распечатается. Если функционал имеет кнопку отмены или же если он выполняется дольше, чем несколько секунд – используйте этот тур.

### *Тур лежебоки*

Идея тура в том, чтобы делать настолько мало усилий, насколько это возможно. Оставляйте поля пустыми, принимайте значения по умолчанию, заполняйте формы наименьших количеством данных, не нажимайте ни на что дополнительное, пропускайте формы без нажатий или подайте туда все, что угодно. Если есть возможность выбора, выбирайте путь наименьшего сопротивления.

# **Исследовательское тестирование ПО (Туры, Туры по неблагополучному району)**

Входные данные должны ломать приложение.

*Тур диверсанта*

Пытайтесь поломать приложение любыми возможными путями. Попросите приложение прочитать какие-либо данные с диска, и далее саботируйте эту попытку, навредив операции и файлу. Попросите сделать какую-либо операцию, которая интенсивно использует память, когда приложение установлено на машине с малым количеством памяти, или, когда одновременно запущено другое приложение, потребляющее большое количество ресурсов.

*Антиобщественный тур*

Этот тур включает требование ввода наименее привлекательных данных и/или так негативного ввода. Если типичный пользователь делает «А», этот тур подразумевает, что тест-инженер никогда не будет делать «А», нужно найти что-то более бессмысленное.

### *Тур навязчивости и даже одержимости*

В этом туре нужно вводить одни и те же данные снова и снова, выполнять одно и то же действие опять и опять. Копируйте, вставляйте, вырезайте, заменяйте, а потом сделайте это еще раз. Название этому, обычно, — репетиция. Выберите товар, а потом снова этот же товар, чтобы проверить появляются ли скидки.

# **Тестирование требований (Почему важно? Что тестируется? Параметры тестирования документации)**

Тестирование требований направлено на то, чтобы уже на начальных этапах проектирования системы устранить максимально возможное количество ошибок.

Начать тестирование требований можно с поверхностного осмотра документации. Это сложно назвать именно тестированием, но нередко уже на данном этапе выявляется немало недочетов. Начнем с обычного сценария. Вы начали читать требования, и уже с первых строк у Вас возникает масса вопросов к автору (например, «Каков ожидаемый результат после нажатия на эту кнопку?» или «Что будет, если я отменю заказ?»). Это плохо. После прочтения документации не должно быть вопросов. Совсем. Требования – это как свод законов для продукта, а законы не допускают двусмысленность, «воду» и неточности. Документация должна давать предельно ясную информацию о том, как должен работать каждый отдельный модуль и весь продукт в целом. К сожалению, после прочтения большинства требований остается целый ряд вопросов.

* Четкость и ясность (без воды, хых)
* Актуальность (мы в 2023 живем…)
* Логика (например, отправка письма без авторизации)
* Возможные сценарии (очевидные, так и неочевидные варианты использования системы)
* Интеграция (с другими сервисами. тип выходим за рамки собственной документации вроде как)

# **Тестирование требований (Уровни и виды требований, Критерии качества требований, Методы тестирования требований)**

Тестирование требований - это процесс проверки и оценки требований, которые были представлены для программного обеспечения, для того чтобы убедиться что они являются полными, консистентными, достижимыми и соответствующими стандартам качества.

Уровни и виды требований :

* Требования бизнеса: описывают функциональность, которая должна быть реализована в системе для достижения целей бизнеса.
* Требования пользователя: описывают ожидания и потребности конечных пользователей от системы.
* Требования системы: описывают функциональность, которая должна быть реализована для достижения целей системы.

Критерии качества требований:

* Полнота: все ожидаемые функциональности системы должны быть описаны в трествует приемочным критериям, которые были определены для проекта.
* Консистентность: требования должны быть логичными и не противоречить друг другу.
* Достижимость: требования должны быть реализуемыми с текущими технологиями и имеющимися ресурсами.
* Соответствие стандартам: требования должны соответствовать общепринятым стандартам качества.

Методы тестирования требований:

* Ревью требований: проведение обзора и анализа требований с помощью команды разработчиков и экспертов.
* Тестирование сценариев: создание и исполнение тест-сценариев для проверки требований в реальных условиях.
* Тестирование системы: исполнение тестов системы для проверки того, что система соответствует требованиям.
* Тестирование приемочных критериев: проверка того, что система соответствует приемочным критериям, которые были определены для проекта.

Важно отметить, что тестирование требований должно проводится на каждом этапе разработки программного обеспечения, чтобы обеспечить соответствие требованиям и качество системы для конечных пользователей.

# **Нефункциональное тестирование (Что это, Его параметры, и виды. Инсталляционное тестирование)**

Нефункциональное тестирование - это процесс проверки нефункциональных характеристик программного обеспечения, таких как надежность, безопасность, производительность, использование ресурсов, качество пользовательского интерфейса и т.д.

Параметры, которые могут быть измерены в процессе нефункционального тестирования, включают

* Надежность: способность системы к восстановлению после сбоя и защита от непреднамеренного или несанкционированного доступа.
* Безопасность: защита от внешних угроз и обеспечение конфиденциальности данных.
* Производительность: скорость выполнения операций и способность системы к обработке большого количества данных и запросов.
* Использование ресурсов: оптимизация использования памяти, процессора и других ресурсов системы.
* Качество пользовательского интерфейса: удобство использования, интуитивность и доступность.

Есть несколько видов нефункционального тестирования, включая:

* Тестирование надежности: определение способности системы к восстановлению после сбоя и защита от непреднамеренного или несанкционированного доступа.
* Тестирование безопасности: оценка защиты системы от внешних угроз и обеспечение конфиденциальности данных.
* Тестирование производительности: измерение скорости выполнения операций и способности системы к обработке большого количества данных и запросов.

Инсталляционное тестирование - это процесс проверки правильности установки и настройки программного обеспечения на новом компьютере или системе. Оно включает в себя проверку соответствия требованиям системы, проверку функциональности и проверку соответствия документации. Это важно, чтобы убедиться, что программное обеспечение установлено и настроено правильно, и что оно может быть использовано для его предназначенного использования.

# **Нефункциональное тестирование (Тестирование производительности)**

Тестирование производительности - это один из видов нефункционального тестирования, который проводится для оценки скорости выполнения операций и способности системы к обработке большого количества данных и запросов.

Цели тестирования производительности:

* Определение точного времени выполнения операций и оценка их скорости
* Определение ограничений и проблем производительности системы
* Определение максимальной нагрузки, которую система может выдержать
* Определение оптимальных настроек для максимальной производительности

Для проведения тестирования производительности используются специальные инструменты и методики, такие как нагрузочное тестирование, тестирование производительности с использованием минимума и тестирование скалярной производительности.

* Нагрузочное тестирование: Это метод, который используется для тестирования производительности системы при использовании реалистичной или максимальной нагрузки.
* Тестирование производительности с использованием минимума : Это метод, который используется для тестирования производительности системы при использовании минимальной нагрузки.
* Тестирование скалярной производительности: Это метод, который используется для тестирования производительности системы в зависимости от количества пользователей или трафика.

Тестирование производительности является важным этапом в разработке программного обеспечения, так как оно позволяет определить ограничения и проблемы производительности системы и предпринять меры для их решения, чтобы обеспечить максимальную скорость и ответность системы для конечных пользователей.

# 

# **Нефункциональное тестирование (Тестирование эргономичности, Тестирование совместимости, Тестирование GUI)**

*Нефункциональное тестирование* - это вид тестирования, который проводится для проверки нефункциональных требований к программному обеспечению, таких как интерфейс пользователя, эргономика, совместимость и производительность.

*Тестирование эргономичности* - это процесс проверки того, насколько хорошо интерфейс пользователя соответствует потребностям и ожиданиям пользователей.

*Тестирование совместимости* - это процесс проверки того, как хорошо программное обеспечение работает в среде, в котоорой оно должно работать. Так, например, тестирование совместимости может включать в себя проверку совместимости программного обеспечения с различными операционными системами или браузерами.

*Тестирование GUI* - это процесс проверки графического интерфейса пользователя. Это может включать в себя тестирование таких аспектов, как внешний вид интерфейса, простота использования, нажимание кнопок, ввод данных, меню и т.д.

В общем нефункциональное тестирование позволяет определить качество интерфейса, совместимость с различными платформами, а также оптимизацию производительности системы.

# **Нефункциональное тестирование (Тестирование глобализации, A/B тестирование, Тестирование на отказ и восстановление системы, Тестирование на соответствие стандартам)**

*Нефункциональное тестирование* - это тип тестирования, который фокусируется на характеристиках программной системы, а не на ее функциональности. Оно включает в себя различные виды тестирования, такие как:

* Тестирование на глобализацию: Этот тип тестирования гарантирует, что программное приложение может поддерживать несколько языков и регионов.
* A/B тестирование: Этот тип тестирования используется для сравнения двух версий программного приложения, чтобы определить, какая из них работает лучше.
* Стресс-тестирование: Этот тип тестирования используется для определения того, как ведет себя программная система при высокой нагрузке или стрессе.
* Восстановительное тестирование: Этот тип тестирования используется для того, чтобы убедиться, что программная система может восстановиться после сбоев или других отказов.
* Тестирование на соответствие: Этот тип тестирования используется для обеспечения соответствия программной системы отраслевым стандартам и нормам.

Целью нефункционального тестирования является обеспечение соответствия программной системы требованиям к производительности, безопасности, удобству использования и другим нефункциональным аспектам.

# **Тестирование безопасности (Что это, Причины пробелов в безопасности, Тестирование проницаемости)**

Тестирование безопасности - это процесс оценки системы, сети или веб-приложения с целью выявления уязвимостей, которые может использовать злоумышленник. Это включает в себя различные техники, такие как тестирование проникновения и сканирование уязвимостей, для идентификации и оценки безопасности системы.

Причиной пробелов в безопасности могут быть различные факторы, такие как плохое проектирование или реализация, недостаток правильной настройки или обслуживания или использование устаревшего или неподдерживаемого ПО. Кроме того, постоянно обнаруживаются новые угрозы безопасности, поэтому даже хорошо защищенные системы могут стать уязвимыми со временем.

Тестирование проникновения (penetration testing) - это тип тестирования безопасности, который имитирует атаку на систему, чтобы выявить и оценить уязвимости. Он включает в себя попытку получить несанкционированный доступ к системе или сети, чтобы выявить слабые места и оценить эффективность контролей безопасности системы.

# **Тестирование безопасности (Виды атак)**

Есть множество различных типов атак, которые могут использоваться для эксплуатации уязвимостей безопасности в системе или сети. Некоторые из них включают:

Атаки на сеть: это атаки на инфраструктуру сети, которые могут включать в себя техники, такие как отказ в обслуживании (DoS) и распределенный отказ в обслуживании (DDoS) атаки.

Атаки на приложения: это атаки на определенные приложения или службы, которые могут включать в себя техники, такие как SQL-инъекция и кросс-сайтовая подделка скриптов (XSS).

Фишинг: это вид атак, который обычно использует техники социальной инженерии, чтобы обмануть пользователей, заставляя их раскрывать чувствительную информацию, например, учетные данные для входа

Атаки с использованием вредоносного ПО: это злонамеренное программное обеспечение, которое распространяется с целью нанести вред компьютерной системе, включая вирусы, черви, Трояны и шифровальщики.

Социальная инженерия: это не техническое использование психологии и поведения человека, обманывая людей нарушать процедуры безопасности или раскрывать чувствительную информацию.

Физические атаки: этот вид атаки включает в себя физическое уничтожение оборудования, кражу устройств и данных или нарушение энергоснабжения или систем охлаждения

Прослушивание/сниффинг: этот вид атаки перехватывает и мониторит трафик сети для кражи чувствительной информации.

Атаки "Man-in-the-middle" позволяют злоумышленнику перехватывать и читать или изменять сетевые коммуникации между двумя сторонами.

Это только некоторые из основных категорий атак, существует множество различных методов и способов атак, которые могут быть использованы злоумышленниками для выявлени

# **Тестирование безопасности (Виды вредоносного ПО)**

Виды вредоносного ПО (malware) включают:

* Вирусы: это программы, которые могут распространяться от компьютера к компьютеру и инфицировать другие файлы, изменяя или повреждая их.
* Черви: это вирусы, которые могут самораспространяться без помощи пользователя, используя сетевые протоколы.
* Трояны: это программы, которые могут удаленно контролировать компьютер или производить нежелательные действия на нем.
* Роботы/боты: это программы, которые могут автоматизировать определенные действия и контролировать удаленно через сеть.
* Adware: это программы, которые показывают рекламу на компьютере без согласия пользователя.
* Шпионские программы: это программы, которые могут собирать личную информацию о пользователе и отпривать ее на удаленный сервер без согласия пользователя.
* Ransomware: это вид вредоносного ПО, который зашифровывает файлы на компьютере и требует выкуп для их расшифровки.

Именно из-за разнообразности видов вредоносного ПО и их методов распространения существует ряд технологий и методов для защиты компьютеров и сетей. Тестирование безопасности - это процесс идентификации и исследования уязвимостей в компьютерной системе или сети, чтобы предотвратить несанкционированный доступ или использование этой системы.

# **Тестирование безопасности (Тестирование на проникновение)**

Тестирование на проникновение является одним из наиболее эффективных способов обнаружения уязвимостей в системе и помочь организациям лучше защитить свои системы и данные. Тестирование на проникновение должно быть проводиться регулярно, как минимум раз в год, чтобы гарантировать, что система защищена от современных угроз.

Важно отметить, что тестирование на проникновение должно проводиться с получением согласия владельца системы и в рамках юридически допустимых ограничений, для того чтобы избежать неожиданных проблем и нарушений закона.

# **Тестовая документация (Ее виды и назначение, Тест план)**

Тестовая документация включает в себя различные документы, которые используются в процессе тестирования программного обеспечения. Некоторые из наиболее распространенных видов тестовой документации это:

* Тест план: Этот документ описывает цели, объекты, процедуры и риски тестирования, а также описывает тест-кейсы и сроки их выполнения.
* Тест-кейсы: Этот документ содержит детализированное описание тест-кейсов, которые будут использоваться для проверки функциональности программного обеспечения.

Отчет о тестировании: Этот документ содержит информацию о результатах тестирования, включая найденные ошибки и рекомендации по устранению.

Назначение тестовой документации - это предоставление детальной и

надежной информации о тестируемом программном обеспечении и о процессе тестирования. Это позволяет стабилизировать и стандартизировать процесс тестирования, а также обеспечивает возможность оценки качества тестирования и качества самого программного обеспечения. Также иметь документированную информацию о тестировании может быть полезно для доказательства соответствия и при возникновении спорных вопросов.

# **Тестовая документация (График тестирования, Матрица трассируемости, Тестовый набор, Чек-лист)**

*График тестирования:* Этот документ описывает календарный план тестирования, включая даты начала и окончания тестирования, а также ответственных за его проведение и контроль за выполнением плана.

*Матрица трассируемости:* Этот документ отображает соответствие между требованиями, тест-кейсами и отчетами о тестировании, дает возможность следить за тем, как требования протестированы и какие ошибки были найдены.

*Тестовый набор:* Этот документ содержит информацию о всех используемых в тестировании данных, инструментах и конфигурациях.

*Чек-лист:* Этот документ содержит список проверок, которые должны быть выполнен , до начала тестирования и/или после завершения тестирования. Чек-лист может включать в себя такие проверки как: проверка конфигурации тестовой среды, проверка доступности всех необходимых инструментов и данных для тестирования, проверка соответствия тестовых данных требованиям и т.д. Этот документ помогает сотрудникам проводящим тестирование не забыть выполнить важные проверки и подготовиться к тестированию.

# **Техники тест дизайна (Что это, Методы черного ящика: классы эквивалентности, Анализ граничных значений)**

Техники тест дизайна - это совокупность методов и процедур, которые используются для планирования, создания и исполнения тестов программного обеспечения. Они помогают разработчикам тестов определять, какие тесты необходимо написать и как их исполнять, чтобы максимально эффективно проверить функциональность программного обеспечения.

Методы черного ящика - это техники, которые используются для тестирования без доступа к исходному коду или к деталям реализации программного обеспечения. Они основаны на использовании входных данных и выходных данных для проверки функциональности.

Классы эквивалентности – один из методов который используется для тестирования, он позволяет группировать входные данные в несколько классов эквивалентности, используя критерии, такие как значения, типы или диапазоны. Это позволяет разработчикам тестов разделить входные данные на группы и разработать тест-кейсы для каждой группы.

Анализ граничных значений - это техника, которая используется для тестирования программного обеспечения на граничных значениях входных данных. Это позволяет найти ошибки, которые могут возникнуть при использовании минимальных или максимальных значений входных данных, или при использовании значений, которые находятся на границе допустимого диапазона. Так же он позволяет найти ошибки связанные с переполнением или недостаточным количеством ресурсов.

Эти техники дизайна теста помогают разработчикам тестов планировать, создавать и исполнять тесты, которые максимально эффективно проверяют функциональность программного обеспечения, и обеспечивают достоверность результатов тестирования.

# **Техники тест дизайна (Методы черного ящика: таблица решений, таблица (диаграмма) переходов)**

Методы черного ящика - это техники, которые используются для тестирования без доступа к исходному коду или к деталям реализации программного обеспечения.



Таблица решений (Decision Table Testing): это техника, которая используется для тестирования программного обеспечения, которое основано на логических вычислениях и принимает решения на основе различных условий. Тестировщик использует таблицу, содержащую ряд условий и соответствующих действий, которые должны быть выполнены в зависимости от условий, и использует эту таблицу для создания тест-кейсов. Таблица решений помогает тестировщику визуализировать все возможные комбинации условий и действий, что позволяет находить ошибки, которые могут произойти при определенных комбинациях условий

.

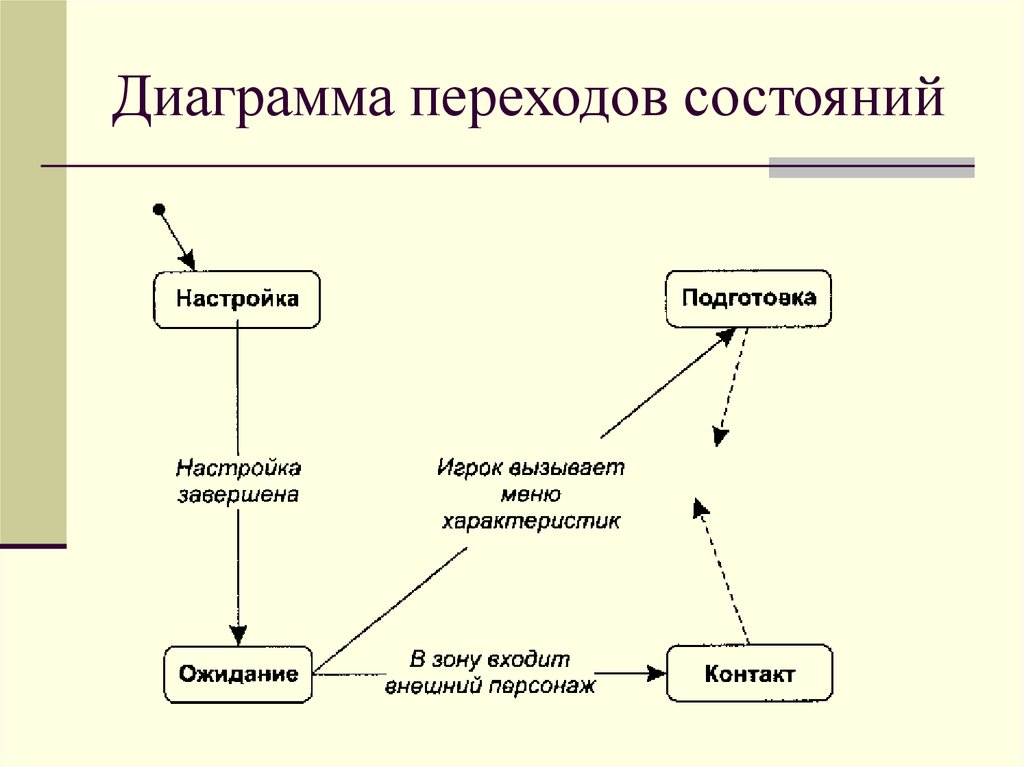
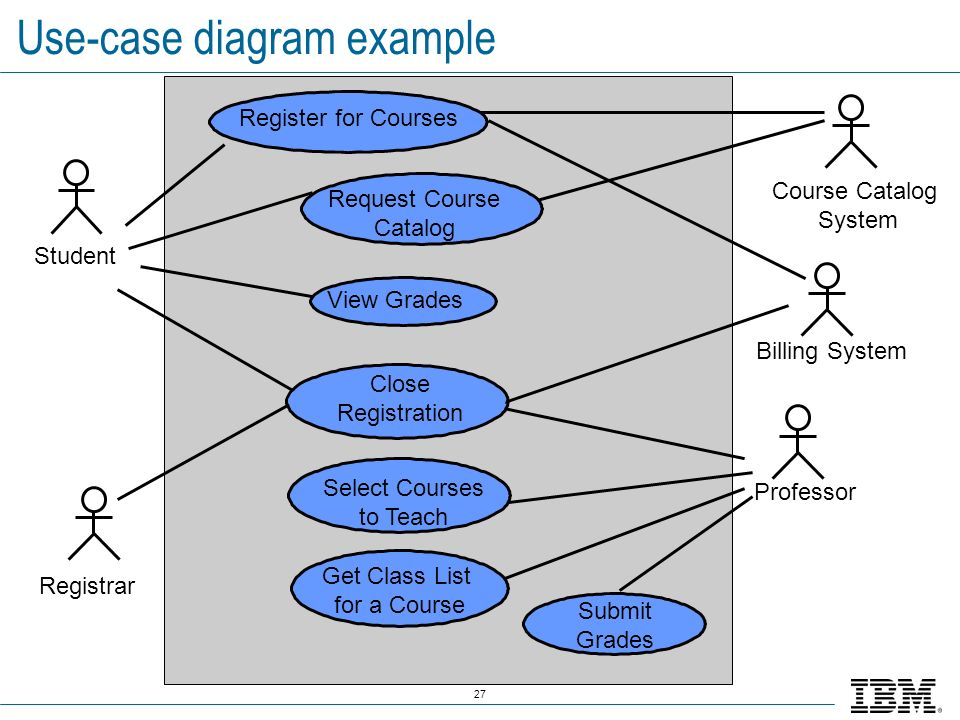


Таблица (диаграмма) переходов (Transition Table/State Transition Testing): это техника, которая используется для тестирования программного обеспечения, которое основано на состояниях и переходах между состояниями. Тестировщик использует таблицу или диаграмму, которая отображает все возможные состояния и переходы между ними, и использует эту таблицу для создания тест-кейсов, которые проверяют, что программное обеспечение переходит из одного состояния в другое корректно в зависимости от входных данных и событий. Таблица переходов или диаграмма переходов помогают тестировщику визуализировать все возможные переходы между состояниями и находить ошибки, которые могут произойти при некорректных переходах.

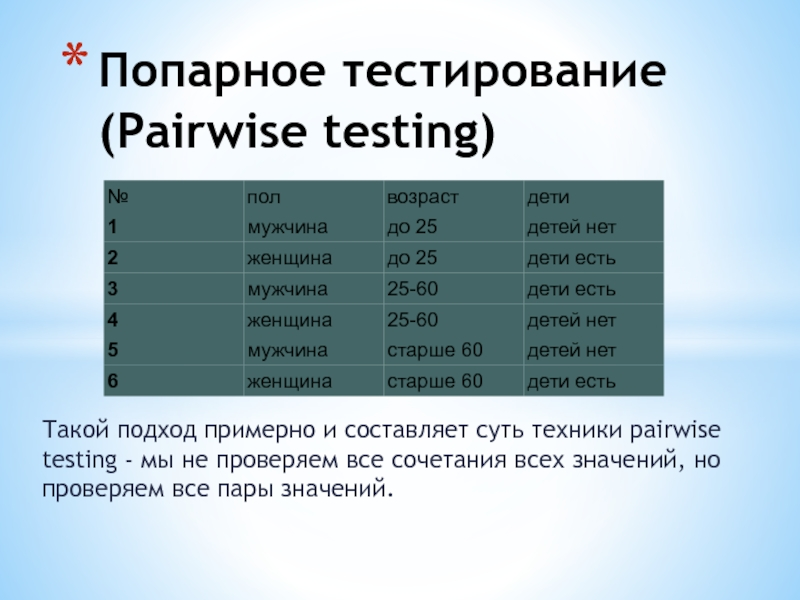
Техники тест-дизайна включают множество различных методов и техник, как методы черного ящика так и методы белого ящика, которые помогают разработчикам тестов планировать, создавать и исполнять тесты, которые максимально эффективно проверяют функциональность программного обеспечения и обеспечивают достоверность результатов тестирования.

# **Техники тест дизайна (Методы черного ящика: тестирование по вариантам использования, попарное тестирование)**

"Черный ящик" тест дизайна - это метод тестирования продукта, в котором особое внимание уделяется его внешнему виду и поведению, а не его внутреннему действию.



"Тестирование по вариантам использования" (use case testing) - это метод, который заключается в тестировании системы в соответствии с различными сценариями использования, которые могут возникнуть в реальной жизни.



"Попарное тестирование" (pairwise testing) - это метод, который заключается в тестировании всех возможных комбинаций входных значений для двух или более параметров, которые могут влиять на работу системы.

# **Техники тест дизайна (Методы белого ящика: тестирование покрытия операторов, тестирование покрытия ветвей. Методы, основанные на опыте)**

"Белый ящик" тест дизайна - это метод тестирования, в котором особое внимание уделяется внутреннему действию продукта, а не его внешнему виду и поведению.

"Тестирование покрытия операторов" (operator coverage testing) - это метод, который заключается в тестировании каждого оператора в коде продукта как минимум один раз.

"Тестирование покрытия ветвей" (branch coverage testing) - это метод, который заключается в тестировании каждой ветви в коде продукта как минимум один раз.

"Методы, основанные на опыте" тест дизайна - это метод тестирования, который заключается в использовании опыта и знаний тестировщика, чтобы предсказать и идентифицировать возможные проблемы в продукте.

# 

# **Тестовые случаи (Цели, Жизненный цикл, Атрибуты тестовых случаев)**

"Тестовые случаи" - это описания того, как тестировать продукт и какие результаты должны быть достигнуты в каждом тесте.

Цели тестовых случаев - это то, что следует достичь с помощью каждого теста. Цели тестовых случаев могут включать в себя проверку того, что продукт работает в соответствии с требованиями, проверку на отсутствие ошибок или проверку его совместимости с другими системами.

Жизненный цикл тестовых случаев - это серия этапов, которые проходит каждый тестовый случай от его создания до окончания его использования. Жизненный цикл тестовых случаев может включать в себя этапы, такие как планирование, создание, выполнение и отчетность.

Атрибуты тестовых случаев - это характеристики, которые описывают тестовый случай и которые помогают идентифицировать его и организовать. Некоторые примеры атрибутов тестовых случаев могут включать в себя название, описание, уровень сложности, идентификатор, категорию, а также связи с другими тестовыми случаями или требованиями.

# **Тестовые случаи (Свойства качественных тестовых случаев, типичные ошибки при создании тестовых случаев)**

"Качественные тестовые случаи" - это тестовые случаи, которые хорошо продуманы и покрывают важные аспекты продукта.

Свойства качественных тестовых случаев включают в себя:

* Ясность и доступность: они должны быть легко читаемыми и понятными;
* Релевантность: они должны соответствовать целям и требованиям продукта;
* Комплексность: они должны покрывать различные аспекты и функциональность продукта;
* Репрезентативность: они должны представлять собой реалистичные и предсказуемые сценарии использования;
* Масштабируемость: они должны быть готовы к изменениям и расширениям продукта.

Типичные ошибки при создании тестовых случаев включают в себя:

* Пропуск важных сценариев использования или требований;
* Слишком много дубликации или ненужных тестов;
* Недостаточная детализация тестового случая или тестовых шагов;
* Недостаточная проверка или отсутствие ожидаемых результатов;
* Сложный или неоднозначный формат тестового случая, сложно его выполнить и понять;

Эти ошибки могут привести к тому, что тесты не смогут достоверно проверить качество продукта или их будет сложно выполнить и интерпретировать, поэтому важно создавать тестовые случаи с достаточной детализацией, чтобы они были ясными и легко воспроизводимыми. Так же важно иметь представление о жизненном цикле тестов и использовать соответствующие методологии, чтобы избежать типичных ошибок.

# **Тестовые сценарии (Цели, Шаги для создания, Атрибуты тестовых сценариев)**

"Тестовые сценарии" - это последовательности действий и входных данных, которые используются для тестирования функциональности продукта. Они описывают конкретный сценарий использования и ожидаемое поведение продукта в ответ на этот сценарий.

Цели тестовых сценариев - это то, что следует достичь с помощью каждого теста. Цели тестовых сценариев могут включать в себя проверку того, что продукт работает в соответствии с требованиями, проверку на отсутствие ошибок или проверку его совместимости с другими системами.

Шаги для создания тестовых сценариев - это процесс создания тестовых сценариев, который может включать в себя следующие этапы:

* Идентификация требований: определение требований, которые должны быть протестированы;
* Планирование тестирования: определение как и когда будут выполняться тесты;
* Создание тестовых сценариев: написание детализированных сценариев, которые описывают действия и ожидаемый результат;
* Ревизия и одобрение: проверка тестовых сценариев на качество и одобрение их для использования.

Атрибуты тестовых сценариев - это данные, которые описывают тестовый сценарий и которые помогают идентифицировать его и организовать. Некоторые примеры атрибутов тестовых сценариев могут включать в себя название, описание, уровень сложности, идентификатор, категорию, связи с другими тестовыми сценариями или требованиями.

# **Тест план, его назначение и структура**

"Тест-план" - это документ, который описывает подходы, методы и процессы, которые будут использоваться для тестирования продукта.

Назначение тест-плана:

* Определение области тестирования, которая должна быть покрыта;
* Планирование ресурсов и времени, необходимых для тестирования;
* Согласование ожидаемого результата тестирования с командой разработки и заказчиком;
* Определение ответственности за различные этапы тестирования;
* Описание процедур отчетности и управления ошибками.

Структура тест-плана может включать следующие разделы:

* Введение: описание целей и области тестирования;
* Область тестирования: описание функциональности, которая будет протестирована;
* План тестирования: описание методов, процессов и ресурсов, которые будут использованы для тестирования;
* Роли и ответственности: описание ответственности за различные этапы тестирования;
* Отчетность и управление ошибками: описание процедур отчетности и управления ошибками, которые будут использоваться во время тестирования.

Тест-план может быть обновлен и изменен в зависимости от проекта и его этапа разработки. Важно обновлять и делиться тест-планом с командой разработки и заказчиком, чтобы все были в курсе планируемых тестовых действий и достигаемых целей. В итоге, тест-план должен быть использован как инструмент управления тестированием, который поможет сохранять фокус и обеспечивать точность процесса тестирования.

# **Отчеты об ошибках (Цель, Жизненный цикл, Атрибуты)**

Отчеты об ошибках - это документы, которые используются для отслеживания и анализа ошибок в программном обеспечении.

Цель использования отчетов об ошибках - обнаружение и исправление ошибок в коде, чтобы обеспечить качество и стабильность программного обеспечения.

Жизненный цикл отчета об ошибке включает в себя следующие этапы: обнаружение ошибки, отправка отчета об ошибке, анализ ошибки, исправление ошибки, тестирование и закрытие отчета.

Атрибуты отчета об ошибке могут включать: ID ошибки, описание ошибки, детали ошибки (код, скриншоты), степень критичности ошибки, имя того, кто отправил отчет об ошибке, дату и время обнаружения ошибки.

# **Отчеты об ошибках (Свойства качественных отчетов об ошибках, Типичные ошибки при написании)**

Свойства качественных отчетов об ошибках включают:

* Полноту: отчет должен быть достаточно детальным и содержать всю необходимую информацию для идентификации и исправления ошибки.
* Ясность: отчет должен быть понятным и легко читаемым для того, кто будет его анализировать и исправлять.
* Конкретность: отчет должен содержать конкретные детали ошибки, такие как код, скриншоты и сообщения об ошибках.

Типичные ошибки, которые могут быть сделаны при написании отчета об ошибке, включают:

* Недостаточно информации: не предоставление достаточной информации для идентификации и исправления ошибки
* Неоднозначность: использование неоднозначной или непонятной терминологии
* Неконкретность: отсутствие конкретных деталей ошибки, таких как код, скриншоты и сообщения об ошибках
* Необоснованность: не дание достаточных обоснований для заявленной ошибки
* Неправильное форматирование: неправильное форматирование или оформление отчета, которое может затруднять чтение и анализ
* Неактуальность: предоставление неактуальной информации или отчет об ошибке, которая уже была исправлена.

Это основные ошибки, которые могут быть сделаны при написании отчета об ошибке, но не являются полным списком. Важно, чтобы отчеты были полными, ясными и конкретными, чтобы они могли быть легко анализированы и исправлены.

# **Отчеты о результатах тестирования**

Отчеты о результатах тестирования - это документы, которые используются для отслеживания и документирования результатов тестирования программного обеспечения. Они могут содержать информацию о том, какие тесты были запущены, какие результаты были получены, и какие дефекты были обнаружены.

Свойства качественных отчетов о результатах тестирования включают:

* Полноту: отчет должен содержать всю необходимую информацию о тестировании, включая информацию о тестируемом ПО, тест-кейсах, покрытии тестами и результатах.
* Ясность: отчет должен быть понятным и легко читаемым для того, кто будет его анализировать.
* Конкретность: отчет должен содержать конкретные детали результатов тестирования, такие как кали ошибок, процент прохождения тестов и информацию о тестовых данных.
* Предоставление контекста: отчет должен содержать дополнительную информацию, такую как контекст проекта или информацию о проведенных изменениях, чтобы лучше понять результаты тестирования.
* Обновление: отчет должен быть актуальным и обновляться по мере проведения тестирования.

Типичные ошибки, которые могут быть сделаны при составлении отчета о результатах тестирования, включают:

* Недостаточное описание методологии тестирования
* Отсутствие статистической обработки данных
* Недостаточное обоснование выводов
* Отсутствие представления данных в графическом виде
* Представление недостоверных данных
* Недостаточное объяснение потенциальных причин ошибок и их влияния на результаты тестирования.

# 

# **Автоматизация тестирования (Виды, Модульное тестирование)**

Автоматизация тестирования - это использование специальных программных средств для автоматизации процесса тестирования. В этом случае, тестирование происходит не с помощью ручного ввода данных, а с помощью специально написанного кода.

Виды автоматизации тестирования:

* Тестирование интеграции
* Тестирование системы
* Тестирование приемочной процедуры
* Регрессионное тестирование

Модульное тестирование - это метод автоматизированного тестирования, который заключается в том, что система разбивается на несколько модулей, каждый из которых тестируется независимо. Этот метод позволяет быстро и эффективно идентифицировать и исправить ошибки, поскольку он обеспечивает детальное тестирование каждого модуля системы, а также позволяет оценить риски и узнать как злоумышленник может использовать уязвимости для атаки. Это помогает разработчикам и специалистам по информационной безопасности лучше понимать и улучшать защиту системы.

# 

# **Автоматизация тестирования (Интеграционные тесты, UI тестирование)**

Автоматизация тестирования может включать в себя различные виды тестирования, такие как интеграционные тесты и тестирование пользовательского интерфейса (UI).

Интеграционное тестирование - это метод проверки работоспособности системы как единого целого, путем проверки взаимодействия между различными компонентами системы. Это позволяет обнаружить ошибки, которые могут возникнуть из-за неправильного взаимодействия компонентов.

Тестирование пользовательского интерфейса (UI) - это метод проверки работоспособности интерфейса пользователя, такой как кнопки, меню, формы ввода и т.д. Автоматизированное тестирование UI позволяет проверить функциональность интерфейса, убедиться в его совместимости и простоте использования. Это позволяет убедиться, что интерфейс прост для использования и доступен для всех пользователей, включая тех, кто имеет ограниченные возможности. Автоматизированное UI тестирование может быть написано для различных платформ и операционных систем, что обеспечивает совместимость с различными устройствами и браузерами.