Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Реферат

на тему: Роль тестирования и обеспечения качества в жизненном цикле разработки ПО

Выполнил(а):
Зиберов Александр Александрович
(Ф.И.О)

Направление
Программная инженерия
_____

направленность (профиль)
_____
_____

Проверил (а):
Воронкин Роман Александрович
(ФИО)
_____
(зачтено/ незачтено)
_____
(дата, подпись)

Ставрополь, 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Программное обеспечение (ПО) занимает ключевое место в различных сферах жизни, от бизнеса и образования до здравоохранения и развлечений. С ростом сложности и масштабов программных систем возрастает необходимость в обеспечении их качества и надежности. Ошибки в ПО могут приводить к серьезным финансовым потерям, нарушению работы критически важных систем и даже к угрозам безопасности. Поэтому роль тестирования и обеспечения качества в жизненном цикле разработки ПО приобретает особую значимость.

В условиях постоянного ускорения разработки и выпуска новых версий ПО важно гарантировать, что продукт будет отвечать высоким стандартам качества. Современные пользователи ожидают от ПО безупречной работы, быстродействия и безопасности. Появление новых технологий, таких как искусственный интеллект, машинное обучение и облачные вычисления, дополнительно усложняет процесс разработки, требуя новых подходов к тестированию и обеспечению качества. Тема тестирования и обеспечения качества особенно важна для современной программной инженерии, так как позволяет минимизировать риски, связанные с дефектами ПО, и повысить общую удовлетворенность пользователей.

В реферате будут рассмотрены основные виды тестирования, включая юнит-тестирование, интеграционное тестирование, нагрузочное тестирование и другие, а также инструменты автоматизации такие как Selenium и другие. Кроме того, будут исследованы практики обеспечения качества, включая контроль качества (QC), управление качеством (QA) и всеобъемлющее управление качеством (TQM). Основные вопросы, которые будут рассмотрены в реферате, включают анализ текущих практик и технологий в области тестирования и обеспечения качества, преимущества и недостатки различных подходов, а также примеры успешных и неуспешных проектов.

1 ВВЕДЕНИЕ В ТЕСТИРОВАНИЕ

1.1 Историческая справка

Первые программные системы разрабатывались для научных исследований и нужд министерств обороны. Тестирование таких продуктов проводилось строго формализовано: фиксировались все тестовые процедуры, тестовые данные и полученные результаты. Этот процесс начинался после завершения кодирования и, как правило, выполнялся тем же персоналом.

В 1960-х годах внимание уделялось «исчерпывающему» тестированию, предполагающему проверку всех путей в коде или всех возможных входных данных [1]. Однако вскоре стало ясно, что полное тестирование ПО невозможно из-за огромного количества возможных входных данных, множества путей выполнения кода и сложности обнаружения проблем в архитектуре и спецификациях. Тем самым, «исчерпывающее» тестирование было признано теоретически невозможным.

В начале 1970-х годов тестирование ПО рассматривалось как "процесс демонстрации корректности продукта" или "подтверждение правильности работы ПО". В теоретической программной инженерии верификация ПО определялась как "доказательство правильности". Однако на практике этот метод оказался слишком трудоемким и недостаточно всеобъемлющим. В результате доказательство правильности было признано неэффективным. Тем не менее, демонстрация правильной работы используется и сегодня, например, в приемочных испытаниях. Во второй половине 1970-х годов тестирование стало рассматриваться как процесс, направленный на поиск ошибок. Успешный тест – это тест, выявляющий ранее неизвестные проблемы. Этот подход, противоположный предыдущему, оказался более продуктивным с точки зрения улучшения качества ПО.

В 1980-е годы концепция тестирования расширилась за счет включения предупреждения дефектов. Проектирование тестов стало рассматриваться как

эффективный метод предупреждения ошибок. В это время стало очевидно, что тестирование должно охватывать весь цикл разработки ПО и быть управляемым процессом. Тестирование включало проверки не только скомпилированной программы, но и требований, кода, архитектуры и самих тестов. Традиционное тестирование, существовавшее до начала 1980-х годов, относилось только к готовой системе (ныне называемое системным тестированием). В дальнейшем тестировщики стали вовлекаться во все этапы жизненного цикла разработки, что позволило раньше выявлять проблемы в требованиях и архитектуре, сокращая сроки и бюджет разработки. В середине 1980-х годов появились первые инструменты автоматизированного тестирования. Первоначально они были простыми и не поддерживали написание сценариев на скриптовых языках.

В начале 1990-х годов тестирование стало включать планирование, проектирование, создание, поддержку и выполнение тестов и тестовых окружений, что привело к переходу от тестирования к обеспечению качества, охватывающего весь цикл разработки ПО. В это время появились разнообразные программные инструменты для поддержки процесса тестирования: более продвинутые среды автоматизации с возможностью создания скриптов и генерации отчетов, системы управления тестами, ПО для нагрузочного тестирования. В середине 1990-х годов, с развитием Интернета и появлением множества веб-приложений, особую популярность получило «гибкое тестирование» (по аналогии с гибкими методологиями разработки ПО).

В начале 2000-х годов развитие автоматизированного тестирования продолжилось, став важной частью практики разработки программного обеспечения. Современные инструменты автоматизированного тестирования предлагают более широкие возможности и интеграции, что позволяет командам более эффективно управлять процессом тестирования и обеспечивать высокое качество программного обеспечения. Популярные инструменты, такие как Selenium, JUnit, TestNG, и Jenkins, стали стандартом для многих команд разработки, помогая автоматизировать не только функциональное, но и

нефункциональное тестирование, включая тестирование производительности, безопасности и совместимости.

1.2 Основные понятия тестирования

Тестирование – процесс, содержащий в себе все активности жизненного цикла, как динамические, так и статические, касающиеся планирования, подготовки и оценки программного продукта и связанных с этим результатов работ с целью определить, что они соответствуют описанным требованиям, показать, что они подходят для заявленных целей и для определения дефектов [2].

Тестирование программного обеспечения (ПО) – это процесс проверки и оценки программного продукта с целью выявления дефектов и обеспечения его соответствия заданным требованиям. Тестирование является неотъемлемой частью разработки ПО и включает в себя различные методики и инструменты для анализа функциональности, производительности, безопасности и других аспектов программного обеспечения.

Цель тестирования – выявление ошибок и дефектов, которые могут возникнуть в процессе разработки ПО. Исправление этих ошибок на ранних стадиях позволяет снизить затраты и улучшить качество конечного продукта.

Основные цели тестирования

техническая: предоставление актуальной информации о состоянии продукта на данный момент.

коммерческая: повышение лояльности к компании и продукту, т.к. любой обнаруженный дефект негативно влияет на доверие пользователей.

Основные термины, необходимые для понимания процесса тестирования:

– Дефект (Defect/Bug) – любое отклонение между ожидаемым и фактическим результатом работы ПО. Идентификация и исправление дефектов направлены на улучшение качества ПО.

– Error – это ситуация, которая происходит, когда разработчики

неправильно понимают требования к продукту, и это приводит к багам [3]

– Failure – сбой в работе компонента, всей программы или системы (может быть как аппаратным, так и вызванным дефектом).

– Тест-кейс (Test Case) – набор условий или переменных для проверки корректности работы системы. Тест-кейсы включают описание входных данных, шаги выполнения и ожидаемые результаты.

– Тест-план (Test Plan) – документ, описывающий цели, объем, подход и фокус тестирования. Тест-план включает стратегии тестирования, критерии начала и окончания, ресурсы и сроки.

– Сценарий использования (Use Case) – это описание взаимодействия между системой и ее пользователем, описывающее конкретный сценарий использования системы для достижения определенной цели или выполнения определенной задачи.

1.3 Классификация типов тестирования ПО

Тестирование по типам можно разделить на функциональное (в число которых входят модульное, интеграционное, системное и приемочное тестирование) и нефункциональное.

Данные типы тестирования представлены на рисунке 1, рассмотрим их подробнее ниже.

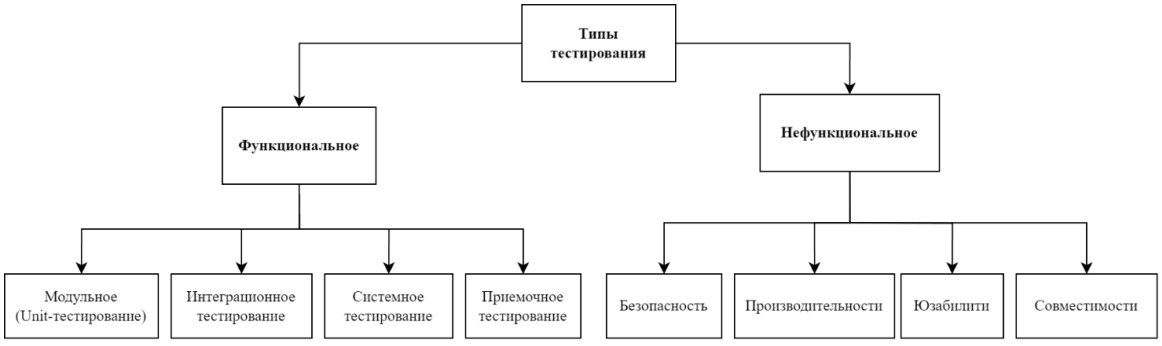

Рисунок 1 – Виды тестирования ПО

Функциональное тестирование (Functional testing) – тестирование, проводимое для оценки соответствия компонента или системы функциональным требованиям [4].

Этот вид тестирования фокусируется на том, чтобы убедиться в том, что программа выполняет то, что от неё ожидается, и что все её функции работают корректно.

Основной подход в функциональном тестировании – это создание тест-кейсов на основе функциональных требований к программе. Эти тест-кейсы затем используются для проверки каждой функции или особенности программы на соответствие заданным критериям.

Процесс функционального тестирования включает в себя следующие этапы:

Шаг 1. Определение входных данных для тестирования

На этом этапе определяются функции, которые необходимо протестировать. Это может включать тестирование функций использования, основных функций и условий ошибок.

Шаг 2. Определение ожидаемых результатов

Создание входных данных на основе спецификаций функции и определение ожидаемых результатов на основе этих спецификаций.

Шаг 3. Выполнение тестовых сценариев

На этом этапе выполняются разработанные тестовые сценарии и записываются результаты.

Шаг 4. Сравнение фактических и ожидаемых результатов

На этом этапе фактический результат, полученный после выполнения тестовых сценариев, сравнивается с ожидаемым результатом для определения отклонения в результатах. Этот этап позволяет определить, работает ли система как ожидалось.

Виды функционального тестирования:

Модульное тестирование (оно же Unit testing/юнит-тестирование) – используется для тестирования какого-либо одного логически выделенного и изолированного элемента системы в коде. Модуль — это независимый

компонент программы, который может быть протестирован отдельно от других модулей [5].

Целью тестирования модуля является не демонстрация правильного функционирования модуля, а демонстрация наличия ошибки в модуле, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования.

На уровне модульного тестирования проще всего обнаружить дефекты, связанные с алгоритмическими ошибками и ошибками кодирования алгоритмов, типа работы с условиями и счетчиками циклов, а также с использованием локальных переменных и ресурсов. Ошибки, связанные с неверной трактовкой данных, некорректной реализацией интерфейсов, совместимостью, производительностью и т.п. обычно пропускаются на уровне модульного тестирования и выявляются на более поздних стадиях тестирования.

Пример. Допустим, у нас есть модуль программы, отвечающий за выполнение математических операций, таких как сложение, вычитание, умножение и деление. Проведем модульное тестирование для операции сложения:

1) Создаем тестовый сценарий, который вызывает функцию сложения с двумя входными аргументами.

2) Проверяем, что результат сложения двух положительных чисел правильный.

3) Проверяем, что результат сложения положительного и отрицательного чисел правильный.

4) Проверяем, что результат сложения отрицательных чисел правильный.

5) Проверяем, что результат сложения нуля с положительным числом равен положительному числу.

6) Проверяем, что результат сложения нуля с отрицательным числом равен отрицательному числу.

7) Проверяем, что результат сложения положительного числа с нулем равен положительному числу.

8) Проверяем, что результат сложения отрицательного числа с нулем равен отрицательному числу.

Как видно из примера выше, в результате такого тестирования, мы проверяем не только правильность результата, но и реакцию функции на различные типы входных данных.

Преимущества модульного тестирования:

– Позволяет выявить дефекты на ранних этапах разработки.
– Упрощает процесс отладки и поиска ошибок.
– Обеспечивает лучшую структурированность кода.
– Снижает затраты на исправление ошибок в долгосрочной перспективе.

Недостатки модульного тестирования:

– Не гарантирует корректную работу взаимодействия между компонентами.
– Требует дополнительных усилий на написание и поддержание тестового кода.
– Может привести к созданию избыточного количества тестов, особенно при неправильном планировании.

Рисунок 2 – Преимущества и недостатки модульного тестирования

Компонентное тестирование – тип тестирования ПО, при котором тестирование выполняется для каждого отдельного компонента отдельно, без интеграции с другими компонентами. Его также называют модульным тестированием (Module testing), если рассматривать его с точки зрения архитектуры. Как правило, любое программное обеспечение в целом состоит из нескольких компонентов. Тестирование на уровне компонентов (Component Level testing) имеет дело с тестированием этих компонентов индивидуально [5].

Для наглядности, в таблице 1 представлены различия между модульным и компонентным тестированием.

Таблица 1 – Сравнение модульного и компонентного тестирований

Характеристика	**Модульное тестирование**	**Компонентное тестирование**
Что тестируем	Тестирование отдельных классов, функций для демонстрации того, что программа выполняется согласно спецификации	Тестирование каждого объекта или частей программного обеспечения отдельно с или без изоляции других объектов
Что тестируется	Проверка на соответствие с проектной документацией (design documents)	Проверка на соответствие с требованиями тестирования и сценариями использования (use case)
Кем выполняется	Пишутся и выполняются разработчиками	Тестировщиками
Когда выполняется	Выполняется первым	Выполняется после модульного тестирования

Практический пример использования компонентного тестирования:

Например, имеется веб-приложение с функциональностью регистрации нового пользователя. В рамках компонентного тестирования мы можем разбить его на отдельные компоненты:

1) Проверка вводимых данных – проверка корректности данных, введенных пользователем в поля регистрации, такие как адрес электронной

почты, пароль и другие.

2) Генерация уникального идентификатора пользователя – здесь мы можем протестировать функцию на генерацию уникальных идентификаторов для каждого пользователя.

3) Отправка уведомлений – проверяем, что уведомления отправляются корректно и содержат необходимую информацию.

Каждый из этих компонентов может быть протестирован отдельно на соответствие своей функциональности требованиям и корректность работы.

Преимущества компонентного тестирования:

- Выявляет ошибки в отдельных компонентах до интеграции.
- Улучшает качество кода и стабильность компонентов.
- Облегчает изоляцию и исправление дефектов.

Недостатки компонентного тестирования:

- Не охватывает взаимодействие между компонентами.
- Требует тщательной настройки тестового окружения.
- Может пропустить ошибки, возникающие на уровне системы.

Интеграционное тестирование – это метод тестирования программного обеспечения, который фокусируется на проверке взаимодействия и обмена данными между различными компонентами или модулями программного приложения [6].

Целью интеграционного тестирования является выявление любых проблем или ошибок, возникающих при объединении и взаимодействии различных компонентов друг с другом.

Интеграционное тестирование обычно проводится после модульного тестирования и перед тестированием системы. Это помогает выявлять и решать проблемы интеграции на ранних этапах цикла разработки, снижая риск

возникновения более серьезных и дорогостоящих проблем в дальнейшем.

Стратегии, методологии и подходы в интеграционном тестировании:

- подход Большого взрыва
- инкрементальный подход.

Подход Большого взрыва подразумевает интеграцию всех модулей одновременно и тестирование только после полной интеграции. Иногда, использование этого метода затрудняет локализацию ошибок и увеличивает время на тестирование и исправление, поэтому также существует и инкрементальный подход. В таком подходе, модули интегрируются и тестируются поэтапно, что позволяет выявлять и исправлять ошибки на каждом этапе, делая процесс тестирования более управляемым и эффективным.

Пример интеграционного тестирования.

В веб-приложении есть два модуля: модуль аутентификации и профиля пользователя. Интеграционное тестирование проверяет взаимодействие этих модулей на соответствие сценария использования приложения.

Например, пользователь регистрируется через модуль аутентификации, и проверяется, что модуль профиля корректно получает и отображает данные о новом пользователе. Затем пользователь обновляет информацию в профиле, и проверяется, что обновленные данные сохраняются и правильно отображаются при повторной аутентификации.

Системное тестирование (System Testing) – это этап, на котором проверяются как функциональные, так и не функциональные требования в системе в целом [7]. При этом выявляются дефекты, такие как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования и т.д., и оцениваются характеристики качества системы

— ее устойчивость, надежность, безопасность и производительность. Иными словами, проверяется, соответствует ли система функциональным требованиям или нет. Приложение или система тестируется в среде, близкой к проектируемой производственной среде.

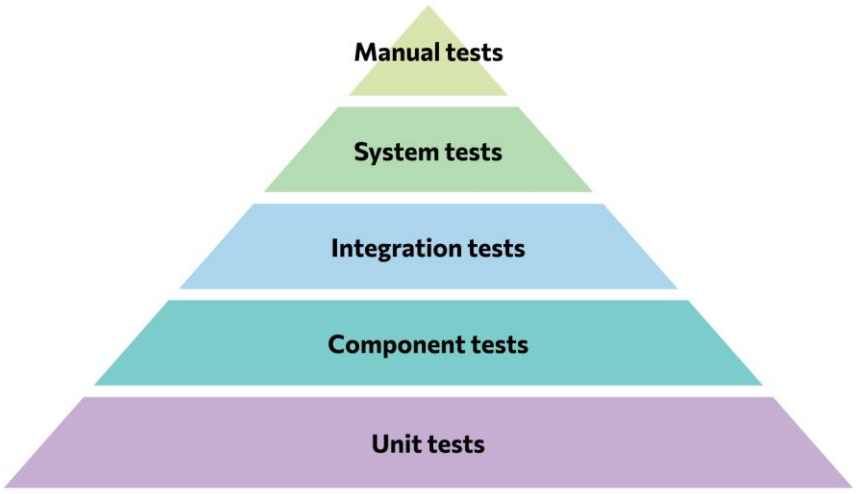

Рисунок 3 – Иерархия тестирования

Нефункциональное тестирование (Non-Functional testing) – тестирование, которое проводится для проверки нефункциональных требований приложения, таких как производительность, безопасность, совместимость, надежность, удобство использования и т. д. В большинстве оно проверяет, соответствует ли поведение системы требованиям по всем аспектам, не охваченные функциональным тестированием. Эти проблемы не связаны с функциональностью системы, но могут негативно повлиять на пользовательский опыт.

Неполный список видов нефункционального тестирования, названия которых соответствует их назначению [8]:

- тестирование производительности (Performance Testing)
- нагрузочное тестирование (Load Testing),
- стрессовое тестирование (Stress Testing)
- объемное тестирование (Volume Testing)

– тестирование восстановления (Recovery Testing)
– тестирование отказоустойчивости (Failover Testing)
– тестирование эффективности (Efficiency Testing)
– тестирование аварийного восстановления (Disaster Recovery Testing)
– тестирование установки (Installation Testing)
– тестирование документации (Documentation Testing)
– тестирование на удобство использования (Usability Testing)
– тестирование графического интерфейса пользователя (User Interface Testing)
– тестирование совместимости (Compatibility Testing)
– тестирование обслуживаемости (Maintainability Testing)
– тестирование безопасности (Security Testing)
– тестирование масштабируемости (Scalability Testing)
– тестирование выносливости (Endurance Testing)
– тестирование надежности (Reliability Testing)
– тестирование соответствия (Compliance Testing)
– тестирование локализации (Localization Testing)
– тестирование интернационализации (Internationalization Testing)
– тестирование переносимости (Portability Testing)
– тестирование на основе базового уровня (Baseline Testing).

Дополнительные виды тестирования, не относящиеся напрямую к категориям, но играющие значительную роль при разработке ПО:

Smoke-тестирование (или смоук-тестирование) – это вид тестирования, который выполняется для быстрой проверки основных функциональных возможностей приложения или системы после каждого крупного изменения или перед выпуском новой версии [9].

Цель smoke-тестирования – удостовериться, что основные функции

приложения или системы работают без критических ошибок и что приложение готово к более глубокому тестированию. Обычно smoke-тестирование включает в себя проверку основных сценариев использования или ключевых функций приложения.

Почему дымовое тестирование так называется? – Наиболее правдоподобная версия: термин инженеров-электротехников. Когда на электрическую цепь впервые подавалось питание, она могла дымиться если пайка была небрежной, или если в устройстве были дефекты.

Предназначение дымового тестирования: проверка важнейших функций приложения. Простой и быстрый метод проверки приложения на ранней стадии разработки.

Данное тестирование может быть и ручным, и автоматизированным. Может быть и гибридное тестирование: сочетание ручного и автоматизированного.

А/Б-тестирование (AB testing) – метод исследования для оценки эффективности двух вариантов одного элемента. В маркетинге это может быть кнопка на странице сайта, рассылка, заголовки и любые другие детали. Суть в том, чтобы на протяжении определенного времени показывать их двум сегментам аудитории.

Допустим, имеется интернет-магазин. Кнопка «Заказать» выделена красным цветом, но команда разработки считает, что это больше отпугивает, чем призывает купить продукт. Узнать, действительно ли это так и насколько это повлияет на продажи можно при помощи такого тестирования.

Идея – разделить аудиторию на контрольную и тестовую группы, создать две разные страницы и проверить опытным путем. Пользователи из каждой группы будут взаимодействовать с разными версиями сайта. В итоге конверсия у одной из кнопок окажется выше – исследование прошло успешно и можно вносить на сайт изменения [10].

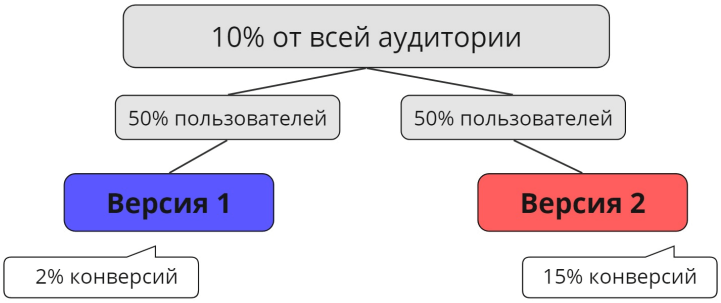

Рисунок 4 – Деление аудитории при A/B тестировании

Альфа- и бета- тестирование (Alpha Testing and Beta Testing) – эксплуатационное тестирование потенциальными пользователями/заказчиками или независимой командой тестирования разрабатываемого продукта [11].

Альфа-тестирование – внутреннее тестирование, проводимое разработчиками и тестировщиками компании в контролируемой среде для выявления дефектов до публичного выпуска.

Бета-тестирование – внешнее тестирование, проводимое реальными пользователями в реальных условиях эксплуатации для выявления дефектов и сбора обратной связи перед окончательным выпуском продукта. Устранение проблем в бета-версии может значительно снизить затраты на разработку, поскольку большинство незначительных сбоев будут исправлены до окончательной версии.

2 МЕТОДЫ И СПОСОБЫ ТЕСТИРОВАНИЯ

2.1 Ручное и автоматизированное тестирование

Тестирование программного обеспечения может проводиться вручную (ручное тестирование) или автоматизировано (автоматизированное тестирование). У каждого из этих подходов есть свои преимущества и недостатки, поэтому выбор между ними зависит от конкретных требований и условий разработки приложения. Рассмотрим каждый из методов подробнее.

Ручное тестирование

Ручное тестирование – это вид тестирования программного обеспечения, при котором тесты выполняются тестировщиком вручную, без использования каких-либо средств автоматизации. Оно существует столько же лет, сколько и сама разработка программного обеспечения, и является наиболее важным компонентом процесса обеспечения качества. Без ручного тестирования популярные программные продукты никогда не смогли бы работать так хорошо, как они работают, иметь такой привлекательный пользовательский интерфейс и быть способными противостоять возможным атакам. Также, этот вид тестирование первое, что обычно выбирает компания в попытке поддержать или улучшить качество приложения [12]. Зачастую оно остается единственным видом тестирования, используемым в проекте.

Ситуации, когда необходимо ручное тестирование:

– Когда продукт находится на начальной стадии разработки. На этом этапе функциональность и состояние приложения подвержены частым изменениям, и ручное тестирование лучше справляется с этими изменениями. Автоматизированное тестирование, с другой стороны, требует значительных ресурсов для успешной работы на этом этапе, что не всегда оправдано.

– Когда проект краткосрочный и небольшой. Запуск автоматизации тестирования требует значительных человеческих и материальных ресурсов и

времени, в отличие от ручного тестирования, которое может быть внедрено в проект за считанные дни, именно поэтому он является предпочтительным решением для малых и средних проектов.

– При тестировании удобства использования продукта. Некоторые средства автоматизации тестирования достаточно успешно справляются с имитацией поведения человека при взаимодействии с пользовательским интерфейсом. Тем не менее, они еще не могут полностью имитировать многие, часто непредсказуемые вещи, которые могут прийти в голову тестировщику при тестировании удобства использования решения.

– Когда проводится интуитивное или исследовательское тестирование. Помимо тестирования удобства использования, это два вида тестирования, которые в значительной степени зависят от реального взаимодействия человека с продуктом. Эти виды тестирования можно в определенной степени автоматизировать, но на данный момент результаты автоматизации этих видов тестирования далеки от тех, которые дает ручное тестирование.

– При работе с физическими продуктами. Тестирование физических устройств, таких как подключенные к интернету встраиваемые системы, автомобильные устройства или медицинская техника, не всегда легко автоматизировать и не всегда нужно автоматизировать. Гибкое ручное тестирование, которое можно легко подстроить под потребности продукта, в большинстве случаев является лучшим вариантом.

Ручное тестирование проводится следующим образом:

1) Планирование тестирования. Начинается с определения целей тестирования, создания тестового плана и составления набора тестовых сценариев.

2) Разработка тест-кейсов. Тестировщики создают конкретные инструкции по тестированию, описывающие шаги, ожидаемые результаты и ожидаемое поведение приложения. Например, если тестируется функция входа

в систему, тест-кейс может содержать такие шаги, как ввод логина и пароля, нажатие кнопки "Войти" и проверку успешного входа или отображение сообщения об ошибке.

3) Выполнение тест-кейсов. Тестировщики последовательно выполняют каждый тест-кейс в соответствии с его инструкциями.

4) Запись результатов. Для каждого тест-кейса фиксируются результаты тестирования, включая обнаруженные ошибки или неполадки.

5) Анализ полученных результатов. Собранные результаты тестирования анализируются для выявления тенденций, частых проблем и общего качества программы.

6) Обратная связь и улучшения. Информация о найденных ошибках передается разработчикам для исправления. Кроме того, результаты тестирования используются для улучшения процесса разработки и качества продукта в целом.

Самый большой из недостатков ручного тестирования – человеческий фактор [13]. Тестировщик может пропустить потенциальную ошибку. Или, например, один и тот же сценарий два тестировщика могут проверить разными способами. Решением проблемы может быть автоматизация рутинных тестов, что снижает риск пропуска ошибок и обеспечивает стандартизированное выполнение тестов.

Автоматизированное тестирование

Автоматизированное тестирование – это метод тестирования программного обеспечения, который предполагает использование инструментов и фреймворков автоматизации для выполнения одного и того же набора тест-кейсов снова и снова. Ключевое различие между ручным и автоматизированным тестированием заключается в том, что ручное тестирование полностью зависит от человека, сидящего за компьютером. В то время как автоматизированные

тесты могут быть написаны один раз и выполняться многократно практически без участия человека.

Основные области применения автоматизированного тестирования:

– При выполнении повторяющихся тестов. Это один из самых распространенных случаев в пользу использования автоматизации: когда один и тот же набор тест-кейсов выполняется каждый день или несколько раз в день, имеет смысл автоматизировать его и вносить незначительные изменения только по мере необходимости.

– При использовании тестирования производительности или при нагрузочном тестировании. Эти два вида тестирования требуют много времени и усилий от команды тестировщиков, поскольку найти уязвимости в производительности продукта может быть непросто. Автоматизация тестирования – это разумный способ проверить производительность продукта с разных сторон.

– Когда имеется большое количество тест-кейсов. После того как команда тестировщиков проработала над продуктом некоторое время, количество тест-кейсов может достигать нескольких тысяч и более. Следовательно, команда, работающая вручную, рискует потратить недели на выполнение набора тестов, в то время как остальная работа будет откладываться.

– Когда необходимо исключить человеческий фактор.

– При работе с большими объемами данных. Например, одним из многих случаев, когда автоматизация тестирования является наилучшим вариантом, является тестирование баз данных. Хорошо написанный набор тест-кейсов может обработать миллионы записей за гораздо меньшее время, чем потребовалось бы ручному тестировщику для выполнения даже малой доли этой задачи.

Как выполняется автоматизированное тестирование?

1) Определение целей автоматизации и выбор подходящего инструмента автоматизации тестирования, учитывая особенности тестируемого приложения, технологии, доступные ресурсы и требования к тестированию.

2) Установка и настройка выбранного инструмента автоматизации, а также создание тестовых сред, баз данных и других необходимых компонентов.

3) Разработка тестовых скриптов: Написание скриптов или сценариев тестирования с использованием языков программирования, таких как Java, Python, C#, JavaScript и другие. Тестовые скрипты определяют шаги тестирования, ожидаемые результаты и проверки.

4) Автоматизированные тесты запускаются с использованием инструмента автоматизации тестирования. Тесты выполняются на тестируемом приложении или системе, а результаты регистрируются автоматически.

5) Анализ результатов выполнения тестов для выявления ошибок, несоответствий и других проблем. Автоматизированные отчеты об ошибках могут быть сгенерированы инструментом автоматизации тестирования для дальнейшего анализа и исправления.

Как было сказано ранее, существуют инструменты, помогающие автоматизировать тесты. Наиболее популярными инстументами для автоматизации тестирования на данный момент являются:

Selenium – открытый инструмент для тестирования веб-приложений, поддерживающий множество браузеров и языков программирования.

JUnit – фреймворк для модульного тестирования на Java, широко используемый для тестирования Java-приложений.

TestNG – фреймворк, похожий на JUnit, но с более расширенными возможностями, такими как параллельное выполнение тестов.

Cucumber – инструмент для поведения-ориентированного тестирования (BDD), который позволяет писать тесты на естественном языке.

Appium – инструмент для автоматизации мобильных приложений, поддерживающий как Android, так и iOS.

Ranorex – инструмент для тестирования настольных, веб- и мобильных приложений с возможностями для кроссплатформенного тестирования.

Robot Framework – открытый фреймворк для приемочного тестирования и тестирования на основе ключевых слов, поддерживающий расширение через библиотеки Python и Java.

В качестве примера использования автоматизированных средств тестирования, рассмотрим использование Selenium.

Selenium представляет собой среду тестирования для проверки веб-приложений в различных браузерах и платформах. Он поддерживается несколькими ОС (Windows, Mac, Linux), а также многими браузерами (Chrome, Firefox, и браузерами Headless) [14]. Скрипты для него можно написать на большинстве популярных сегодня языках программирования, включая Python, C#, Java и другие. Использование Selenium позволяет разработчикам создавать скрипты тестирования, которые воспроизводят действия пользователей, такие как нажатие кнопок, ввод текста, навигация по страницам и многое другое. Путем создания таких скриптов тестирования разработчики могут проверять функциональность и корректность работы веб-приложений на различных браузерах и платформах без необходимости ручного взаимодействия с интерфейсом.

Практический пример использования Selenium

Например, у нас есть веб-приложение для управления списком задач. Нам необходимо проверить добавление новой задачи в список задач, для чего мы решили использовать язык Python и библиотеку Selenium.

Тестовый скрипт для проверки может выглядеть следующим образом:

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time
# Инициализация драйвера браузера (например, Chrome)
driver = webdriver.Chrome()
# Открытие веб-приложения
driver.get("https://my-secret-todo.ru")
# Находим поле для ввода новой задачи
input_field = driver.find_element_by_id("task-input")
# Вводим новую задачу в поле
new_task = "Полить цветы"
input_field.send_keys(new_task)
# Нажимаем клавишу Enter для добавления задачи
input_field.send_keys(Keys.ENTER)
# Добавленная задача должна отобразиться в списке задач
time.sleep(2)  # Ждем немного, чтобы список успел обновиться
task_list = driver.find_element_by_id("task-list")
tasks = task_list.find_elements_by_tag_name("li")
# Проверяем, содержится ли новая задача в списке
task_added = False
for task in tasks:
    if task.text == new_task:
        task_added = True
        break
# Проверяем, была ли задача успешно добавлена
assert task_added, "New task was not added to the list!"
# Закрываем браузер после завершения теста
driver.quit()
```

Рассмотрим код скрипта выше. Сначала, мы инициируем драйвер браузера Chrome (программу, которая позволяет управлять веб-браузером из кода), который открывает страницу указанного ранее веб-приложения. Затем он находит элемент на странице по его id и вводит текст в поле. После этого он нажимает клавишу Enter, чтобы выполнить поиск. Затем он проверяет результат поиска, убедившись, что введенный текст появляется на странице. Наконец, браузер закрывается после завершения теста.

Может возникнуть вопрос, почему же пример, приведенный выше является автоматизированным тестированием, код для тестов пишется вручную? Ответ исходит из ранее рассмотренного определения автоматизированного тестирования – вместо того, чтобы вручную выполнять каждый шаг теста в браузере, Selenium позволяет написать скрипт, который автоматически выполняет эти шаги. Иными словами, автоматизация состоит в том, что тесты могут быть запущены без необходимости вручную взаимодействовать с приложением, что экономит время и повышает эффективность тестирования.

Сравнение ручного и автоматизированного тестирования

Существуют различные способы сравнить и провести различие между ручным и автоматизированным тестированием. Можно посмотреть, например, чего эти два метода могут достичь, и на инструменты, которые они используют. Однако некоторые из наиболее важных аспектов спора выбора между автоматизированным и ручным тестированием можно найти в более практической сфере. За каждым проектом, будь то ручное или автоматизированное тестирование, стоят человеческие и материальные ресурсы. Время выхода на рынок также является важной метрикой, которую необходимо учитывать [12]. Ниже представлена разбивка по этим ключевым параметрам.

Таблица 2 – Сравнение ручного и автоматизированного тестирования

Характеристика	**Ручное тестирование**	**Автоматизированное тестирование**
Кем выполняется	Ручные QA специалисты и инструменты ручного тестирования.	Специалисты по автоматизированному тестированию со знанием кода и фреймворков тестирования.
Время	Может быть запущено очень быстро.	На настройку могут уйти недели.
Стоимость	Относительно низкая, поскольку ручные QA специалисты оплачиваются не так высоко, как специалисты по автоматизации, и может использоваться имеющееся оборудование.	Специалисты по автоматизации стоят дороже, и может потребоваться дополнительное оборудование.
Рентабельность	Низкая, поскольку ручные тест-кейсы не всегда можно использовать повторно.	Высокая, так как помогает экономить ресурсы на повторных тестах.
Необходимые навыки программирования	Нет	Да
Повторение	Ручной QA специалист, выполняющий одни и те же тесты раз за разом, может потерять фокус и пропустить ошибки.	Можно повторять снова и снова с одинаковой эффективностью.
Человеческие ошибки	Есть склонность к человеческим ошибкам.	Исключение человеческой ошибки.

Поясним некоторые характеристики из таблицы 1:

Стоимость – по некоторым оценкам, тестирование ПО может составлять до 60% от общей стоимости проекта. Автоматизация тестирования дороже на начальных этапах из-за затрат на высокооплачиваемых специалистов и сложные инструменты. Однако в долгосрочной перспективе автоматизация экономит деньги благодаря повторному использованию тестов, что делает её идеальной для долгосрочных проектов. Ручное тестирование лучше подходит для небольших, краткосрочных задач.

Человеческие ресурсы – квалифицированная команда ручных тестировщиков может существенно улучшить качество ПО, но требует значительных человеческих ресурсов для выполнения всех тестов вручную. Автоматизация позволяет эффективнее использовать человеческие ресурсы, так как один специалист по автоматизации может заменить нескольких ручных тестировщиков, что делает это более выгодной инвестицией.

Доступность для новичков – ручное тестирование привлекательно для новичков из-за низкого порога входа, не требующего глубоких знаний программирования. Со временем многие ручные тестировщики переходят к автоматизации, что является горизонтальным продвижением, а не вертикальным карьерным ростом. Разработчики, переходящие в автоматизацию тестирования, могут использовать свои знания кода для более эффективной работы.

Время выхода на рынок – в условиях жесткой конкуренции быстрое время выхода на рынок может стать ключевым преимуществом. Ручное тестирование подходит для активно разрабатываемых приложений, но может замедлить вывод продукта на рынок из-за больших временных и людских затрат. Автоматизация тестирования генерирует результаты быстрее, выявляя больше ошибок за меньшее время и ускоряя выход продукта на рынок, при условии регулярного обновления тест-кейсов для поддержания их актуальности.

2.2 Методики тестирования ПО

Существуют стратегии проведения тестирования, среди них:

1. Тестирование по стратегии чёрного ящика

Это такой процесс тестирования, в котором тестировщику ничего неизвестно. Тестировщик, как обычный пользователь, что-то делает, не зная никаких особенностей реализации [15].

Почему именно чёрный ящик? – Такое название обусловлено тем, что тестировщик, как и наблюдатель за черным ящиком, не имеет прямого доступа к внутренним механизмам программы или приложения. Он тестирует их функциональность, используя различные входные данные, но не раскрывает подробности внутренней реализации, визуализация представлена на рисунке.

Рисунок 5 – Тестирование «черный ящик»

Данный процесс тестирования системы и её поведения вне зависимости от её внутренней структуры, архитектуры и реализации. Тестировщик осуществляет ввод, а вывод рассматривается как часть этой методики тестирования ПО, что позволяет определить реакцию системы на ожидаемые и неожиданные действия пользователя, время реакции, сложности юзабилити и проблемы с надёжностью.

Типы тестирования, используемые в стратегии чёрного ящика: рассмотренные в первом пункте типы тестирования – функциональное, нефункциональное и регрессионное.

Методики стратегии чёрного ящика:

– Эквивалентное разделение. Группировка входных данных и тестирование одного представителя из каждой группы.

– Анализ граничных значений. Проверка поведения системы вблизи некоторых граничных значений.

– Симуляция таблицы решений. Создание тестовых случаев на основе правил, определяющих результаты на основе входных данных.

– Тестирование на изменение состояния. Проверка системы при переходе между различными состояниями.

– Угадывание ошибок. Тестирование на частые ошибки, такие как неправильная обработка входных данных или известные уязвимости ПО.

В результате такого тестирования можно обнаружить ошибки, недочеты и проблемы в интерфейсе, логике работы программы, а также проверить соответствие требованиям и ожиданиям пользователя.

2. Тестирование методом «белого ящика»

Тестирование «белого ящика» — это способ проверки ПО, который концентрируется на внутренней системе и коде программы. У QA-специалистов есть доступ к исходному коду и документации проекта, что позволяет им исследовать и проверять внутреннюю работу, инфраструктуру и интеграцию программного обеспечения [16]. Обычно используется во время модульного тестирования, хотя может применяться и на других этапах, таких как интеграционные тесты. Для применения этого метода QA-специалист должен владеть обширными знаниями о технологии, используемой для разработки программы.

Методики тестирования белого ящика:

– Модульное тестирование

– Интеграционное тестирование

– Регрессионное тестирование

– Тестирование ветвей, указывающее на то, все ли «ветви» в кодовой базе проверяются тестами.

– Тестирование пути, предполагающее использование исходного кода программы для поиска всех возможных путей его исполнения.

– Тестирование циклов для проверки валидности циклов алгоритмов ПО, чтобы найти ошибки, которые могут там присутствовать.

Какие проблемы могут быть найдены таким методом тестирования?

– Дефекты в коде: этот способ тестирования позволяет тщательно проверить код на наличие ошибок.

– Недостатки производительности: данный метод находит участки кода, которые могут привести к снижению эффективности работы ПО.

– Нарушения безопасности: тестирование помогает проверить, защищено ли ваше приложение от взлома и других видов атак.

– Определение некорректного поведения программы: мы можем узнать, работает ли ПО так, как было задумано, и соответствует ли оно требованиям.

3. Тестирование методом «серого ящика»

Тестирование «серого ящика» представляет собой комбинацию тестирования «черного ящика» и «белого ящика».

Цели тестирования «серого ящика»: комбинировать преимущества «черного» и «белого» тестирования, объединить усилия разработчиков и тестировщиков, улучшить общее качество продукции, сократить расходы на тестирование, предоставить разработчикам больше времени на исправления и оценить продукт с точки зрения пользователя.

Методики тестирования «серым ящиком» [17]:

– Матричное тестирование (Matrix Testing), разработчики предоставляют все переменные в программе, а также связанные с ними технические и бизнес-риски. Методика матричного тестирования проверяет риски, определенные разработчиками. Матричный метод устанавливает все используемые переменные в программе. Этот метод помогает идентифицировать и удалять переменные, которые не используются в программе, и, в свою очередь, помогает увеличить скорость работы программного обеспечения;

– Регрессионное тестирование

– Тестирование ортогональных массивов или OAT (Orthogonal Array Testing or OAT) используется для сложных функций или приложений, когда требуется максимальное покрытие кода с минимальным количеством тест-кейсов и имеет большие тестовые данные с n числом комбинаций;

– Паттерны проектирования, тестирование по образцу на основе предыдущих дефектов, обнаруженных в ПО. Записи о дефектах анализируются на предмет причин дефектов, и создаются тест-кейсы на основе этих дефектов и их причин.

2.3 Практики обеспечения качества ПО

Что значит качество ПО?

Качество программного обеспечения (Software Quality) – это совокупность характеристик программного обеспечения, относящихся к его способности удовлетворять установленные и предполагаемые потребности [18].

Качество (Quality) – степень соответствия совокупности присущих характеристик объекта требованиям.

Обеспечение качества (Quality Assurance): Часть менеджмента качества, направленная на создание уверенности, что требования к качеству будут выполнены.

Управление качеством (Quality Control) – часть менеджмента качества,

направленная на выполнение требований к качеству.

Всеобщее управление качеством (Total Quality Management, TQM) – это общеорганизационный метод непрерывного повышения качества всех организационных процессов. Цель TQM заключается в достижении долгосрочного успеха за счет удовлетворения потребностей клиентов и всех заинтересованных сторон. Применяется в разных сферах, в том числе и при разработке ПО.

Практики обеспечения качества ПО включают стандартизацию процессов, регулярные аудиты и инспекции, обучение и развитие сотрудников, непрерывное улучшение процессов, использование автоматизированных тестов, метрики и аналитику качества, а также управление рисками.

Стандартизация процессов предполагает установление стандартов и протоколов для всех этапов разработки и тестирования ПО, что помогает поддерживать последовательность и предсказуемость процессов. Регулярные аудиты и инспекции проводятся для выявления несоответствий и областей для улучшения, включая как внутренние, так и внешние проверки. Обучение и развитие сотрудников заключается в инвестициях в их профессиональное развитие и повышение навыков в области обеспечения качества.

Непрерывное улучшение процессов достигается применением различных методологий, таких как Lean и Six Sigma, которые способствуют постоянному совершенствованию процессов разработки и тестирования. Использование автоматизированных тестов повышает эффективность и покрытие тестов, ускоряя процесс тестирования и уменьшая количество человеческих ошибок. Метрики и аналитика качества помогают определить и отслеживать ключевые показатели эффективности (KPI), что способствует мониторингу качества ПО на всех этапах разработки и принятию обоснованных решений для корректировки процессов.

Управление рисками включает идентификацию и управление рисками, связанными с качеством ПО, проведение анализа рисков и разработку стратегий

для их минимизации. Применение этих практик помогает обеспечить высокое качество программного обеспечения, удовлетворяющее потребности пользователей и соответствующее требованиям рынка.

Применение этих практик помогает обеспечить высокое качество программного обеспечения, удовлетворяющее потребности пользователей и соответствующее требованиям рынка.

3 АНАЛИЗ ПРИМЕРОВ ТЕСТИРОВАНИЯ

В основном, процессы тестирования приложений закрыты и нельзя точно определить, как тестировался тот или иной программный продукт, если об этом не написал кто-нибудь из команды разработчиков или сама компания. Тем не менее, рассмотрим несколько результатов хорошо проведенных тестирований, которые в итоге повлияли на восприятие продукта позитивно и негативно.

3.1 Пример успешно проведенного тестирования

1) Финансовый маркетплейс Финуслуги

Финуслуги – это платформа личных финансов, с помощью которой можно выбирать и открывать банковские вклады онлайн и осуществлять другие операции. Перед командой разработчиков была поставлена цель сделать мобильное приложение. Команда QA активно участвовала во всех этапах разработки, от архитектуры до поддержки. Процесс тестирования был четко структурирован: тестирование фич проводилось в течение первых 5-6 дней спринта, затем следовала стабилизация сборки, смоук-тестирование и проверка [19]. В ходе тестирования использовались автотесты. Процесс тестирования был измеримым и прозрачным благодаря метрикам качества и четко установленным регламентам. Результатом, стало удовлетворение пользователями и получение премии Рунета за разработку мобильного приложения, что подтверждает успешность процесса тестирования и работы всей команды.

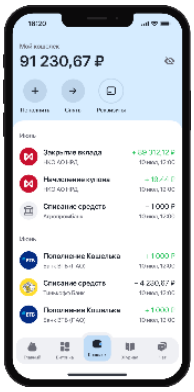

Рисунок 6 – Приложение Финуслуги на iOS

3.2 Примеры неудачного тестирования

1) Microsoft и Windows Vista

Windows Vista была одним из наиболее известных примеров неудачного тестирования. Несмотря на то, что у Microsoft было много амбициозных планов для Vista, включая значительные изменения в пользовательском интерфейсе и улучшенную безопасность, реальность оказалась намного менее впечатляющей. Проблемы с производительностью, высокие требования к аппаратному обеспечению и проблемы совместимости с программным обеспечением стали серьезными препятствиями для пользователей. Это привело к негативным отзывам и недовольству со стороны пользователей, а также к низким продажам. Кроме того, ряд функций, таких как User Account Control (UAC), были реализованы таким образом, что вызывали частые запросы на подтверждение действий, что раздражало пользователей и снижало удобство использования. Эти проблемы оказали значительное влияние на репутацию и финансовые результаты Microsoft, а Vista была позднее заменена Windows 7, которая исправила многие из этих проблем [20].

В данном случае, недостаточное внимание к производительности, совместимости и удобству использования в рамках системного тестирования привело к серьезным проблемам при реальном использовании продукта. Также отсутствие достаточного тестирования совместимости и функциональности в реальных условиях использования подчеркивает важность обширного и всестороннего тестирования перед выпуском продукта на рынок.

Рисунок 7 – Операционная система Windows Vista

2) ВКонтакте и редизайн 2016 года

1 апреля 2016 года социальная сеть «ВКонтакте» представила редизайн своего сайта. Новый дизайн был тогда запущен в тестовом режиме (A/B тестирование). Работа над ним велась полтора года. 17 августа произошел окончательный переход. Все пользователи социальной сети были принудительно переведены на новый дизайн.

Пользователи были недовольны не только внешним видом, но и по части юзабилити нового интерфейса, например, страницы новостей стали ниже на 42px [21]. Из-за фиксированной шапки вертикальное пространство сайта стало меньше и нужно больше проматывать – рисунок 1.

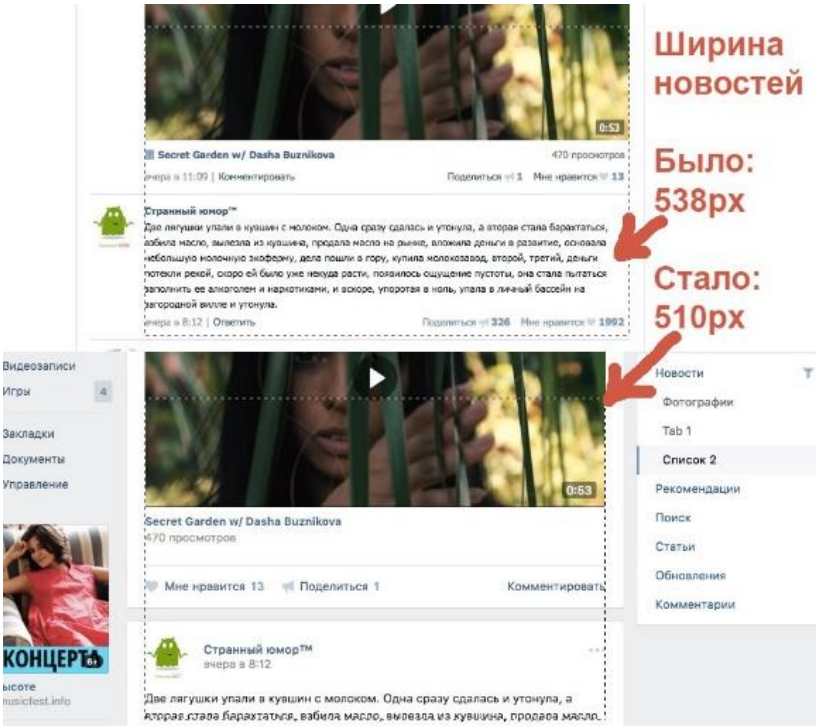

Рисунок 8 – Проблема редизайна ВК

Можно сделать вывод, что неудача связана с недостаточной проверкой и тестированием юзабилити нового интерфейса. Это свидетельствует о необходимости более тщательного тестирования и анализа изменений в пользовательском опыте.

4 БУДУЩИЕ НАПРАВЛЕНИЯ И ПЕРСПЕКТИВЫ

Использование нейросетей и моделей машинного обучения

Анализировать человеческое поведение – сложная задача, но нейросети и машинное обучение могут помочь в этом процессе. Благодаря им можно оптимизировать наборы тестов и повторно использовать тестовые сценарии, а также составлять подробные отчёты о результатах испытаний.

Нейросети и модели машинного обучения позволят свести к минимуму ручное тестирование и повысить точность тестов при минимальном вмешательстве человека [22]. С помощью нейросетей и машинного обучения можно также создавать тестовые данные, прогнозировать сбои, оптимизировать процессы и мониторинг и так далее.

Автоматическое тестирование без сценариев

Инструменты для автоматического тестирования без сценариев уже существуют и предполагают проведение тестов без кода или с его минимальным использованием. Эти инструменты уступают традиционным методам по эффективности, но ожидается, что автоматическое тестирование без сценариев будет развиваться, чтобы процессы стали быстрее и проще. Если тестирование станет более доступным тем, кто не является разработчиком, нетехнические специалисты смогут участвовать в тестировании: планировать, выполнять и анализировать тесты без написания тестового кода. Это повысит производительность команд и компаний, которые разрабатывают различные продукты, и сделает тестирование более простым и дешёвым.

Интеграция с девопсом

Проектирование качества уже давно вошло в практику девопса, и эта тенденция будет только усиливаться. Для этого уже даже есть названия — QAOps (Quality Assessment Operations) и TestOps (Testing Operations). Эти области деятельности используют тот же подход, что и девопс, с той оговоркой,

что он основан на непрерывном тестировании. Другие их свойства состоят в автоматизации контроля качества для ускорения цикла разработки.

Ранняя интеграция тестирования в жизненный цикл разработки ПО даст возможность создавать надёжные программы и исключит необходимость исправлять код, экономя таким образом ресурсы, время и затраты.

Автоматизация для улучшения всего жизненного цикла тестирования затронет все этапы от планирования до мониторинга. Это сократит объём ручного труда и обеспечит непрерывное тестирование на каждом этапе разработки.

Фокус на пользовательский опыт

Функциональной эффективности уже недостаточно, веб-сайты, приложения и программы должны быть удобными и иметь интуитивно понятный пользовательский интерфейс. Из-за этого UI/UX-дизайн начинают ценить едва ли не больше, чем качественную разработку.

Тщательное и эффективное UX-тестирование может играть решающую роль в будущем продукта. При этом такое тестирование можно проводить только вручную: только люди могут судить, подходит ли продукт другим людям и будут ли они им удовлетворены. Кроме этого, пользовательские интерфейсы становятся сложнее. В них добавляются голосовое взаимодействие, дополненная реальность и персонализированный контент. Поэтому от тестировщиков требуется ориентироваться на пользователей, чтобы гарантировать, что эти функции работают как надо, они доступны и не вызывают негативных впечатлений.

Для наилучшего UX-тестирования необходимо анализировать данные о поведении пользователей, чтобы постоянно улучшать интерфейсы, функции и пользовательский опыт.

ЗАКЛЮЧЕНИЕ

Роль тестирования и обеспечения качества в жизненном цикле разработки программного обеспечения является неотъемлемой и ключевой. В процессе разработки ПО тестирование играет решающую роль в обеспечении функциональности, надежности, производительности и безопасности разрабатываемых продуктов. Качественное тестирование позволяет выявлять дефекты на ранних этапах разработки, улучшать процессы разработки и повышать удовлетворенность пользователей. Можно сделать вывод, что проведение качественного тестирования помогает

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Краткая история тестирования ПО // Хабр URL: https://habr.com/ru/companies/sberbank/articles/687356/ (дата обращения: 24.05.2024).

2. ISTQB Glossary // ISTQB Glossary URL: https://glossary.istqb.org/en_US/ [Электронный ресурс] (дата обращения: 23.05.2024).

3. Error, Defect, Fault, Bug и Failure — в чем разница // TestEngineer URL: https://testengineer.ru/error-defect-fault-bug-failure/ (дата обращения: 24.05.2024).

4. Functional Testing – Software Testing // GeeksForGeeks URL: https://www.geeksforgeeks.org/software-testing-functional-testing/ [Электронный ресурс] (дата обращения: 23.05.2024).

5. Модульное/юнит/компонентное тестирование (Module/Unit/Component testing) // QA_Bible URL: https://vladislaveremeev.gitbook.io/qa_bible/vidy-metody-urovni-testirovaniya/modulnoe-yunit-komponentnoe-testirovanie-module-unit-component-testing (дата обращения: 24.05.2024).

6. Интеграционное тестирование: что это? Виды, примеры. // Logrocon URL: https://logrocon.ru/news/intgration_testing (дата обращения: 24.05.2024).

7. Теория тестирования ПО просто и понятно // Хабр URL: https://habr.com/ru/articles/587620/ (дата обращения: 24.05.2024).

8. Non Functional Testing // Guru99 URL: https://www.guru99.com/non-functional-testing.html (дата обращения: 24.05.2024).

9. Что такое smoke-тестирование? // TestEngineer URL: https://testengineer.ru/chto-takoe-smok-testirovanie/ (дата обращения: 24.05.2024).

10. Полный гайд по A/B-тестированию // РЕГ.РУ URL: https://www.reg.ru/blog/polnyj-gajd-po-b-testam/ (дата обращения: 24.05.2024).

11. Alpha Testing Vs Beta Testing – Difference Between Them // Guru99 URL: https://www.guru99.com/alpha-beta-testing-demystified.html (дата обращения: 24.05.2024).

12. Automation Testing vs. Manual Testing: Will Automation Replace Manual QA? // DZone URL: https://dzone.com/articles/automation-testing-vs-manual-testing-will-automati (дата обращения: 24.05.2024).

13. Гид по ручному тестированию приложений: преимущества, этапы и методологии // Хабр URL: https://habr.com/ru/companies/skillbox/articles/418889/ (дата обращения: 24.05.2024).

14. Топ 10 инструментов автоматизации тестирования 2023 // Хабр URL: https://habr.com/ru/articles/342234/ (дата обращения: 24.05.2024).

15. What Is Black Box Testing? // DZone URL: https://dzone.com/articles/what-is-black-box-testing (дата обращения: 24.05.2024).

16. Тестирование методом «белого ящика» (White Box Testing) // Точка качества URL: https://tquality.ru/blog/testirovanie-metodom-belogo-yashchika/ (дата обращения: 24.05.2024).

17. Gray Box Testing – Software Testing // GeeksforGeeks URL: https://www.geeksforgeeks.org/gray-box-testing-software-testing/ (дата обращения: 24.05.2024).

18. Что такое качество. Разбираемся в иерархии терминов «QA», «QC» и «тестирование» // Хабр URL: https://habr.com/ru/companies/rostelecom/articles/647963/ (дата обращения: 24.05.2024).

19. Как мы тестировали первый в России финансовый маркетплейс // Хабр URL: https://habr.com/ru/companies/agima/articles/706312/ (дата обращения: 24.05.2024).

20. What Really Happened with Vista // HackerNoon URL: https://hackernoon.com/what-really-happened-with-vista-4ca7ffb5a1a (дата обращения: 24.05.2024).

21. Что думают эксперты и пользователи про новый дизайн «ВКонтакте» // Хабр URL: https://habr.com/ru/articles/308066/ (дата обращения: 24.05.2024).

22. Как дела с тестированием в 2024 году // Код. Журнал Яндекс Практикума URL: https://thecode.media/devops-2024/ (дата обращения: 24.05.2024).