Тестирование мобильных приложений

Отличия мобильных приложений от декстопных и веб-приложений

- Воздействие на работу различнiх факторов (прерывания в работе)
- Особенности беспроводной сети
- Уведомления
- Ориентация экрана
- Установка (маркеты)
- Особенности прошивок
- Множество каналов ввода
- Биометрия
- Энергопотребление
- Ограниченная память
- Обновления
- Локализация
- Размер дисплея

Прерывания:
- ✓ оповещения мобильного устройства (входящие звонки, состояние батареи и др.)
- ✓ оповещения других мобильных приложений (например, мессенджеров)
- ✓ блокировка экрана
- ✓ подключение зарядки, наушников…
- ✓ и др. …

Мобильные приложения должны соответствовать правилам и ограничениям площадок, на которых они размещаются.

Тач-интерфейс, мультитач, жесты, голос…

Частые обновления

Типы мобильных приложений

1. Нативные приложения

Написаны на родном для определенной платформы языке программирования. У них высокая скорость работы, и они позволяют задействовать разные функции телефона — например, камеру, датчики и т.д.. Могут всецело или частично обходиться без наличия интернет-соединения.

2. Веб-приложения

Представляют собой сайты, которые адаптированы и оптимизированы под любой смартфон. Веб-приложения способны функционировать, независимо от платформы девайса. Недостатки: необходимо интернет-соединение, производительность зависит от качества интернет-соединения. Веб-приложения не могут получать доступ к функциям устройства.

3. Гибридный тип

Это веб-приложение в обертке нативного приложения. Нативный “фундамент” в виде браузерной оболочки даёт преимущества нативных приложений: доступ к функционалу смартфона и т.п., а сторона веб-приложений дает плюсы в виде кроссплатформенности и простоты обновления контента. Компания, имеющая веб-приложение, может практически “на коленке” собрать гибридные приложения для основных платформ и обеспечить себе присутствие в маркете и на рабочем столе клиентов.

4. Кроссплатформенные приложения. Разрабатываются с помощью кроссплатформенных фреймворков: React Native (JavaScript), Ionic (JavaScript), Xamarin (.NET and C#) и т.п. и имеют общий код для iOS и Android.

Платформы для тестирования

1. Реальные устройства
2. Симуляторы

   Симуляторы воспроизводят поведение системы и её интерфейса. Симуляции имитируют выполнение кода. В большинстве случаев, для запуска симулятора, используются XCode или Android Studio.

3. Эмуляторы

   Это устройство, компьютерная программа или система, которая принимает те же самые входные данные и выдает те же самые выходные данные, что и данная система. Эмулятор представляет собой полную повторную реализацию конкретного устройства или платформы изолированно внутри нашей хост-системы.

Главный недостаток симуляторов и эмуляторов в том, что они не могут полностью заменить физический девайс. С их помощью нельзя проверить корректную работу приложения при взаимодействии с остальными системами телефона, то есть как будет функционировать приложение при медленном интернете, прерываниях, использовании камеры, микрофона или геолокации. Работа приложения на симуляторе и эмуляторе может отличаться от работы на реальном девайсе ещё и из-за мощности компьютера, на котором выполняется тестирование.

Подбор девайсов для тестирования мобильных приложений

1. Определить на какой платформе будет разрабатываться приложение. Или оно будет кроссплатформенным?
2. Проанализировать целевую аудиторию.
3. В каком географическом регионе будет распространяться приложение.
4. Изучить тенденции на рынке (самые популярные устройства в крупных магазинах, сервисы статистических данных, например https://gs.statcounter.com/vendor-market-share/mobile, https://www.appbrain.com/stats/top-android-phones)
5. Если приложение опубликовано, то изучить статистику использования, в первую очередь важны те девайсы, которыми пользуются ваши клиенты..
6. По версиям операционных систем в идеале иметь все версии начиная с минимальной поддерживаемой вашим приложением.
7. На Android особое внимание нужно уделить производителям, так как каждый из них имеет свои особенности, такие как отсутствие google services на устройствах от huawei

Матрица мобильных устройств - это таблица мобильных устройств, поддерживаемых приложением, на которых будут проводиться различные виды тестов.

Матрица формируется с учетом:

- ✓ вида мобильного приложения
- ✓ категории, к которой относится приложение
- ✓ пожеланий заказчика

Вид мобильного приложения

Для веб приложений:

- размер экрана
- разрешение экрана
- веб движок

Для нативных приложений:

- размер экрана
- разрешение экрана
- объем оперативной памяти
- ядра
- графический процессор
- версия ОС
- особенности мобильного приложения

Как категории мобильного ПО влияют на выбор устройств:

- ***Мессенджеры***: для них важны различные типы связи
- ***Игры***: для них важен графический процессор
- ***Навигация***: для них важна система позиционирования (GPS, ГЛОНАСС)

Как пожелания заказчика могут влиять на выбор устройства:

Устаревшие платформы
Специфические мобильные устройства

1	Model	OS version	Screen size	Screen resolution	Bluetooth
2	**iOS Phones**				
3	IPhone 4S	7.1	3,5 in	960x640	4.0
4	iPhone 5S	7.1	4 in	1136x640	4.0
5	IPhone 6+	8+	5,5 in	1920x1080	4.0
6	IPhone 6S	9.0	4,7 in	1334x750	4.2
7	**iOS Tablets**				
8	iPad Air	8+	9,7 in	2048x1536	4.0
9	iPad Pro	9.1	12,9 in	2732x2048	4.2
10	iPad Mini	7.1	7,9 in	1024x768	4.0
11					
12	Model	OS version	Screen size	Screen resolution	Bluetooth
13	**Android Phones**				
14	Galaxy S5	6.0 Marshmallow	5,1 in	1920x1080	4.0
15	Galaxy S3	4.3 Jelly Bean	4,8 in	1280x720	4.0
16	Galaxy Note 3	4.3 Jelly Bean	5,7 in	1280x720	4.0
17	Galaxy Note 4	4.4 KitKat	5,7 in	2560x1440	4.1
18	LG G3	5.1 Lollipop	5,5 in	2560x1440	4.0
19	LG Leon	5.0 Lollipop	4,5 in	854x480	4.0
20	**Android Tablets**				
21	Samsung Galaxy Tab S2	5.0 Lollipop	9,7 in	2048x1536	4.1
22	Samsung Galaxy Tab S	4.4 KitKat	10,5 in	2560x1600	4.0
23	Samsung Nexus 10	4.3 Jelly Bean	10,1 in	2560x1600	4.0
24	HTC Nexus 9	6.0 Marshmallow	8,9 in	2048x1536	4.1

← **1. Составили матрицу**

2. Далее решаем где взять данные устройства

Это могут быть:

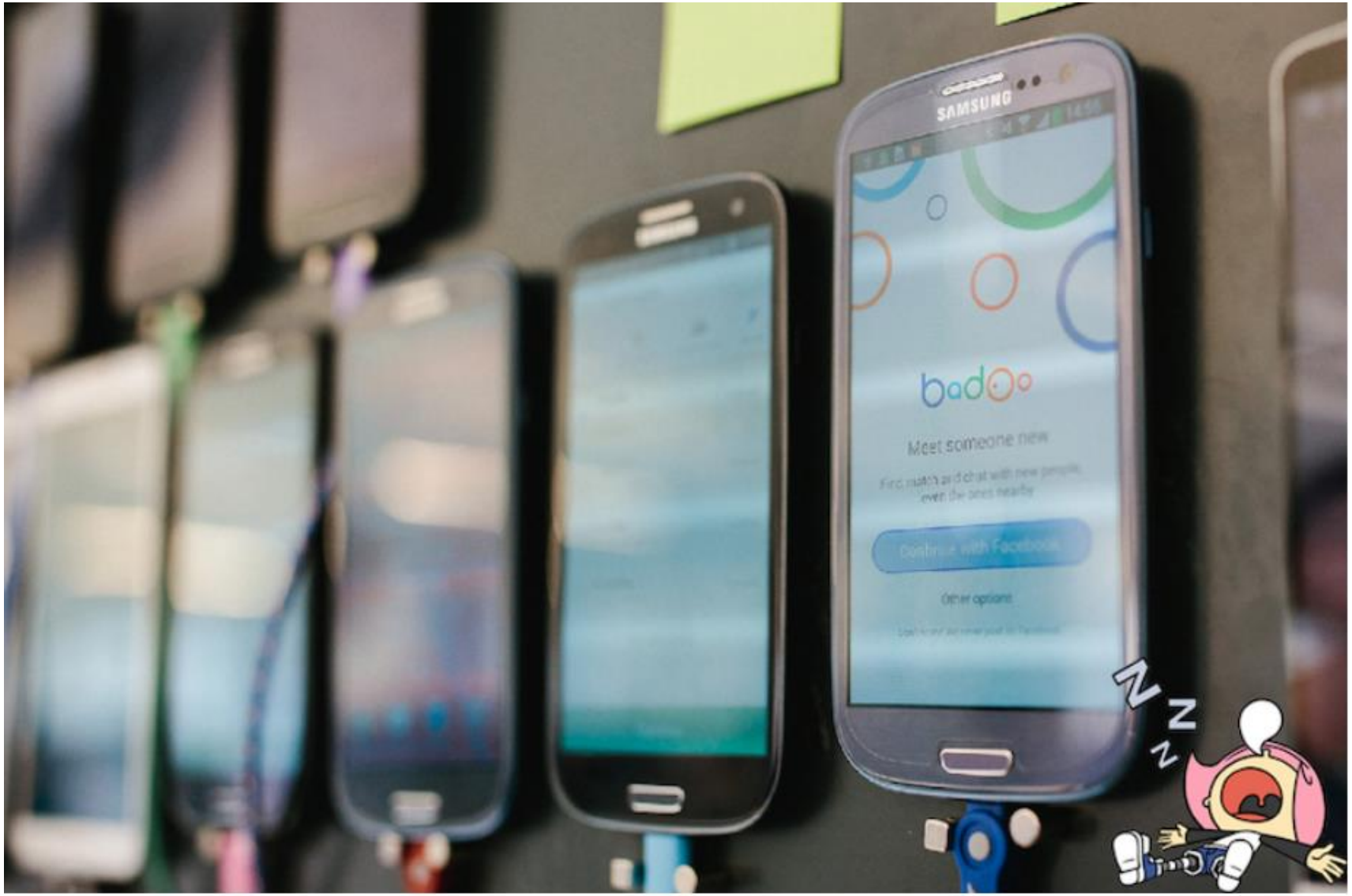
Реальные

- На руках
- Тестовые фермы для удаленного тестирования

Виртуальные:

- Эмуляторы и симуляторы на своих мощностях
- Облачные сервисы

Стенд для автоматизации тестирования



Тестировщик спит, автотесты работают!

Тестовая ферма устройств Яндекса (свыше 800 устройств по данным на 2022год)

Инструменты для тестирования мобильных приложений

1. Эмуляторы устройств
У iOS — это симулятор Apple iOS, для Android — Android Virtual Device.

2. DevTools
Тестировать мобильное веб-приложение можно в обычном браузере на компьютере с помощью инструмента DevTools. Он помогает проверить адаптивность вёрстки, смену ориентации экрана, разные скорости интернет-соединения.

3. Сервисы TestFlight и Beta
Для бета-тестирования мобильных приложений используют сервисы TestFlight для iOS и Beta для Android.

4. Снифферы
Для тестирования взаимодействия с бэкендом — частью приложения, работающей на сервере, — применяют снифферы. **Сниффер** — это анализатор трафика, то есть всей информации, которая проходит через компьютерные сети. С его помощью можно проверять http-запросы, различные коды ответов и реакцию приложения на них. Самые популярные снифферы — Fiddler и Charles.

Чек-лист тестирования мобильных приложений состоит из следующих разделов:

- Функциональное тестирование;
- Тестирование совместимости;
- Тестирование безопасности;
- Тестирование локализации и глобализации;
- Тестирование удобства использования;
- Стрессовое тестирование;
- Кросс-платформенное тестирование;
- Тестирование производительности.

Функциональное тестирование:

1. Установка/удаление/накатка версий;
2. Запуск приложения (отображение Splash Screen);
3. Работоспособность основного функционала приложения;
   - Авторизация (по номеру телефона/через соц. сети/e-mail);
   - Регистрация (по номеру телефона/через соц. сети/e-mail);
   - Онбординг новых пользователей;
   - Валидация обязательных полей;
   - Навигация между разделами приложения;
   - Редактирование данных в профиле пользователя;
   - Проверка оплаты;
   - Тестирование фильтров;
   - Бонусы;
4. Корректное отображение ошибок;
5. Работа с файлами (отправка/получение/просмотр);
6. Тестирование тайм-аутов;
7. Тестирование заглушек (не соединения с интернетом/нет, например, товаров и т.д);
8. Тестирование pop-up, алертов;
9. Тестирование WebView;
10. Скролл/свайп элементов;
11. Тестирование PUSH уведомлений;
12. Сворачивание/разворачивание приложения;
13. Разные типы подключений (сотовая связь/Wi-Fi);
14. Ориентация экрана (альбомная/портретная);
15. Темная/светлая темы;
16. Реклама в приложении;
17. Шаринг контента в соц. сети;
18. Работа приложения в фоне;
19. Работа приложения в режиме сна и lock/unlock девайса;
20. Пагинация страниц;
21. Политики конфиденциальности и прочие ссылки на документы.

Тестирование совместимости.

Тестирование совместимости используется, чтобы убедиться, что ваше приложение совместимо с другими версиями ОС, различными оболочками и сторонними сервисами, а также аппаратным обеспечением устройства.

1. Корректное отображение гео;
2. Информации об операциях (чеки и т.д.);
3. Различные способы оплаты (Google Pay, Apple Pay);
4. Тестирование датчиков (освещенности, температуры устройства, гироскоп и т.д.);
5. Тестирование прерываний (входящий звонок/смс/push/будильник/режим «Не беспокоить» и т.д.);
6. Подключение внешних устройств (карта памяти/наушники и т.д.).

Тестирование безопасности.

Данная проверка нацелена на поиск недостатков и пробелов с точки зрения безопасности приложения.

1. Тестирование разрешений (доступ к камере/микрофону/галерее/и т.д.);
2. Данные пользователя (пароли) не передаются в открытом виде;
3. В полях, с вводом пароля и подтверждением пароля, данные скрываются астерисками.

Тестирование локализации и глобализации.

Тестирование интернационализации/глобализации приложения включает тестирование приложения для различных местоположений, форматов дат, чисел и валют, а также замену фактических строк псевдостроками. Тестирование локализации включает тестирование приложения с локализованными строками, изображениями и рабочими процессами для определенного региона.

Что проверяем?

1. Все элементы в приложении переведены на соответствующий язык;
2. Тексты зашиты внутри приложения и пользователь в настройках приложения может выставить необходимый язык;
3. Тексты зависят от языка в системных настройках;
4. Тексты приходят с сервера;
5. Корректное отображение форматов дат (ГОД - МЕСЯЦ - ДЕНЬ или ДЕНЬ - МЕСЯЦ - ГОД.);
6. Корректное отображение времени в зависимости от часового пояса.

Тестирование удобства использования

Тестирование удобства использования помогает удостовериться в простоте и эффективности использования продукта пользователем, с целью достижения поставленных целей. Иными словами, это не что иное, как тестирование дружелюбности приложения для пользователя.

Что проверяем?

1. Корректное отображение элементов на устройствах с различными разрешениями экранов;
2. Все шрифты соответствуют требованиям;
3. Все тексты правильно выровнены;
4. Все сообщения об ошибках верные, без орфографических и грамматических ошибок;
5. Корректные заголовки экранов;
6. В поисковых строках присутствуют плейсхолдеры;
7. Неактивные элементы отображаются серым;
8. Ссылки на документы ведут на соответствующий раздел на сайте;
9. Анимация между переходами;
10. Корректный возврат на предыдущий экран;
11. Поддерживаются основные жесты при работе с сенсорными экранами (swipe back и т.д.);
12. Пиксель-перфект.

Стрессовое тестирование.

Стрессовое тестирование направлено на определение эффективности производительности приложения в условиях повышенной нагрузки. Стресс-тест в этом контексте ориентирован только на мобильные устройства.

Что проверяем?

1. Высокая загрузка центрального процессора;
2. Нехватка памяти;
3. Загрузка батареи;
4. Отказы;
5. Низкая пропускная способность сети;
6. Большое количество взаимодействий пользователя с приложением (для этого может понадобиться имитация реальных условий состояния сети).

Кросс-платформенное тестирование

Важный вид тестирования, который необходимо проводить для понимания того, будет ли должным образом отображаться тестируемый продукт на различных платформах, используемых целевой аудиторией.

1. Работоспособность приложения на различных устройствах разных производителей

Тестирование производительности.

Если пользователь устанавливает приложение, и оно не отображается достаточно быстро (например, в течение трех секунд), оно может быть удалено в пользу другого приложения. Аспекты потребления времени и ресурсов являются важными факторами успеха для приложения, и для измерения этих аспектов проводится тестирование производительности.

1. Время загрузки приложения;
2. Обработка запросов;
3. Кэширование данных;
4. Потребление ресурсов приложением (например расход заряда батареи).

Основные дефекты приложения при тестировании

1. **Не работает адаптивная вёрстка**
2. **Сбой работы приложения**

   внезапная остановка работы приложения:

   приложение некорректно отзывается на действия пользователя.
3. **Неконтролируемое потребление ресурсов**

   Приложение потребляет слишком много заряда батареи устройства, и оно нагревается.
4. **Несоблюдение принципов безопасности**

   Дефект заключается в том, что приложение может не иметь никакого отношения к контактам в телефоне, но всё равно запрашивать к ним доступ.

   Поэтому при тестировании надо обязательно отслеживать функции, когда приложение запрашивает доступ к контактам, фото, геопозиции и работе других приложений.
5. **Локализация**

   Вёрстка часто портится после перевода контента на другой язык. Например, в японском приложении для российского рынка может быть не адаптировано поле для текста. Там, где требуется три иероглифа, предложение на русском языке не поместится, и наоборот.

   Есть страны с жёсткой цензурой, и часть контента приходится полностью менять при локализации. Нужно следовать традициям и правилам, которые действуют в конкретной стране, чтобы не получить негативные отзывы от пользователей и заблокированное приложение.

Баги, которые можно встретить на каких-то девайсах, но не получить на других, можно условно разделить на три категории:

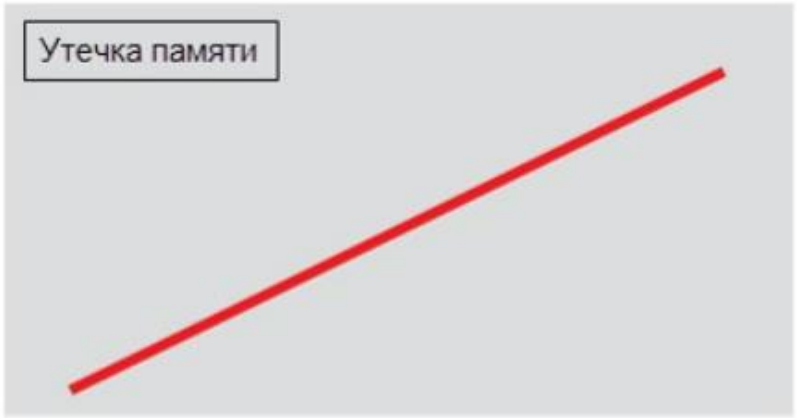
- ✓ связанные с производителем устройства – например, нестандартный API камеры или кастомная системная библиотека;
- ✓ связанные с версией ОС Android – например, несовместимые API и проблемы производительности;
- ✓ связанные с размером/ разрешением экрана, чипсетом и прочими различиями в железе.

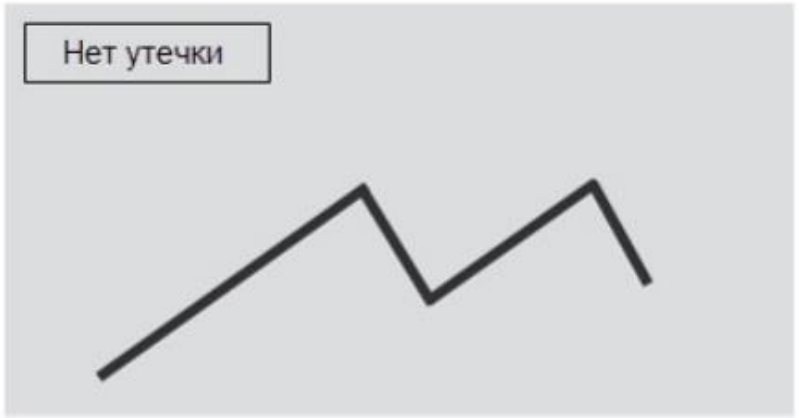
Перед публикацией приложение должно быть протестировано на соответствие правилам площадки

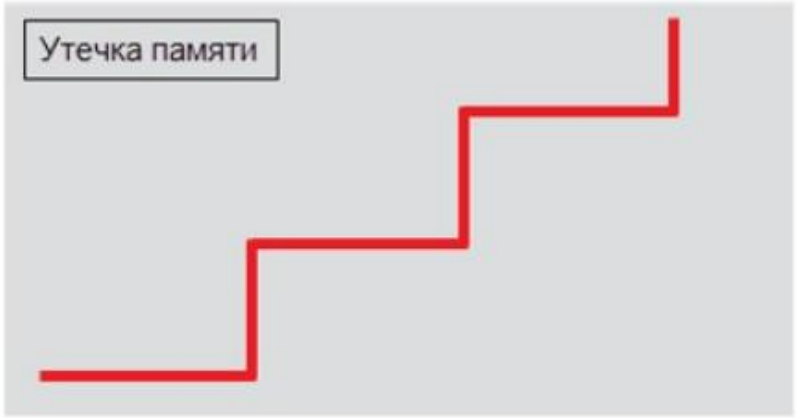
Подготовка к Google Play Review

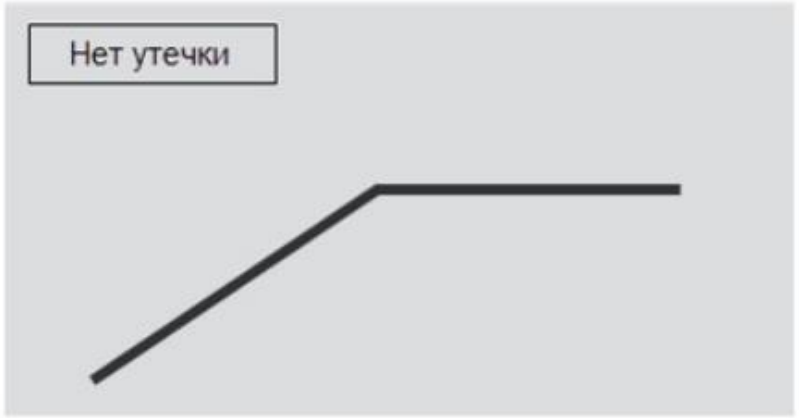
- Проверить, что мобильное приложение соответствует Google Play Policy > Restricted Content;
- Проверить, что metadata приложения соответствует Metadata Policy (Store Listing and Promotion > Metadata);
- Картинки и фотографии, используемые в приложении должны соответствовать правилам на использование и копирование;
- Приложение должно иметь Rating. Необходимо заполнить опросник на Play Console;
- Если в приложении используются данные о пользователях, необходимо опубликовать Privacy Policy;
- Если в приложении используется реклама, приложение необходимо проверить на соответствие Google Play Policy > Monetization and Ads;
- Если в приложении предполагается контент, который будет загружать конечный пользователь, в Terms of Use приложения должны быть включены пункты ограничивающие пользователя от запрещенного контента.

Утечки памяти - это неконтролируемое уменьшение объема свободной оперативной и виртуальной памяти, связанное с ошибками в работающих приложениях, не освобождающих ненужные участки памяти, которые становятся недоступными для других приложений и для него самого.

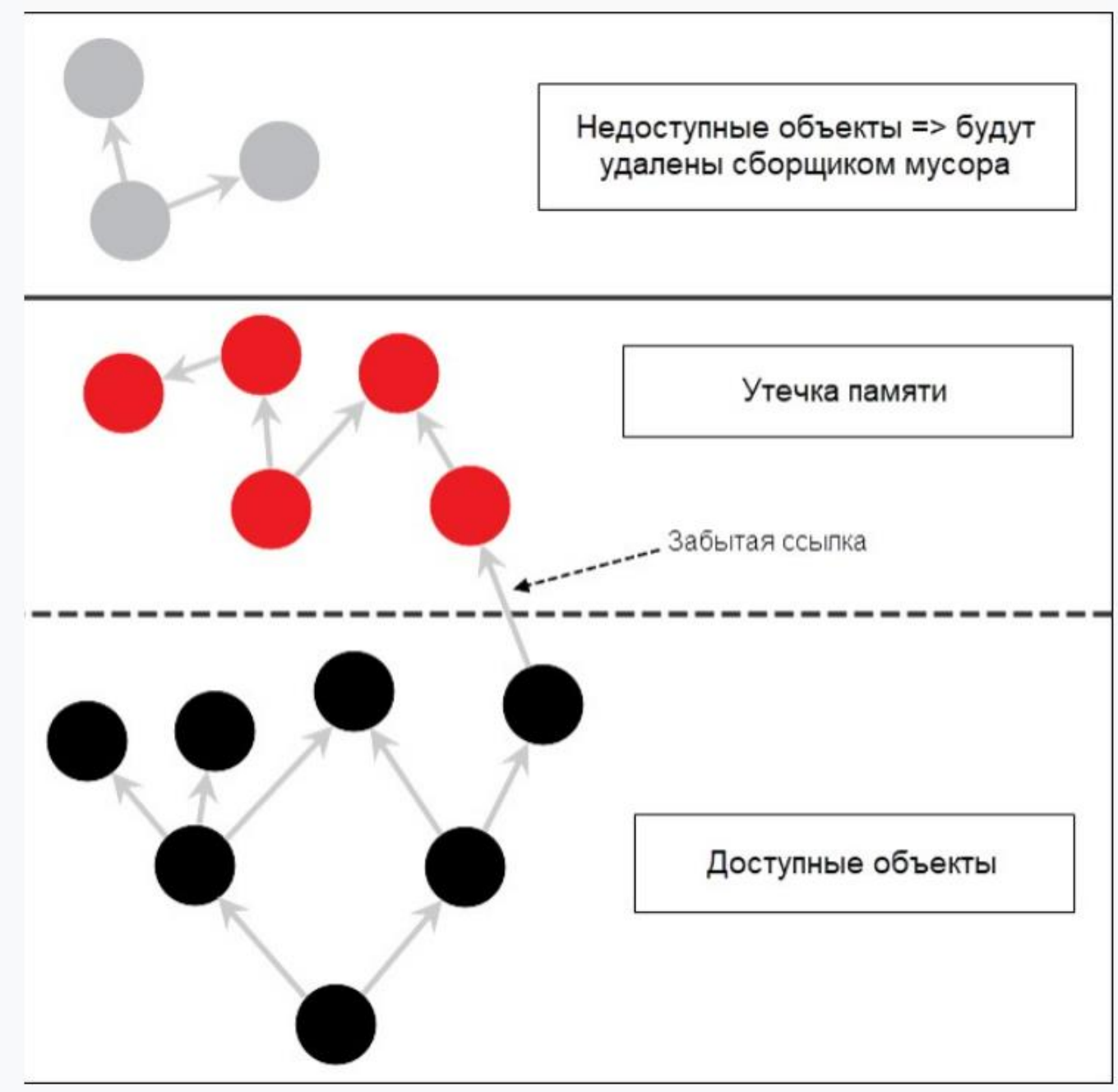








Потенциальная проблема для утечки памяти: Объект не будет уничтожен, если он доступен по цепочке ссылок, даже если он уже не используется.



Почему важно тестировать приложение на наличие утечек памяти:

- ✓ Незаметные утечки памяти в будущем могут привести к непредвиденным и трудно отлаживаемым проблемам.
- ✓ Не все приложения потребляют мало памяти, и не все девайсы выделяют много памяти, поэтому даже незначительные утечки могут вызывать проблемы в работе приложения.
- ✓ Если приложение запущено продолжительное время, то утечки в нем будут накапливаться.

Утечки памяти порождают ряд проблем:

- зависания
- снижение производительности
- аварийные остановки приложения (краш)
- проблемы работы с другими приложениями

Что такое утечка памяти в Java

https://javarush.com/groups/posts/6518-kofe-breyk-264-utechki-pamjati-java-kak-ikh-obnaruzhitjh-i-predotvratitjh

Утечки памяти в Java — языке, известном своей автоматической сборкой мусора, доставляют разработчикам немало хлопот. В отличие от языков, в которых управление памятью осуществляется самим программистом, Java автоматизирует это с помощью своего сборщика мусора. При этом важно учитывать, что такая автоматизация совершенно не освобождает Java-приложения от риска утечек памяти. Утечка памяти — это ситуация, когда объекты, которые больше не нужны приложению, продолжают удерживаться в памяти, поскольку на них есть ссылки в другом месте. В результате сборщик мусора не может очистить это пространство.

Наиболее частые причины утечек памяти в Java

1. **Статические ссылки.** Статические поля в Java связаны с классом, а не с отдельными экземплярами. Это означает, что они могут оставаться в памяти на протяжении всего срока службы приложения, если с ними внимательно не обращаться. Например, статическая коллекция, в которую продолжают добавляться элементы без своевременного удаления.

Наиболее частые причины утечек памяти в Java (продолжение)

2. **Прослушиватели и обратные вызовы (Listeners и Callbacks).** В Java, особенно в приложениях с графическим интерфейсом или тех, которые используют шаблон наблюдателя (observer), прослушиватели и обратные вызовы являются достаточно обычным явлением. Если эти прослушиватели не отменяются, когда они больше не нужны, то они могут препятствовать сбору мусора для объектов, а это тоже приводит к утечкам памяти.
3. **Кэшированные объекты.** Объекты, которые кэшируются и не удаляются должным образом, когда они больше не нужны, могут занимать значительный объем памяти, что также приводит к утечке.
4. **Неправильное использование коллекций.** Такие коллекции, как HashMap или ArrayList, имеют фундаментальное значение для программирования на Java. Однако неправильное управление коллекциями может привести к утечкам памяти. Например, добавление объектов в коллекцию и невозможность их удаления, когда они больше не нужны.
5. **Незакрытые ресурсы.** Такие ресурсы, как подключения к базе данных, сетевые подключения или потоки файлов, если они не закрыты должным образом, могут привести к утечкам памяти, потому что каждый открытый ресурс хранится в памяти.
6. **Внутренние классы.** Нестатические внутренние классы содержат неявную ссылку на свой внешний класс. Если их экземпляры передаются и сохраняются в приложении, они также могут непреднамеренно сохранить экземпляры своих внешних классов в памяти.


I heard you want to be a web developer
Here are a few devices to test your site