

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО МГТУ «СТАНКИН»)**

|  |  |
| --- | --- |
| **Институт**  информационных технологий | **Кафедра**  информационных систем |

**Реферат № 7**

по дисциплине «**Основы программной инженерии**»

на тему: «Тестирование программного обеспечения»

|  |  |  |
| --- | --- | --- |
| **Студент** группа ИДБ–22–06 |  | **Мустафаева П.М.** |
|  | подпись |  |
| **Руководитель**  к.э.н., доцент |  | **Ахмедов Э.Р.** |
|  | подпись |  |

**1. Какова структура процесса разработки ПП с точки зрения тестирования? Поясните каждый этап.**

Процесс разработки ПП с точки зрения тестирования выглядит следующим образом (рис. 1). Процесс тестирования в разработке ПО начинается с взаимодействия между отделами, обеспечивающего четкое понимание требований и функциональности будущего продукта. Отдел разработки бизнес-требований передает группе тестирования и отделу разработки два ключевых документа: описание бизнес-логики продукта и функционального дизайна.

Бизнес-логика продукта описывает процессы, которые система должна обрабатывать, с точки зрения конечного пользователя. Этот документ фокусируется на том, что должен делать продукт, а не на том, как он это реализует. Например, если разрабатывается интернет-магазин, бизнес-логика может включать описание процесса оформления заказа, оплаты и доставки товаров.

Функциональный дизайн дополняет бизнес-логику, детализируя функции, которые должен выполнять продукт. В нем также могут быть описаны пользовательские интерфейсы и требования к ним. Например, для того же интернет-магазина в функциональном дизайне может быть указано, что система должна поддерживать фильтрацию товаров по категориям, поиск по названию и сортировку по цене.

После получения этих документов аналитики отдела разработки проводят их детальный анализ и формируют технический дизайн, который включает описание архитектуры системы, технические задания и другие инженерные аспекты. Этот этап критически важен, так как он переводит бизнес-требования в технически реализуемую форму. Например, технический дизайн может определять, будет ли система использовать монолитную или микросервисную архитектуру, какие базы данных и API будут задействованы.

Сформированные документы технического дизайна отправляются на согласование обратно в отдел разработки бизнес-требований и группе тестирования. После их утверждения группа тестирования приступает к созданию тест-плана, тест-сценариев и тест-кейсов.

Тест-план определяет стратегию тестирования, включая цели, объем работ, ресурсы, сроки и критерии приемки. Тест-сценарии описывают последовательности действий для проверки различных аспектов системы, а тест-кейсы детализируют конкретные шаги, ожидаемые результаты и условия выполнения тестов.

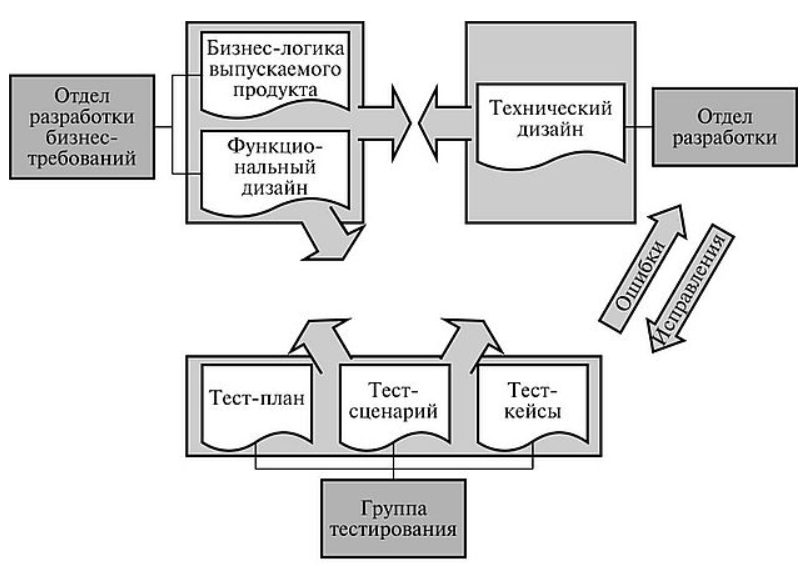


Рис. 1 Структура процесса разработки ПП

**2. Какая основа для тестирования продукта закладывается на начальном  
этапе?**

Первым и наиболее важным документом является тест-план, который служит руководством для всей команды тестирования. В нем описывается объект тестирования, то есть какие компоненты системы будут подвергаться проверке. Также в тест-плане фиксируется стратегия тестирования, включающая методы и виды проверок, такие как функциональное, интеграционное или нагрузочное тестирование. Документ содержит расписание работ, определяющее сроки выполнения этапов, и критерии, которые указывают, когда тестирование можно начинать и когда оно считается завершенным. Кроме того, в тест-плане учитываются необходимые ресурсы, такие как оборудование, программное обеспечение и специальные знания команды. Важной частью является оценка потенциальных рисков и способы их минимизации, что позволяет заранее предусмотреть возможные проблемы и пути их решения.

На следующем уровне детализации создаются тестовые сценарии, которые описывают, как именно должна проверяться система. В них указываются начальные условия, например, состояние приложения перед выполнением теста, входные данные, которые будут использоваться, и последовательность действий пользователя или автоматизированного скрипта. Каждый сценарий завершается ожидаемым результатом, что позволяет четко определить, соответствует ли поведение системы требованиям.

Для еще более точной проверки разрабатываются тест-кейсы, представляющие собой пошаговые инструкции для тестирования отдельных функций или их частей. В них прописываются конкретные условия выполнения, параметры и шаги, которые необходимо выполнить, чтобы убедиться в корректности работы системы. Тест-кейсы обеспечивают воспроизводимость проверок и помогают избежать пропуска критических сценариев.

**3. Что такое «дымовое» тестирование?**

Дымовое тестирование – это первичная проверка программы, позволяющая убедиться, что основные функции работают, а приложение запускается без ошибок.

Такой тест сразу выявляет критические неполадки, которые могут сделать дальнейшую проверку бессмысленной. Его обычно проводят после каждой сборки, чтобы убедиться, что продукт готов к следующему этапу тестирования.

Цель дымового тестирования – выявить любые серьёзные проблемы, которые могут помешать дальнейшему тестированию приложения, а не определить все дефекты в программном обеспечении.

Некоторые аспекты, которые тестируют на этом этапе:

* Запуск приложения. Оно должно загружаться без ошибок, а интерфейс – открываться без сбоев.
* Работа ключевых функций. Например, регистрация, вход в аккаунт, открытие страниц или отправка форм должны работать корректно.
* Навигация и ссылки. Переключение между разделами и работа всех ссылок должны быть безупречными.
* Обработка данных. Проверяют, правильно ли вводятся данные в поля и корректно ли они отображаются на экране.
* Уведомления и ошибки. Все сообщения должны быть понятными, чтобы пользователи знали, как действовать при сбое.
* Кросс-браузерность и совместимость. Важно протестировать приложение на разных браузерах и устройствах, чтобы избежать проблем с отображением.

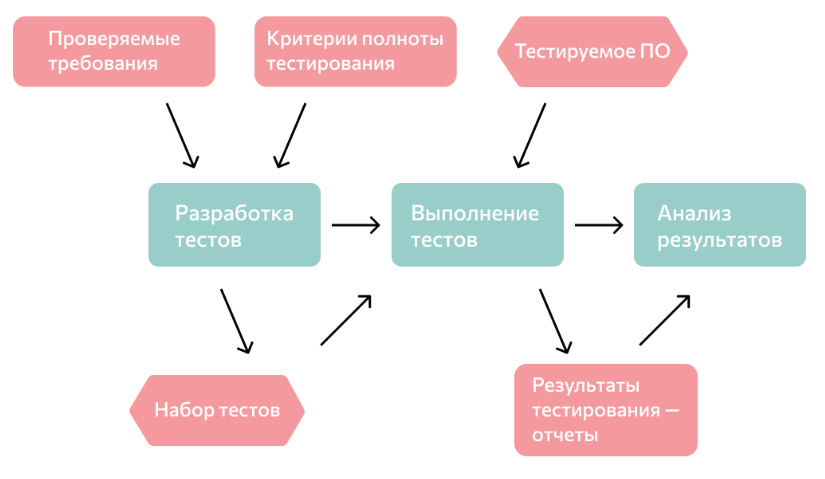


Рис. 2 Общий алгоритм «дымового» тестирования

**4. Что такое позитивное тестирование?**

Позитивное тестирование – использование только корректных данных и сценариев для проверки работоспособности приложения.

Позитивное тестирование направлено на проверку корректной работы приложения при использовании правильных и ожидаемых данных. Оно имитирует действия конечного пользователя, который использует продукт согласно инструкции. Цель такого тестирования – убедиться, что приложение выполняет свои функции в соответствии с требованиями и спецификациями.

В процессе тестирования специалист внимательно изучает требования и рекомендации по использованию приложения. Так, при проверке формы регистрации в приложении он вводит в неё верные данные и активирует кнопку «Зарегистрироваться».

Примеры позитивного тестирования:

* Ввод корректных данных в форму регистрации;
* Переход по ссылкам, которые должны вести на нужные страницы;
* Выполнение операций, предусмотренных функционалом приложения.

При проведении тестирования сначала всегда выполняются позитивные тесты. Это необходимо, чтобы убедиться, что основные функции приложения работают правильно. И только после этого можно переходить к негативным проверкам, которых обычно бывает гораздо больше, чем позитивных.

**5. Что такое негативное тестирование?**

Негативное тестирование, или тестирование на ошибочные случаи, является важной составляющей процесса обеспечения качества программного обеспечения. Оно направлено на проверку того, как система реагирует на некорректные, невалидные или неожиданные данные, что позволяет выявить уязвимости и улучшить надежность продукта. В отличие от позитивного тестирования, которое проверяет систему на корректное выполнение при правильных входных данных, негативное тестирование исследует сценарии, в которых пользователь или внешний фактор вводит данные, отклоняющиеся от нормы.

Главная цель негативного тестирования – убедиться, что система правильно обрабатывает некорректные данные и сохраняет свою стабильность при любых вводных. Это помогает предотвратить потенциальные ошибки, которые могут возникнуть в реальных условиях эксплуатации, где пользователи могут допускать ошибки или вводить данные, которые не соответствуют ожиданиям системы.

Негативное тестирование позволяет:

* Проверить, что система корректно сообщает об ошибках при вводе некорректных данных.
* Оценить устойчивость системы при экстремальных условиях или непредсказуемом поведении.
* Обнаружить уязвимости, которые могут привести к сбоям или критическим ошибкам.

**6. Дайте определение функциональному, нагрузочному, стресс-тестированию и тестированию стабильности.**

Функциональное тестирование представляет собой процесс проверки соответствия программного обеспечения заданным требованиям и спецификациям. Оно направлено на верификацию того, что система корректно выполняет свои функции в соответствии с ожидаемым поведением. Тестировщики проверяют работу отдельных модулей, интеграцию компонентов и систему в целом, используя различные входные данные и сравнивая полученные результаты с ожидаемыми. Основная цель – убедиться, что продукт работает так, как было задумано на этапе проектирования.

Нагрузочное тестирование используется для оценки производительности системы под определенной нагрузкой. Оно имитирует реальные условия эксплуатации, когда множество пользователей одновременно взаимодействуют с приложением. В ходе тестирования измеряются такие показатели, как время отклика, пропускная способность и использование ресурсов (процессор, память, сеть). Это позволяет выявить «узкие места» в архитектуре и определить максимальную рабочую нагрузку, которую система может выдержать без потери производительности.

Стресс-тестирование выходит за рамки обычной нагрузки и проверяет поведение системы в экстремальных условиях. Оно предполагает создание ситуации, когда нагрузка превышает нормальные рабочие параметры, чтобы выяснить, как система справляется с перегрузками, насколько устойчива к отказам и как восстанавливается после них. Например, тестирование может включать моделирование резкого увеличения числа пользователей или нехватки системных ресурсов. Цель – убедиться, что приложение не только не теряет данные в критических ситуациях, но и корректно завершает работу или переходит в безопасный режим.

Тестирование стабильности (или надежности) направлено на проверку работы системы в течение длительного времени под средней или высокой нагрузкой. Оно помогает выявить такие проблемы, как утечки памяти, постепенное замедление работы или накопление ошибок, которые могут проявиться только после продолжительной эксплуатации. Тестирование проводится в условиях, максимально приближенных к реальным, чтобы гарантировать, что приложение будет стабильно работать не только в первые часы, но и дни или недели непрерывного использования.

**7. В чем заключается проверка эргономичности?**

Проверка эргономичности (юзабилити-тестирование) заключается в определении удобства искусственного объекта (например, веб-страницы, пользовательского интерфейса или устройства) для его предполагаемого применения.

Процесс проверки основан на привлечении пользователей в качестве тестировщиков и испытателей. Им предлагают в «лабораторных» условиях решить основные задачи, для выполнения которых продукт разрабатывался, и просят высказывать во время выполнения тестов свои замечания.

Процесс тестирования фиксируется в протоколе (логе) и/или на аудио- и видеоустройства – с целью последующего более детального анализа.

Ещё один подход к проверке эргономичности заключается в разработке «идеального» сценария решения задачи, предложенной пользователю. При выполнении задачи пользователями регистрируются их отклонения от задуманного сценария для последующего анализа.

Если проверка эргономичности выявляет какие-либо трудности (например, сложности в понимании инструкций, выполнении действий или интерпретации ответов системы), то разработчики должны доработать продукт и повторить тестирование.

**8. В чем цель тестирования безопасности?**

Цель тестирования безопасности заключается в выявлении уязвимостей и слабых мест программного обеспечения, которые могут быть использованы злоумышленниками для несанкционированного доступа, нарушения конфиденциальности или целостности данных. Этот вид тестирования помогает оценить, насколько система защищена от потенциальных атак, таких как SQL-инъекции, межсайтовый скриптинг (XSS), подбор паролей или отказ в обслуживании (DDoS).

Тестирование безопасности также направлено на проверку соответствия системы стандартам и нормативным требованиям, таким как GDPR, PCI DSS или ISO 27001. Оно включает анализ механизмов аутентификации и авторизации, шифрования данных, контроля доступа и журналирования событий. Важно убедиться, что даже в случае взлома система минимизирует возможный ущерб и предотвращает утечку критически важной информации.

Кроме того, тестирование безопасности помогает выявить риски, связанные с человеческим фактором, например, использование слабых паролей или недостаточную осведомленность пользователей о фишинговых атаках. В конечном итоге, его цель – обеспечить защиту не только данных, но и репутации компании, поскольку утечка информации или частые кибератаки могут привести к потере доверия клиентов и финансовым потерям.

**9. Зачем нужна проверка совместимости?**

Проверка совместимости играет критически важную роль в процессе разработки и внедрения программного обеспечения, поскольку современные цифровые экосистемы отличаются чрезвычайным разнообразием компонентов. В реальных условиях пользователи могут запускать приложение на разных аппаратных конфигурациях, под управлением различных операционных систем, с разнообразными периферийными устройствами и в неодинаковых сетевых средах. Без тщательной проверки совместимости велик риск столкнуться с ситуацией, когда программа, идеально работающая в лабораторных условиях, окажется неработоспособной у значительной части целевой аудитории.

Особую актуальность такое тестирование приобретает в условиях массового распространения программных продуктов, когда разработчик не может контролировать среду, в которой будет эксплуатироваться его решение. Например, веб-приложение должно одинаково корректно отображаться в различных браузерах, несмотря на их особенности интерпретации кода, а бизнес-приложение – стабильно работать с разными СУБД, сохраняя целостность данных. Проблемы совместимости могут проявляться на разных уровнях – от графического интерфейса до низкоуровневого взаимодействия с оборудованием.

Кроме того, проверка совместимости помогает выявить потенциальные конфликты с системным ПО, таким как антивирусы или межсетевые экраны, которые могут блокировать критически важные функции приложения. Это особенно важно для корпоративных решений, где программное обеспечение должно интегрироваться в сложную ИТ-инфраструктуру предприятия.

**10. В чем достоинства и недостатки автоматизированного тестирования?**

Автоматизированное тестирование стало неотъемлемой частью современной разработки программного обеспечения, предлагая целый ряд значительных преимуществ. Главное достоинство автоматизации заключается в существенном ускорении процесса проверки – автоматические тесты могут выполняться значительно быстрее ручных, особенно при необходимости частых повторных прогонов, таких как регрессионное тестирование. Это позволяет выявлять ошибки на ранних этапах, снижая стоимость их исправления. Автоматизация обеспечивает высокую точность выполнения тестов, исключая человеческий фактор, который может привести к ошибкам при ручной проверке. Повторяемость тестов гарантирует, что одни и те же проверки выполняются одинаково каждый раз, что особенно важно для критически важных функций системы.

Однако автоматизированное тестирование имеет и существенные ограничения. Начальные затраты на создание тестового фреймворка и написание тестовых сценариев могут быть весьма высокими, требуя значительных временных и финансовых ресурсов. Автоматизация не всегда оправдана для проверок, которые выполняются однократно или требуют творческого подхода, такого как исследовательское тестирование. Тесты могут становиться хрупкими – даже небольшие изменения в интерфейсе приложения часто требуют корректировки тестовых скриптов. Кроме того, автоматические тесты не способны заменить человеческую интуицию и способность находить неочевидные проблемы, выходящие за рамки строго определенных сценариев.

Важно понимать, что автоматизированное тестирование наиболее эффективно в сочетании с ручными проверками. Автоматизация идеально подходит для повторяющихся, трудоемких задач, в то время как человек лучше справляется с исследовательским тестированием и оценкой пользовательского опыта.

**11. Для чего предназначено модульное тестирование, какие у него достоинства и недостатки?**

Модульное тестирование служит фундаментом процесса проверки программного обеспечения, сосредотачиваясь на изолированной проверке отдельных компонентов системы – функций, методов или классов. Его главное предназначение заключается в верификации корректности работы минимальных логических единиц кода в отрыве от остальных частей приложения. Такой подход позволяет выявлять ошибки на самом раннем этапе разработки, когда стоимость их исправления минимальна. Разработчики получают возможность быстро проверять работоспособность новых или измененных фрагментов кода, не дожидаясь сборки всей системы.

Ключевым достоинством модульного тестирования является его высокая детализация – тесты фокусируются на конкретной функциональности, что упрощает локализацию ошибок. Автоматизированные модульные тесты образуют "безопасную сетку", позволяющую смело рефакторить код, будучи уверенным в сохранении его работоспособности. Они выполняются чрезвычайно быстро, что способствует практике непрерывной интеграции. Модульные тесты также служат живой документацией, наглядно демонстрируя ожидаемое поведение компонентов системы. Их наличие дисциплинирует разработчиков, побуждая создавать более модульный и слабосвязанный код.

Однако модульное тестирование имеет и существенные ограничения. Оно требует значительных временных затрат на написание и поддержку тестов, что может замедлять начальные этапы разработки. Изолированная проверка компонентов не выявляет проблем интеграции между модулями. Тесты могут становиться хрупкими при частых изменениях внутренней реализации, даже если внешнее поведение сохраняется.

**12. В чем заключается системное тестирование?**

Системное тестирование – это процесс, в ходе которого проверяется работоспособность и соответствие требованиям всей системы в целом. Оно выполняется после завершения интеграционного тестирования и перед выпуском продукта на рынок.

Основная цель системного тестирования – удостовериться, что вся система работает корректно и взаимодействует с окружением без каких-либо проблем.

Некоторые аспекты системного тестирования:

* Проверка работы системы в целом. Проверяется взаимодействие всех компонентов системы, их совместимость и плавность работы процессов.
* Тестирование производительности. Оценивается скорость работы системы при различных нагрузках, определяются узкие места.
* Проверка безопасности. Оценивается уровень защищённости системы от внешних угроз и атак.
* Тестирование восстановления после сбоев. Проверяется работоспособность механизмов восстановления и резервирования, а также откат к предыдущей версии системы.
* Тестирование совместимости. Проверяется работоспособность системы на различных платформах и с разным программным обеспечением.

**13. В чем заключается регрессионное тестирование?**

Регрессионное тестирование – это метод тестирования программного обеспечения, который направлен на проверку того, что изменения в коде не привели к возникновению новых ошибок в уже работающих функциях. Этот вид тестирования особенно важен в условиях постоянного обновления и улучшения программного обеспечения. В современном мире, где программные продукты постоянно обновляются и улучшаются, регрессионное тестирование становится неотъемлемой частью процесса разработки.

Регрессионное тестирование помогает убедиться, что новые изменения, исправления ошибок или добавление новых функций не нарушили работу существующих компонентов. Это особенно актуально для крупных проектов, где любое изменение может иметь непредсказуемые последствия. Важно понимать, что даже незначительные изменения в коде могут привести к неожиданным сбоям в других частях системы, поэтому регулярное проведение регрессионного тестирования помогает минимизировать такие риски.

Регрессионное тестирование необходимо для обеспечения стабильности и качества программного обеспечения. Вот несколько причин, почему оно важно:

* Обеспечение стабильности: Регрессионное тестирование помогает убедиться, что новые изменения не нарушили работу существующих функций. Это особенно важно в условиях, когда программное обеспечение используется большим количеством пользователей, и любая ошибка может привести к серьезным последствиям.
* Снижение рисков: Оно позволяет выявить и устранить ошибки, которые могли быть внесены в код в результате изменений. Регулярное проведение регрессионного тестирования помогает минимизировать риски возникновения критических ошибок, которые могут повлиять на работу всего приложения.
* Экономия времени и ресурсов: Автоматизация регрессионного тестирования позволяет сократить время на ручное тестирование и сосредоточиться на более сложных задачах. Это особенно актуально для крупных проектов, где ручное тестирование может занимать значительное количество времени и ресурсов.
* Повышение доверия: Регулярное проведение регрессионного тестирования повышает доверие к качеству продукта как со стороны команды разработчиков, так и со стороны пользователей. Пользователи могут быть уверены, что новые обновления не приведут к возникновению новых проблем, а разработчики могут сосредоточиться на создании новых функций, не беспокоясь о стабильности существующих.

**14. Что такое «тест-кейс»? Опишите структуру «тест-кейса».**

Тест-кейс – это форма записи проверки, которую проводит тестировщик. По сути, это алгоритм действий, по которому предполагается тестировать уже написанную программу. В нём подробно прописаны шаги, которые нужно сделать для подготовки к тесту, сама проверка и ожидаемый результат.

Тестировщик выполняет тест-кейс последовательно, шаг за шагом. Если фактический результат соответствует ожидаемому – всё хорошо. Если нет, тестировщик анализирует, что произошло. Это может быть ошибка в программе, в тест-кейсе из-за его неактуальности или в тестовом стенде. Если дело в программе, инженер составляет отчёт об ошибке и отправляет его разработчикам. Если в тест-кейсе – исправляет сам. Если в стенде – обращается к техническим специалистам.

Как правило, один тест-кейс описывает небольшую последовательность действий с одним конкретным результатом.

Структура тест-кейса:

* **Идентификатор тест-кейса:** Каждый тест-кейс должен иметь уникальный идентификатор, который позволяет легко его найти и ссылаться на него. Обычно идентификатор состоит из буквенно-цифрового кода, например, TC-001. Этот идентификатор помогает организовать и систематизировать тест-кейсы, особенно в больших проектах, где количество тест-кейсов может исчисляться сотнями или даже тысячами.
* **Название тест-кейса:** Название должно быть кратким и описательным, чтобы сразу было понятно, что проверяется. Например, "Проверка авторизации пользователя". Хорошее название тест-кейса помогает быстро понять его суть без необходимости вчитываться в детали. Это особенно полезно, когда нужно быстро найти нужный тест-кейс в большом списке.
* **Описание:** Описание предоставляет более детальную информацию о том, что именно проверяется. Это помогает тестировщикам и другим участникам проекта понять цель тест-кейса. Описание должно быть достаточно подробным, чтобы любой человек, читающий его, мог понять, что именно проверяется и почему это важно. Например, "Этот тест-кейс проверяет возможность авторизации пользователя с корректными учетными данными, чтобы убедиться, что система правильно обрабатывает процесс входа."
* **Предусловия:** Предусловия описывают начальное состояние системы перед выполнением теста. Например, "Пользователь должен быть зарегистрирован в системе". Предусловия помогают установить контекст для выполнения тест-кейса и гарантируют, что все необходимые условия выполнены перед началом тестирования. Это может включать в себя настройки системы, данные пользователя или другие факторы, которые могут повлиять на результат теста.
* **Шаги выполнения:** Это ключевая часть тест-кейса, включающая последовательность действий, которые необходимо выполнить для проверки функционала. Каждый шаг должен быть четко описан, чтобы исключить двусмысленность. Например, "1. Открыть страницу авторизации. 2. Ввести корректный логин и пароль. 3. Нажать кнопку 'Войти'." Четкое описание шагов помогает избежать ошибок и недоразумений при выполнении теста.
* **Ожидаемый результат**: Ожидаемый результат описывает, что должно произойти после выполнения каждого шага. Это позволяет сравнить фактический результат с ожидаемым и определить, прошел ли тест-кейс. Например, "Пользователь успешно авторизуется и попадает на главную страницу." Ожидаемый результат должен быть конкретным и измеримым, чтобы можно было однозначно определить, прошел ли тест.
* **Фактический результат**: Фактический результат записывается после выполнения тест-кейса. Если он совпадает с ожидаемым результатом, тест-кейс считается пройденным. Фактический результат помогает документировать, что именно произошло во время тестирования и позволяет легко выявить расхождения с ожидаемым результатом. Это важная часть процесса анализа и отчетности.
* **Статус**: Статус указывает на результат выполнения тест-кейса. Обычно используются статусы "Пройден", "Не пройден" и "Заблокирован". Статус помогает быстро оценить общее состояние тестирования и определить, какие тест-кейсы требуют дополнительного внимания. Например, если тест-кейс имеет статус "Не пройден", это может указывать на наличие дефекта, который нужно исправить.
* **Примечания:** Примечания могут включать любую дополнительную информацию, которая может быть полезна для выполнения или анализа тест-кейса. Это может быть информация о специфических настройках системы, дополнительных шагах, которые были выполнены, или любых других факторах, которые могут повлиять на результат теста. Примечания помогают документировать контекст выполнения теста и могут быть полезны для будущих анализов.

**15. Какие результаты могут быть после выполнения «тест-кейса»?**

После выполнения тест-кейса возможны несколько вариантов результатов, каждый из которых дает важную информацию о состоянии тестируемого функционала. Основным и желаемым исходом является успешное прохождение теста, когда фактический результат работы системы полностью соответствует ожидаемому, указанному в тест-кейсе. Это означает, что проверяемая функция ведет себя корректно при заданных входных данных и условиях выполнения.

В случае неудачного прохождения тест-кейса фиксируется расхождение между ожидаемым и фактическим результатом, что свидетельствует о наличии дефекта в системе. Такой результат требует детального анализа: необходимо определить, является ли проблема действительно ошибкой в коде, или же возможно, тест-кейс составлен некорректно и требует пересмотра. Иногда выявляются неочевидные зависимости или крайние случаи, не учтенные при первоначальном проектировании.

Особым случаем является блокировка выполнения тест-кейса, когда тестирование невозможно завершить из-за критической ошибки, препятствующей дальнейшей проверке. Например, система может аварийно завершаться на определенном шаге или ключевые элементы интерфейса оказываются недоступными. Такой результат требует срочного вмешательства, так как блокирует не только текущую проверку, но и потенциально другие связанные тесты.

Иногда тест-кейс может дать неоднозначный результат, когда поведение системы частично соответствует ожиданиям, но содержит незначительные отклонения. В таких случаях важно проанализировать, являются ли эти расхождения допустимыми в рамках текущих требований или указывают на потенциальную проблему. Каждый результат выполнения тест-кейса, независимо от исхода, предоставляет ценную информацию для улучшения качества продукта и совершенствования процесса тестирования.