GeekBrains

Введение в тестирование

Урок 5

Виды тестирования

Оглавление

На этом уроке 3
Функциональное тестирование **Error! Bookmark not defined.**
Три ящика тестирования 3
Метод чёрного ящика 3
Метод белого ящика 5
Метод серого ящика 5
Виды тестирования после изменений в коде 6
Статическое и динамическое тестирование 6
Позитивное и негативное тестирование 7
Альфа- и бета-тестирование 8
Тестирование на основе тест-кейсов и исследовательское тестирование 10
Смоук-тесты 11
Нефункциональное тестирование 112
Тестирование производительности 122
Нагрузочное тестирование 123
Тестирование масштабируемости 123
Объёмное тестирование 134
Стрессовое тестирование 134
Основные понятия при тестировании производительности 134
Нефункциональное тестирование 116
Контрольные вопросы 16
Дополнительные материалы 17

На этом уроке

1. Познакомимся с функциональным и нефункциональным тестированием, их видами и особенностями.

Функциональное тестирование

Функциональное тестирование - это тестирование ПО, в процессе которого проверяется реализация функциональных требований, то есть проверка работы функциональностей, направленных на решение задач пользователя.

Три ящика тестирования

Работая с программой, тестировщик обычно держит в уме её архитектурные компоненты и особенности их взаимодействия.

Бывают ситуации, когда тестировщику ничего не известно об устройстве программы «под капотом». А иногда тестировщик видит код программы и пишет тесты, опираясь на него. В каждом случае у тестирования есть особенности, обусловленные уровнем знаний о внутреннем устройстве программы. В зависимости от этого уровня выделяются три вида тестирования: чёрного, белого и серого ящика.

Метод чёрного ящика

Чёрный ящик — это система, внутреннее устройство и механизм работы которой сложны, неизвестны или не важны в рамках решения задачи.

Метод чёрного ящика — метод исследования, при котором вместо свойств и взаимосвязей составных частей системы изучается реакция системы как целого на изменяющиеся условия.

У чёрного ящика есть «вход» для ввода информации и «выход» для отображения результатов работы. При этом происходящие процессы во время работы системы наблюдателю неизвестны. Состояние выходов функционально зависит от состояния входов.

Тестирование чёрного ящика (black box testing) — тестирование, основанное на анализе функциональной или нефункциональной спецификации системы без знания внутренней структуры.

У тестировщика нет доступа к коду, он видит приложение как пользователь. Тестирование проводится через интерфейс приложения. Это ручное тестирование без знания, что находится «за кулисами» интерфейса.

Тест-дизайн, основанный на технике чёрного ящика, это написание или отбор тест-кейсов на основе анализа документации без знания внутреннего устройства программы.

Тестирование чёрного ящика находит ошибки:

1. Функции неправильно реализуются или их нет.
2. Интерфейс отличается от макетов.
3. Данные не записываются в базы или записываются неверно.
4. Недостаточная производительность системы.

Таким образом, тестировщик концентрируется на том, **что** программа делает, а не **как**.

Пример: тестировщик проводит тестирование веб-сайта, не зная особенностей его реализации. Он использует только предусмотренные разработчиком поля ввода и кнопки. Источник ожидаемого результата — спецификация (иными словами, техническое задание).

Преимущества чёрного ящика:

1. Тестирование производится с позиции пользователя, обнаруживает неточности и противоречия в поведении ПО.
2. Тестировщику не обязательно знать языки программирования.
3. Тестирование проводят независимые специалисты, что помогает избежать предвзятого отношения.
4. Тест-кейсы пишутся как только готова спецификация.

Недостатки чёрного ящика

1. Тестируется ограниченное количество сценариев.
2. Без чёткой спецификации трудно составить эффективные тест-кейсы.
3. Тесты избыточны, если их уже проверил разработчик на уровне модульного тестирования.

Метод белого ящика

Тестирование белого ящика (white box testing) основано на анализе внутренней структуры системы, на знании и понимании исходного кода. У тестировщика есть полный доступ к исходному коду.

Для тестирования методом белого ящика нужно знать язык программирования, на котором написано приложение. Обычно этот вид тестирования применяют разработчики при написании юнит-тестов. Входные значения отбираются на основе кода, который будет их обрабатывать.

Техника белого ящика применяется на разных уровнях тестирования, но главным образом для модульного тестирования компонентов.

Преимущества:

1. Тестирование производится на ранних этапах: пользовательский интерфейс не требуется.
2. Тестирование более тщательное, с покрытием путей выполнения программы (условий и операторов).

Недостатки:

1. Для тестирования нужны специальные знания. В первую очередь — язык программирования.

Метод серого ящика

Тестирование серого ящика (gray box testing) — тестирование в условиях, когда часть внутренней структуры программы известна. Тестировщик работает не с кодом приложения, а с часть его внутренней структуры: проверяет запись в базе данных, лог-файлы, коды ответа от сервера.

Для тестирования веб-приложений методом серого ящика тестировщик использует инструменты разработчика, например, Chrome DevTools.

Техника серого ящика применяется на интеграционном уровне для проверки взаимодействия компонентов программы, например, API-интерфейса и базы данных.

Пример: метод серого ящика используется при тестировании веб-сайтов на битые ссылки. Если тестер сталкивается с проблемой таких ссылок, он может сразу внести изменения в HTML-код и проверить в реальном времени.

Методы тестирования серого ящика:

1. **Матричное тестирование** — определение всех переменных, которые есть в программе.
2. **Тестирование ортогональных массивов** обеспечивает максимальное покрытие кода с минимальным количеством тестов.
3. **Pattern Testing** выполняется на данных истории предыдущих дефектов системы.

Виды тестирования после изменений в коде

На протяжении всей разработки в код приложения вносят изменения: при добавлении новых функций и при исправлении дефектов. В результате тестировщик снова проводит тестирование той части приложения, которая уже была проверена, но подверглась изменениям. В зависимости от изменений выделяют регрессионное и повторное тестирование.

Регрессионное тестирование (regression testing) — тестирование уже проверенной функциональности после изменений в коде. Цель — убедиться, что эти изменения не добавили или не активизировали ошибки в изменённых областях.

Повторное или подтверждающее тестирование (re-testing/confirmation testing) — исполняются тестовые сценарии, выявившие ошибки во время последнего запуска. Цель — подтвердить, что ошибки исправили, и приложение работает в соответствии с требованиями.

Повторное тестирование — обязательный этап. Тестировщик должен проверить, исправлен ли дефект, повторив сценарий, который выявил ошибку.

Статическое и динамическое тестирование

В зависимости от того, запускается код программы или нет, выделяют два виды тестирования — статическое и динамическое.

Статическое тестирование (static testing) — тестирование системы на уровне спецификации или реализации без исполнения кода. Так проводится тестирование:

- документации — требований, схем баз данных, тест-кейсов;

- кода приложения — проверка кода перед запуском специалистом, не участвовавшем в его написании или изменении, то есть аудит кода, или code review;
- параметров настройки среды приложения;
- подготовленных тестовых данных;
- прототипов пользовательского интерфейса.

Статическое тестирование начинается на ранних этапах жизненного цикла ПО и продолжается на протяжении всей разработки.

Динамическое тестирование (dynamic testing) — тестирование во время выполнения программного обеспечения, компонента или системы. Проверка — реальное поведение ПО во время его работы.

Чтобы выполнить динамическое тестирование нужно, чтобы код программы запустился. Тестируется как система в целом, так и отдельные компоненты. Все виды функционального тестирования — динамические.

Позитивное и негативное тестирование

Тестирование можно делать по инструкции и проверять, работает ли приложение согласно требованиям. А можно намеренно совершать некорректные действия, чтобы понять, как оно будет реагировать. Первый вид тестирования называют позитивным, а второй — негативным.

Позитивное тестирование (positive testing) — тестирование с использованием только корректных данных. Проверяет, правильно ли приложение выполняет вызываемые функции. Проводится, чтобы подтвердить работоспособность объекта тестирования.

Тестировщик полностью следует требованиям и инструкции по работе с приложением. Например, при тестировании формы регистрации заполняет её корректными данными и нажимает кнопку «Зарегистрироваться».

Негативное тестирование (negative testing) направлено на исследование работы приложения в ситуациях, когда выполняют некорректные операции или используют данные, потенциально приводящие к ошибкам.

Негативное тестирование — это не попытка «сломать» систему, а проверка системы на правильность обработки некорректных действий пользователя.

1. Если пользователь при регистрации укажет email без символа @, приложение выведет сообщение об ошибке.
2. Если сообщение об ошибке не появится, и пользователь не зарегистрируется, это будет считаться дефектом.
3. Если банковское приложение для выдачи кредита требует, чтобы возраст заёмщика был больше или равен 18, негативным тестом будет проверка возраста 15 лет, а успешным завершением — сообщение о невозможности выдать кредит из-за возраста, не соответствующего требованиям.

Позитивные тесты проводятся, чтобы убедиться в правильной работе приложения. После этого переходят к негативным проверкам.

Альфа- и бета-тестирование

В зависимости от того, кто выполняет тестирование перед выпуском продукта на рынок, выделяют альфа- и бета-тестирование.

Альфа-тестирование (alpha testing) — внутреннее пробное использование. Выполняется внутри организации-разработчика, иногда — с частичным привлечением пользователей.

Альфа-тестирование проводится после модульного, интеграционного и системного тестирования, когда продукт уже частично готов к выпуску на рынок, но нужно его доработать. Представляет собой имитацию реального использования, но выполняется либо командой тестирования, либо другими сотрудниками компании-разработчика в тестовой среде. Например, на тестовых стендах, недоступных внешним пользователям.

После внутреннего альфа-тестирования выпускают бета-версию продукта и передают её на внешнее (публичное) бета-тестирование.

Бета-тестирование (beta testing) выполняется вне организации с активным привлечением пользователей. Обычно представляет собой форму внешнего приёмочного тестирования.

Продукт должен быть стабилен, но не исключено появление проблем и выявление недостатков. Поэтому сначала доступ открывают для небольшой группы лояльных пользователей, чтобы проверить работоспособность и получить обратную связь.

Часто при тестировании игр применяют ОБТ — открытое бета-тестирование. Привлекают либо всех желающих (например, заполнившие заявку), либо людей с

опытом в играх такого типа. Обычно у компаний есть контакты тех, с кем они постоянно сотрудничают при проведении бета-тестирования.

Иногда бета-версия размещается в конкретной стране или регионе, чтобы собрать статистику или получить обратную связь прежде, чем полность вывести продукт на рынок.

Тестирование на основе тест-кейсов и исследовательское тестирование

Тестирование на основе тест-кейсов (scripted testing, test case based testing) — формализованный подход, при котором тестирование проводится на основе заранее подготовленных тест-кейсов. Часто используется на проектах по разработке ПО, так как позволяет структурировать процесс тестирования и сделать его контролируемым.

Исследовательское тестирование (exploratory testing) — частично формализованный подход, при котором тестировщик работает с приложением по сценарию. В процессе сценарий дорабатывается для более полного исследования приложения.

Во время исследовательского тестирования неформальные (не созданные заранее) тестовые сценарии разрабатываются, выполняются, анализируются и оцениваются динамически. Результаты тестирования используют для изучения компонента или системы и последующей разработки тестовых сценариев для непокрытых областей.

Исследовательское тестирование проводится сессиями. Сессия — это выделенный промежуток времени, в котором тестировщик исследует программу, ориентируясь на поставленную цель. Например, требуется проверить все поля ввода на странице. Во время сессии ведётся протокол, а тестировщик фиксирует действия и результаты.

Исследовательское тестирование лучше всего подходит, когда документация недостаточная или вовсе отсутствует, в условиях сжатых сроков и как дополнение к другим, более формальным методам тестирования.

Свободное (интуитивное) тестирование (ad hoc testing) — неформальный подход, в котором не предполагается использование тест-кейсов, чек-листов и сценариев. Тестировщик полностью опирается на свой профессионализм и интуицию при спонтанном выполнении проверок.

Часто при таком подходе предполагается, что тестировщик плохо знаком с тестируемым приложением. Этот вид тестирования используется редко и только как дополнение к полностью или частично формализованному тестированию, когда для исследования некоторых функций приложения нет тест-кейсов, или они ещё не написаны.

Смоук-тесты

Дымовой тест или смоук - тест (от англ. smoke - дым) — это тип тестирования программного обеспечения, который проверяет, правильно ли работают наиболее важные функции программного приложения. Термин «дымовое тестирование» происходит от идеи тестирования приложения ровно настолько, чтобы увидеть, «горит» ли оно (т. е. сильно ли сломано), прежде чем проводить более глубокое тестирование.

Дымовые тесты обычно запускаются после сборки, чтобы убедиться, что приложение может быть запущено и работают самые основные функции, прежде чем будет выполнено более всестороннее тестирование.

Дымовое тестирование может быть как ручным, так и автоматизированным.

Ручное дымовое тестирование — это процесс, при котором тестировщики вручную выполняют набор предопределенных тестовых случаев для проверки основных функций приложения. Этот метод предполагает ручное вмешательство, когда тестировщики физически выполняют шаги в приложении и наблюдают за результатами.

Автоматизированное дымовое тестирование использует программные инструменты для автоматического выполнения набора предопределенных тестовых случаев. Тестовые примеры написаны на языке сценариев, таком как Python или Java, и автоматически выполняются инструментом тестирования. Подробнее про автоматизированное тестирование мы поговорим в соответствующих курсах.

Этапы дымового теста могут варьироваться в зависимости от конкретного приложения и целей теста, но обычно включают следующее:

- **Проверка сборки:** убедитесь, что сборка завершена и может быть установлена в целевой среде.
- **Настройка среды:** убедитесь, что тестовая среда настроена правильно и соответствует необходимым требованиям.

- **Выполнение теста:** запустите наиболее важные и часто используемые функции приложения, чтобы определить, работают ли они должным образом.
- **Оценка результатов испытаний:** проанализируйте результаты дымовых испытаний и определите, произошли ли какие-либо сбои.
- **Отчет о дефектах:** задокументируйте любые сбои или дефекты, обнаруженные во время дымового теста.
- **Очистка:** удалите все временные файлы и восстановите тестовую среду до исходного состояния.

Примечание. Конкретные этапы дымового теста могут различаться в зависимости от процесса разработки программного обеспечения и используемой методологии тестирования.

Помните, что **цель дымового теста** — быстро выявить основные проблемы с приложением и обеспечить уверенность в том, что приложение достаточно стабильно, чтобы приступить к дальнейшему тестированию.

Нефункциональное тестирование

Нефункциональное тестирование (non-functional testing) — анализ свойств компонента или системы, не относящихся к функциональности. То есть, проверяется, «как работает система».

Нефункциональное тестирование включает:

1. Тестирование производительности:
    a. нагрузочное тестирование,
    b. тестирование масштабируемости,
    c. объёмное тестирование,
    d. стрессовое тестирование.
2. Тестирование безопасности.
3. Инсталляционное тестирование.
4. Тестирование интерфейса (GUI/UI-тестирование).
5. Тестирование удобства использования.

6. Тестирование локализации.
7. Тестирование надёжности.

Тестирование производительности

Тестирование производительности (performance testing) помогает определить работоспособность, стабильность, потребление ресурсов в условиях разных сценариев использования и нагрузок.

Задача системы — обрабатывать нужное количество данных за установленное время. В случае превышения запланированных объёмов входных данных, система восстанавливается после отказа без потери данных.

Пример: в требованиях указано, что система обрабатывает тысячу запросов пользователей в секунду без потери производительности. Чтобы проверить выполнение этого требования, тестировщик формирует тысячу запросов пользователей и направляет их на сервер.

Нагрузочное тестирование

Нагрузочное тестирование (load testing) — тип тестирования производительности, цель которого — оценить поведение системы при возрастающей нагрузке, а также определить нагрузку, которую может выдержать компонент или система.

Нагрузка повышается, пока не будут достигнуты нужные характеристики. Затем отслеживается поведение на протяжении повышения загрузки системы. При этом:

- измеряют время выполнения операций при определённой интенсивности этих операций;
- определяют количество пользователей, одновременно работающих с приложением;
- определяют границы приемлемой производительности при увеличении нагрузки и при увеличении интенсивности выполнения операций.

Тестирование масштабируемости

Тестирование масштабируемости (scalability testing) — тестирование для измерения возможностей вертикального и горизонтального масштабирования с точки зрения любой из нефункциональных возможностей: увеличение количества пользователей, рост количества транзакций, увеличение объёма данных.

Вертикальное масштабирование — это увеличение производительности каждого компонента системы для повышения общей производительности. Например, увеличивается объём оперативной памяти на сервере, чтобы он быстрее обрабатывал запросы. Это повысит производительность всей системы.

Горизонтальное масштабирование — разбиение системы на структурные компоненты и разнесение их по отдельным физическим машинам. А также увеличение количества серверов, параллельно выполняющих одну и ту же функцию. Например, увеличивается количество серверов, но каждый выполняет одну и ту же задачу: принимает одни и те же запросы и отвечает на них.

Если разработчики заранее не подумают, как они увеличат ресурсы при росте популярности, они потеряют значительную часть прибыли.

Объёмное тестирование

Объёмное тестирование (volume testing) — тестирование на больших объёмах данных. Например, тестируется поведение приложения при попытке загрузить в его базу данных нескольких файлов очень большого размера.

Стрессовое тестирование

Стрессовое тестирование (stress testing) оценивает систему на граничных значениях рабочих нагрузок, за их пределами или в состоянии ограниченных ресурсов — памяти или доступа к серверу.

Например, стандартная нагрузка на сервер приложения — 1000 запросов в секунду. При стрессовом тестировании нужно проверить её поведение при увеличении нагрузки до 10 000 запросов в секунду. Если система не обработает такое количество запросов и отключится, при перезапуске все данные и настройки сохранятся.

Основные понятия при тестировании производительности

Время задержки (Latency) — временной интервал между запросом и ответом. Если говорят, что у сервиса время задержки составляет 100ms, значит, ему требуется 100 миллисекунд на обработку запроса и ответ. Чем ниже время задержки, тем лучше клиентский опыт.

Время ответа (Response time) — время, нужное для ответа на запрос.

Пропускная способность (Throughput) — фактическое количество запросов, которое обрабатывает система за определённое время. Метрика пропускной способности показывает объём данных, полученных и обработанных в момент времени.

Важно не отделять показатели времени задержки от пропускной способности. Высокий показатель времени задержки часто напрямую связан с увеличением показателей метрики пропускной способности. Пропускная способность обычно измеряется в rps — количестве запросов в секунду (requests per second).

Ширина пропускания канала (Bandwidth) — максимальное число запросов, обрабатываемых системой. Используется, чтобы измерять максимальный объём, который обрабатывает приложение.

Процент ошибок — отношение невалидных ответов к валидным за промежуток времени.

Инсталляционное тестирование (installation testing) — тестирование, направленное на проверку успешной установки и настройки, обновления или удаления приложения при различном программном и аппаратном окружении. Оно позволяет оценить работоспособность системы после завершения работы инсталлятора.

Тестирование пользовательского интерфейса — проверка соответствия интерфейса и требований, насколько удобно пользователям работать с программным продуктом. Проверяют, ведёт ли себя программное обеспечение в соответствии со спецификацией, когда пользователь взаимодействует с ним с помощью клавиатуры и мыши (когда тестируется десктопное приложение), или с помощью сенсорного экрана, жестов или движений устройства (когда тестируется мобильное приложение).

UX-тестирование (юзабилити-тестирование, Usability testing) — проверка того, насколько легко пользователь понимает и осваивает программу, включая функциональную составляющую (саму систему) и её документацию — пользовательские инструкции.

Результат грамотного UX-тестирования — перечень рекомендаций: что и как изменить, чтобы повысить количество конверсий и превратить посетителей сайта в постоянных и преданных пользователей.

Тестирование локализации (localization testing) — проверка адаптации программного обеспечения для нового места эксплуатации. Включает проверку изменения языка и культурной адаптации.

Например, проверяют, насколько перевод приложения на русский язык корректен с точки зрения орфографии, грамматики, культурных особенностей. Проверка затрагивает все части приложения, в том числе названия кнопок, всплывающие подсказки, надписи над полями. Это касается приложений, которые планируется внедрять в разных странах.

Тестирование безопасности (security testing) — тестирование программного продукта на предмет его защищённости. Основные понятия, которые охватывает тестирование: конфиденциальность, целостность и сохранность данных, аутентификация, авторизация и невозможность отказа от авторства (атрибуты качества). Проводится для тех объектов, в работе которых обеспечение защищённости — одна из важнейших задач.

Тестирование надёжности (reliability testing) — тестирование способности приложения выполнять свои функции в заданных условиях на протяжении заданного времени или установленного количества операций.

Неважно, как долго идёт это тестирование, основная задача — наблюдая за потреблением ресурсов в течение определённого времени, выявить утечки памяти и проследить, чтобы скорость обработки данных или время отклика в начале теста и с течением времени не уменьшалась. Иначе вероятны сбои в работе продукта и перезагрузки системы.

Контрольные вопросы

1. Что такое функциональное тестирование?
2. Чем статическое тестирование отличается от динамического?
3. Что такое позитивное тестирование? Что такое негативное тестирование?
4. Что такое нефункциональное тестирование?
5. Какие виды входят в нефункциональное тестирование?

Дополнительные материалы

1. Классификация видов тестирования
2. Поговорим о нагрузочном тестировании

3. 50 оттенков нагрузочного тестирования
4. Обзор инструментария для нагрузочного и перформанс-тестирования
5. Тестирование безопасности: изнутри и снаружи
6. Гайд по тестированию локализации и интернационализации, а также большой и полезный checklist