

Нефункциональное тестирование

Процесс тестирования
мобильных приложений



Оглавление

На этом уроке	3
Термины, используемые в лекции	3
Что такое нефункциональное тестирование?	3
Разница между функциональными и нефункциональными требованиями	4
Процесс нефункционального тестирования	5
Разница между функциональным тестированием мобильных и веб-приложений	6
Установочное тестирование	6
Тестирование безопасности	8
SQL-инъекция	9
Тестирование производительности	12
Производительность устройства	12
Производительность сервера/API	15
Производительность сети	16
Нагрузочное тестирование	20
Стрессовое тестирование	23
Тестирование локализации и глобализации	24
Подведение итогов	26
Домашнее задание	26
Используемая литература	27
Дополнительные материалы	27

На этом уроке

Прошлая лекция была посвящена функциональному тестированию мобильных приложений. Сегодня мы поговорим про нефункциональное.

- Разберемся, в чем разница между функциональными и нефункциональными требованиями.
- Рассмотрим виды нефункциональных тестов.
- Изучим Jmeter и составим в нем нагрузочный тест.

Термины, используемые в лекции

Нефункциональные требования — ограничения или требования, предъявляемые к системе. Определяют атрибут качества программного обеспечения.

SQL-инъекция — тип уязвимости системы безопасности, которая может возникнуть в приложениях, использующих базы данных.

Пакет — небольшая единица данных, которая передается по сети. Это стандартизированный формат для передачи данных по сетям, который обычно включает информацию о сетевых адресах отправителя и получателя, управляющую информацию и фактические передаваемые данные.

Потеря пакетов — явление, возникающее в компьютерных сетях, когда один или несколько пакетов данных не достигают своего назначения.

Что такое нефункциональное тестирование?

Нефункциональное тестирование — это тестирование нефункциональных требований: масштабируемости, производительности, безопасности, надежности и так далее.

Нефункциональные требования — это ограничения или требования, предъявляемые к системе. Они определяют атрибут качества программного обеспечения и помогают:

- сделать эффективной систему в целом и ее компоненты в частности;
- удовлетворить потребности пользователей;
- при необходимости — наложить ограничения или запреты на систему.

Нефункциональное тестирование



Схема крупнее: [виды тестирования](#)

Сегодня мы разберем основные виды нефункциональных тестов:

- установочное тестирование;
- тестирование безопасности;
- тестирование производительности (и его виды);
- стрессовое тестирование;
- тестирование локализации и глобализации.

Разница между функциональными и нефункциональными требованиями

Чтобы разница была наглядной, возьмем простые примеры функциональных и нефункциональных требований в разделе авторизации.

- **Функциональное требование** — у пользователя должна быть возможность авторизоваться.
- **Нефункциональное требование** — время авторизации после нажатия кнопки «Авторизоваться» должно быть не больше 5 секунд.

Функциональные требования	Нефункциональные требования
Описывают, что должен делать продукт (его функционал)	Описывают, как продукт работает
Покрывают все функции, которые должно выполнять приложение	Покрывают все аспекты хорошего пользовательского опыта
Обязательно должны быть	Необязательны, но желательны
Легче специфицировать	Сложнее специфицировать
Тестирование проводится в первую очередь	Тестирование проводится после тестирования функционала
Нужно хорошее понимание функций и требований программного обеспечения	Нужно хорошее понимание архитектуры, инфраструктуры и показателей производительности ПО

Функциональное тестирование фокусируется на поведении и функциях программного обеспечения, а нефункциональное — на производительности, удобстве использования, надежности и безопасности. Оба типа тестирования необходимы, чтобы обеспечить качество программного обеспечения и удовлетворить ожиданиям конечных пользователей.



Функциональное тестирование проверяет, соответствует ли программное обеспечение функциональным требованиям.

Нефункциональное тестирование проверяет, соответствует ли программное обеспечение нефункциональным требованиям.

Процесс нефункционального тестирования

Нефункциональное тестирование организуется по такому же принципу, что и функциональное, и проводится после него.

Для нефункциональных тестов мы точно так же составляем тест-кейсы, выбираем устройства для тестирования и нужные инструменты. Если ожидаемый результат не совпадает с фактическим, заводим баг-репорты.

Разница между функциональным тестированием мобильных и веб-приложений

Различия между нефункциональным тестированием веб-приложений и мобильных приложений заключаются в характеристиках каждой платформы.

- **Факторы, зависящие от платформы:** у веб-приложений и мобильных приложений разная архитектура, дизайн и технологии — все это влияет на способ тестирования. Например, мобильные приложения предназначены для небольших экранов и ограниченных ресурсов, а доступ к веб-приложениям осуществляется через браузеры, и они могут работать на разных устройствах.
- **Разница в подходах к тестированию производительности:** мобильные приложения можно тестировать на разных устройствах и в разных условиях сети, а веб-приложения — на разных браузерах и операционных системах.

Таким образом, у нефункционального тестирования веб-приложений и мобильных приложений много общего, но есть и различия, обусловленные характеристиками каждой платформы. Важно понимать их и адаптировать подход к тестированию, чтобы обеспечить наилучшее взаимодействие с пользователем и производительность приложения.

А теперь — перейдем к видам нефункциональных тестов.

Установочное тестирование

Установочное тестирование проверяет, что процесс установки проходит гладко и пользователь не сталкивается с трудностями.

Тестирование установки проверяет, можно ли без ошибок установить и удалить мобильное приложение, корректно ли устанавливаются обновления.

Мы можем установить приложение, как опубликованное в магазине (App Store, Google Play), так и не опубликованное. Во втором случае сделать это можно через:

- закрытые тесты в Google Play либо Testflight на iOS;
- сервисы дистрибуции;
- компьютер ([со стр 19 в лекции](#))

Чек-лист проверок:

- Проверка установки.
- Проверка прерывания установки.
- Проверка переустановки приложения — важно учитывать, должно ли приложение сохранять данные в памяти телефона после удаления.
- Проверка удаления.
- Проверка обновления.

Обновление может проводиться несколькими способами в зависимости от того, как мы тестируем приложение. Например, если тестирование проводится через TestFlight или Google Play, мы просто нажимаем «Установить» в этих приложениях. Если через сервисы мобильной дистрибуции (например, Firebase), скачиваем и устанавливаем приложение «поверх» старой версии, не удаляя ее.

Важно учитывать, что после обновления данные пользователя должны сохраняться. Например, после обновления приложения интернет-магазина, пользователь должен остаться залогиненным в нем. После обновления мобильной игры весь прогресс должен сохраниться.

Составим тест-кейс на проверку установки:

<p>Название: установка приложения из магазина приложений</p> <p>Приоритет: высокий</p> <p>Предусловия: версия выпущена в магазине приложений</p> <p>Тестовые данные: логин — test, пароль — Testtest</p> <p>Шаги:</p> <ol style="list-style-type: none"> 1. Зайти в магазин приложений 2. Ввести в поиск название приложений 3. Нажать «Установить» 4. Открыть список приложений на телефоне 5. Проверить иконку 6. Запустить приложение <p>Ожидаемый результат: приложение запускается. При запуске пользователь видит окно авторизации</p>
--

Тестирование установки лучше проводить на большом количестве устройств. Часто берут все поддерживаемые версии операционных систем и проверяют установку на каждой из них. Например, если приложение должно поддерживать версии Android 9–14, нужно взять устройства в каждой версии.

Напомню, что выбор устройств для тестирования мы разбирали на уроке «Этапы подготовки к тестированию».

Тестирование безопасности

Тестирование безопасности — обширная тема, которая требует специальных знаний и навыков. Есть даже отдельные специалисты, которые занимаются безопасностью приложений:

- Security Engineer — инженер по безопасности,
- Penetration Tester — специалист по тестированию на проникновение,
- Security Expert — эксперт по безопасности,
- Специалист по информационной безопасности ПО.

Такие специалисты знают:

- типы кибератак,
- особенности server-side- и client-side-уязвимостей — уязвимостей со стороны клиента и со стороны сервера,
- методику анализа защищенности и многое другое.

Безопасность приложений — важный аспект. Пользователи хотят быть уверенными, что их личные данные и платежная информация хранится надежно, а системы защищены от внешних атак. Большая часть пользователей прекратит или не начнет пользоваться приложением, если будут подозрения в плохом обеспечении безопасности.

Основная цель тестирования безопасности — обнаружение и оценка потенциальных уязвимостей в программном обеспечении. Рассмотрим несколько примеров таких проверок.

Зона ответственности специалиста по тестированию безопасности:

- Проверить, не позволяет ли приложение злоумышленнику получить доступ к конфиденциальному содержимому или функциям без надлежащей аутентификации.
- Проверить, что в приложении надежная система защиты паролем и злоумышленник не сможет получить, изменить или восстановить пароль другого пользователя.
- Проанализировать требования к хранению и проверке данных.
- Проверить, защищена ли реализация бизнес-логики, не уязвима ли она для атак извне.
- Проверить, что передаваемые данные шифруются.
- Проверить, что локально хранимые данные шифруются.
- Проверить, что временные данные и файлы удаляются.

Для проведения тестирования безопасности можно использовать различные приложения. Подробнее о них — в дополнительных материалах к лекции.

Зона ответственности специалиста по ручному тестированию — это более общие проверки:

- Проверить, что прямой доступ к страницам, требующим авторизации, невозможен.
- Проверить, что после авторизации пользователю в приложении показываются только те разделы и данные, которые разрешены его уровню доступа.
- Проверить, что по истечении определенного времени происходит логат пользователя, если такое требование есть.
- Проверить, устойчиво ли приложение к SQL-инъекциям.

SQL-инъекции — интересный кейс. Давайте подробнее разберемся, что это такое.

SQL-инъекция

SQL-инъекция — это тип уязвимости системы безопасности, которая может возникнуть в приложениях, использующих базы данных. Происходит, когда злоумышленник вставляет вредоносные запросы SQL в поле ввода для выполнения, тем самым манипулируя базой данных и потенциально получая

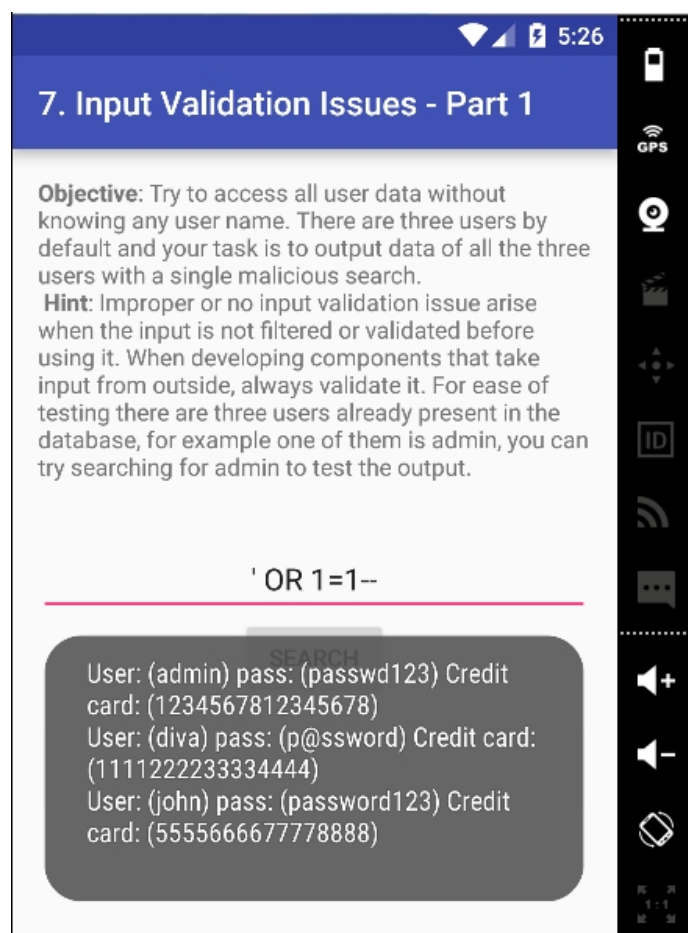
несанкционированный доступ к конфиденциальным данным. Относится ко всем видам приложений, не только мобильным.

Такая атака возможна, потому что приложения часто принимают пользовательский ввод (например, имя пользователя или поисковый запрос), который затем используется для создания SQL-запросов к базе данных.

Если злоумышленник внедрит в этот процесс свой код SQL, он сможет изменить или извлечь данные из базы. Например, он может ввести SQL-команду, чтобы удалить или изменить содержимое базы данных, украсть конфиденциальную информацию (пароли или данные карт), получить доступ к административным функциям веб-приложения.

Сейчас практически все приложения защищены от SQL-инъекций. Для этого разработчики внедряют параметризованные запросы, проверку ввода и очистку ввода пользователя. Однако нужно знать, что это, и уметь проверять.

Рассмотрим пример:



Источник: [teckk2](#)

Есть простой способ — сделать запрос всегда истинным и после этого вставить свой код. Изменение запроса к базе данных на всегда истинное может быть выполнено с помощью простого кода, такого как ' или $1 = 1$;-.

На скриншоте выше вводится запрос в поле ввода. Так как у приложения нет должной защиты, в ответ системным сообщением выдается информация из базы данных. Напомню: большинство систем сейчас надежно защищены, поэтому воспроизвести такую проблему практически невозможно.

Здесь важно условие $1=1$. Оно всегда будет истинным, и база данных воспримет его как команду вернуть в ответе все, что находится в таблице. Таким образом, этот SELECT-запрос запрашивает все данные на всех пользователей.

При вводе ' OR $1=1$ – в строку поиска мы можем увидеть данные, которые хранятся в базе данных.

Также это можно использовать для построения более сложных запросов.

Подробнее про этот тип атак можно почитать в статье [Что такое SQL-инъекция: изучаем на примерах.](#)

Составим тест-кейс на безопасность.

Проверим, что пользователь с правами простого пользователя может выполнять только те действия, которые ему положены. Представим, что у нас есть приложение с функцией чата с различными настройками.

Название: пользователь не может удалять чаты

Требование: только пользователь с ролью «Администратор» может удалять чат

Приоритет: высокий

Предусловия: выполнен логин под ролью «Обычный пользователь»

Тестовые данные: логин — test, пароль — Testtest

Шаги:

1. Открыть приложение
2. Перейти во вкладку «Чат»
3. Открыть любой чат
4. Перейти в настройки чата

Ожидаемый результат: в настройках чата нет опции «Удалить чат»

Тестирование производительности

Тестирование производительности мобильных приложений покрывает больше, чем тестирование производительности веб-приложений.

Эффективность производительности должна быть проверена на самом устройстве, а также при его взаимодействии с серверной частью системы и другими мобильными устройствами.

Производительность мобильных измеряется в трех категориях:

- производительность устройства;
- производительность сервера/API;
- производительность сети.

Рассмотрим каждую подробнее.

Производительность устройства

Аспекты потребления времени и ресурсов — важные факторы успеха приложения. Для их измерения проводится тестирование производительности.

Разберем, что нужно проверить при проверке производительности устройства.

Запуск приложения — сколько времени требуется приложению для запуска. Это первый параметр производительности, с которым сталкивается пользователь.

Как правило, после того как пользователь нажимает на значок приложения, первый экран должен отображаться через 1–2 секунды. Хорошо, когда такие показатели прописаны в требованиях, чтобы было на что опираться при проведении замеров.

Рендеринг экрана — отрисовка всего, что должно на нем быть: изображений, текста, видео и анимации.

Среднее время рендеринга — это время, нужное приложению для загрузки содержимого. Хорошее время отклика приложения при долгом рендеринге — плохой показатель для приложения. В контексте мобильных приложений долгий рендеринг означает время, нужное мобильному приложению для обработки и отображения сложных или ресурсоемких задач, таких как рендеринг изображений с высоким разрешением, создание подробных визуальных эффектов или выполнение операций, требующих больших вычислительных ресурсов.

Время рендеринга зависит от сложности приложения. Как правило, стандартное время — от 1 секунды для легких приложений до 3 секунд для сложных.

Причины медленного рендеринга экрана: неточные размеры экрана, несовместимые шрифты, немасштабированные изображения, блокирующие сценарии и так далее.

Время работы от батареи — при постоянном использовании некоторые мобильные приложения расходуют много энергии батареи и нагревают телефон.

Этот фактор значительно ухудшает производительность приложения. Может происходить, когда приложение использует больше ресурсов, чем требуется. Чрезмерное использование ресурсов создает нагрузку на процессор, и телефон нагревается.

Потребление памяти — при реализации определенных функций в приложении увеличивается потребление памяти. Например, в Android-приложениях — при реализации push-уведомлений.

Варианты аппаратного/программного обеспечения — обязательно нужно проверять приложение на разных устройствах. Иногда приложение без сбоев работает на одном устройстве, но сильно тормозит на другом. Например, Android-приложение можно проверить на телефонах Samsung, Xiaomi и Google Pixel.

Также приложение важно протестировать с различными характеристиками ОЗУ (оперативной памяти) и процессора, например 6 ГБ или 8 ГБ. Если приложение плохо работает на слабых устройствах, обычно программисты делают оптимизацию или убирают поддержку совсем слабых устройств.

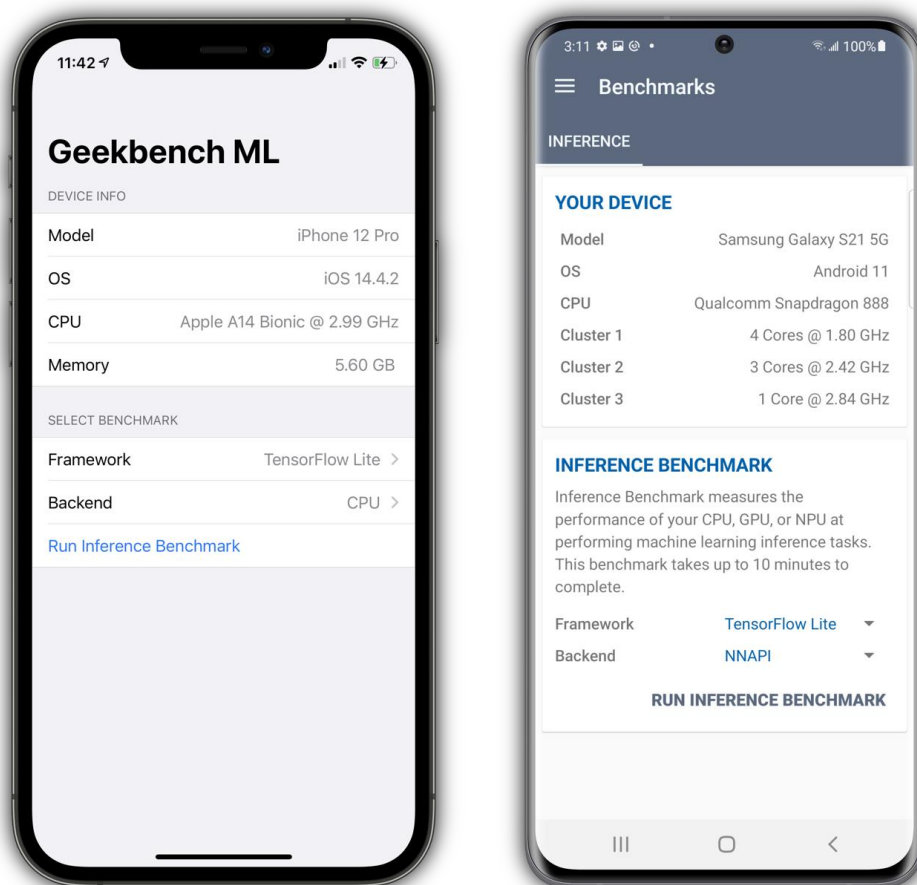
Использование с другими приложениями — когда тестируемое приложение работает параллельно с другими, проблем (например, лагов) быть не должно. Лучший способ проверить это — переключиться с тестируемого приложения на другие. В этом случае нужно загрузить на тестовое устройство и запустить как можно больше приложений.

Приложение в фоновом режиме должно оставаться в том же состоянии, что и до перехода в фоновый режим. Если этот сценарий не обрабатывается должным образом, данные теряются. Например, нам снова придется вводить данные с нуля после выхода приложения из фонового режима.

Представим, что в приложении есть форма оформления доставки с большим количеством данных. Если после возвращения приложения из фонового режима придется вводить все данные заново, пользователи могут раздражаться.

Нам нужно выполнять разные действия и запускать большое количество приложений, чтобы нагрузить мобильное устройство, а затем смотреть его показатели. Для отслеживания этих показателей можем использовать разные приложения, например Geekbench.

Geekbench – кроссплатформенный бенчмарк, который измеряет нагрузку на процессор в реальном времени, и рассчитывает средний показатель скорости работы системы. Программа предоставляет информацию о модели телефона, процессоре, чипсете и операционной системе.



Источник: [Geekbench](#)

Помимо хронометрических измерений, важно учитывать субъективную оценку производительности пользователем. Впечатления от использования приложения могут оказать огромное влияние на то, как долго пользователь будет ждать завершения функции.

Производительность сервера/API

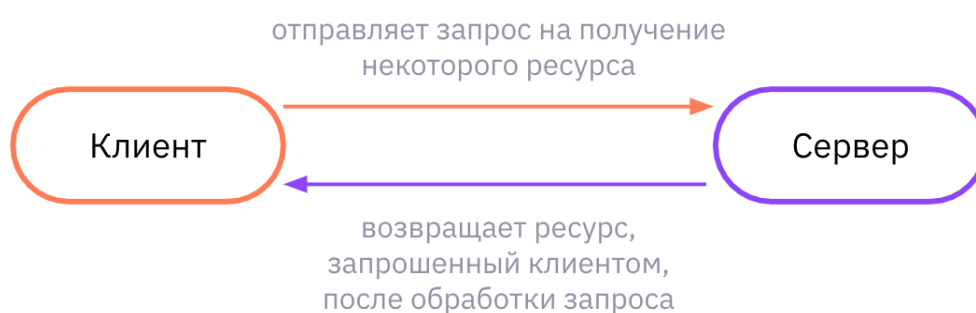
Когда приложение взаимодействует с сервером через API, время отклика становится критическим для производительности.

Разберем, что нужно проверить при проверке производительности сервера/API.

Получение данных с сервера — приложение должно эффективно обрабатывать данные, отправляемые с сервера. Загрузка данных не должна занимать слишком много времени. Точные метрики по времени ответа мы, как правило, получаем от отдела разработки.

В некоторых приложениях данные отправляются в указанном формате. Прежде чем отобразиться в приложении, они преобразуются в соответствующий формат. В этом случае приложение может работать медленнее, а время отклика — увеличиваться.

Напомню, как происходит взаимодействие клиента с сервером:



Пример того, в каком виде мы можем получать данные с сервера в ответ на наш запрос:

Responses application/json	
● 201	Schema Example ✕ collapse all
● 400	
● 401	
	<pre> { "id": 986724, "title": "March Newsletter", "subject": "New Products for Spring!", "sender_id": 124451, "list_ids": [110, 124], "segment_ids": [110], "categories": ["spring line"], "suppression_group_id": 42, "custom_unsubscribe_url": "", "ip_pool": "marketing", "html_content": "<html><head><title></title></head><body><p>Check out c "plain_content": "Check out our spring line!", "status": "Draft" } </pre>

На скриншоте пример удачного ответа (это видим по коду 201) на API-запрос на получение списка маркетинговых писем. Мы видим тело ответа: данные, которые пришли нам с сервера. Например, мы получили данные по рассылке с id 986724 с названием March Newsletter (мартовская рассылка) и так далее.

Вызовы API — когда приложение делает одновременно слишком много запросов, оно может работать медленнее.

Нам нужно проанализировать, все ли запросы, отправляемые на сервер, нужно выполнять, а также проверить, что ответы приходят в корректном формате.

Для тестирования API мобильных приложений мы обычно используем **Charles Proxy**. Работу с ним вы уже изучали на вводном курсе.

Производительность сети

При проверке производительности сети важно проверить **потери пакетов**.

Пакет — небольшая единица данных, которая передается по сети. Это стандартизированный формат для передачи данных по сетям, который обычно

включает информацию о сетевых адресах отправителя и получателя, управляющую информацию и фактические передаваемые данные.

То есть все данные, которые мы отправляем и получаем с помощью сети, разбиваются на маленькие кусочки данных — пакеты. Например, когда мы отправляем письмо, оно идет до получателя не одним файлом, а разбивается на несколько маленьких пакетов.

Мелкие пакеты, на которые разбивается большой объем данных, более управляемые и эффективные: можно одновременно передавать несколько пакетов, что помогает повысить общую производительность и пропускную способность сети. Когда пакет передается по сети, он может пройти через несколько маршрутизаторов и других сетевых устройств, прежде чем достигнет пункта назначения. Каждое устройство в пути проверяет заголовок пакета, чтобы определить, куда его пересылать дальше, а также может изменить заголовок, чтобы включить дополнительную информацию о путешествии пакета. Как только пакеты достигают пункта назначения, принимающее устройство повторно собирает их в исходный поток данных, используя информацию в заголовках для упорядочивания и учета всех данных.

Потеря пакетов — это явление, возникающее в компьютерных сетях, когда один или несколько пакетов данных не достигают своего назначения. Происходит по разным причинам: из-за перегрузки сети, аппаратных сбоев и программных ошибок. Может существенно повлиять на производительность сети и привести к замедлению времени отклика, снижению пропускной способности и даже к перебоям в обслуживании.

💡 При полной потере пакета приложение должно иметь возможность повторно отправить запрос на информацию или создать соответствующее предупреждение.

Если данные неполные, пользователь не сможет понять информацию в приложении. Это может его раздражать, поэтому лучше отобразить подходящее сообщение или предложить повторить попытку.

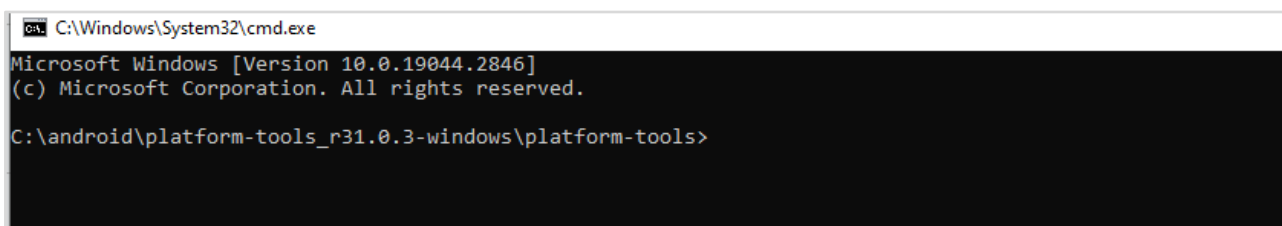
В некоторых случаях потеря пакетов может быть допустимой, например, при потоковой передаче видео- или аудиоконтента, когда определенная степень буферизации может компенсировать потерю некоторых пакетов. Однако в других ситуациях, таких как игры в режиме реального времени или финансовые транзакции, даже небольшая потеря пакетов будет вредной, вызовет задержки или другие проблемы.

Для проверки этого мы можем использовать специальные инструменты мониторинга сети, которые могут отслеживать сетевой трафик и выявлять потерю пакетов. Они помогут определить источник потери пакетов и дать представление о производительности сети.

Для нас основные тестируемые параметры — это задержка и пропускная способность. Мы можем моделировать 3G-, 4G- и 5G-сети для проверки медленного соединения и того, как приложение будет вести себя в случае слишком долгого ответа. А, имитируя потерю пакетов, мы можем определить, как приложение ведет себя в плохих сетевых условиях, и предпринять шаги для оптимизации его производительности.

Например, на Android для проведения тестирования мы можем использовать **Network Connection Emulator**. Для этого нужно:

1. Скачать Android SDK.
2. Включить режим разработчика на телефоне.
3. Включить отладку по USB: Настройки → Параметры разработчика → Отладка по USB.
4. Подключить устройство к компьютеру.
5. Открыть командную строку на компьютере и перейти в каталог инструментов платформы Android SDK (все эти шаги мы уже умеем выполнять).



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.2846]
(c) Microsoft Corporation. All rights reserved.

C:\android\platform-tools_r31.0.3-windows\platform-tools>
```

6. Выполнить команду **adb shell svc network delay [time] [packet loss rate]**, заменив [time] [packet loss rate] на время и коэффициент потери пакетов.

Например, чтобы ввести задержку в 1000 миллисекунд и имитировать скорость потери пакетов 10%, нам нужно использовать команду:

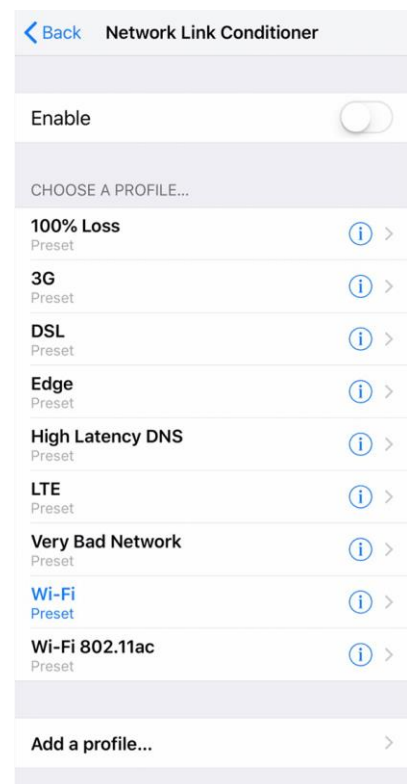
```
adb shell svc network delay 1000 10%
```

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.2846]
(c) Microsoft Corporation. All rights reserved.

C:\android\platform-tools_r31.0.3-windows\platform-tools>adb shell svc network delay 1000 10%
```

Для iOS мы можем использовать **Network Link Conditioner**.

1. Запускаем предустановленный Xcode.
2. Чтобы использовать Network Link Conditioner на iOS, настроим свое устройство для разработки:
 - подключаем iOS-устройство к Mac,
 - в Xcode переходим в Window → Devices & Simulators,
 - выбираем устройство,
 - выбираем «Использовать для разработки».
3. В устройстве переходим: Настройки → Разработчик.
4. Включаем Network Link Conditioner.
5. Задаем нужные параметры сети.



Составим тест-кейс на проверку отображения сообщений в чате:

Название: корректная работа загрузки сообщений при 3G-сети

Требование: все сообщения должны корректно загружаться и отображаться при медленном интернет-соединении

Приоритет: средний

Предусловия: выполнен логин под ролью «Обычный пользователь». У пользователя есть чат с историей переписки. Подключен инструмент для имитации 3G-сети

Тестовые данные: логин — test, пароль — Testtest

Шаги:

1. Открыть приложение
2. Перейти во вкладку «Чат»
3. Открыть любой чат
4. Подождать загрузки сообщений

Ожидаемый результат: все сообщения корректно загружаются и отображаются. Во время загрузки чата отображается спиннер

В этом кейсе мы проверяем, что даже при плохом соединении все должно отображаться и подгружаться. Но требования к приложениям бывают разными, в некоторых случаях при плохом соединении должно показываться соответствующее сообщение.

Нагрузочное тестирование

Нагрузочное тестирование — одна из разновидностей тестирования производительности.

Представим, что у нас есть приложение интернет-магазина. Оно нормально работает в обычные дни, но наступает черная пятница и отдел маркетинга запускает большую акцию. В интернет-магазин одновременно заходит большое количество пользователей — они ищут товары и делают заказы. В какой-то момент сервер перестает справляться с нагрузкой, все начинает тормозить, пользователи не могут оформить заказы. Для компании это потеря прибыли.

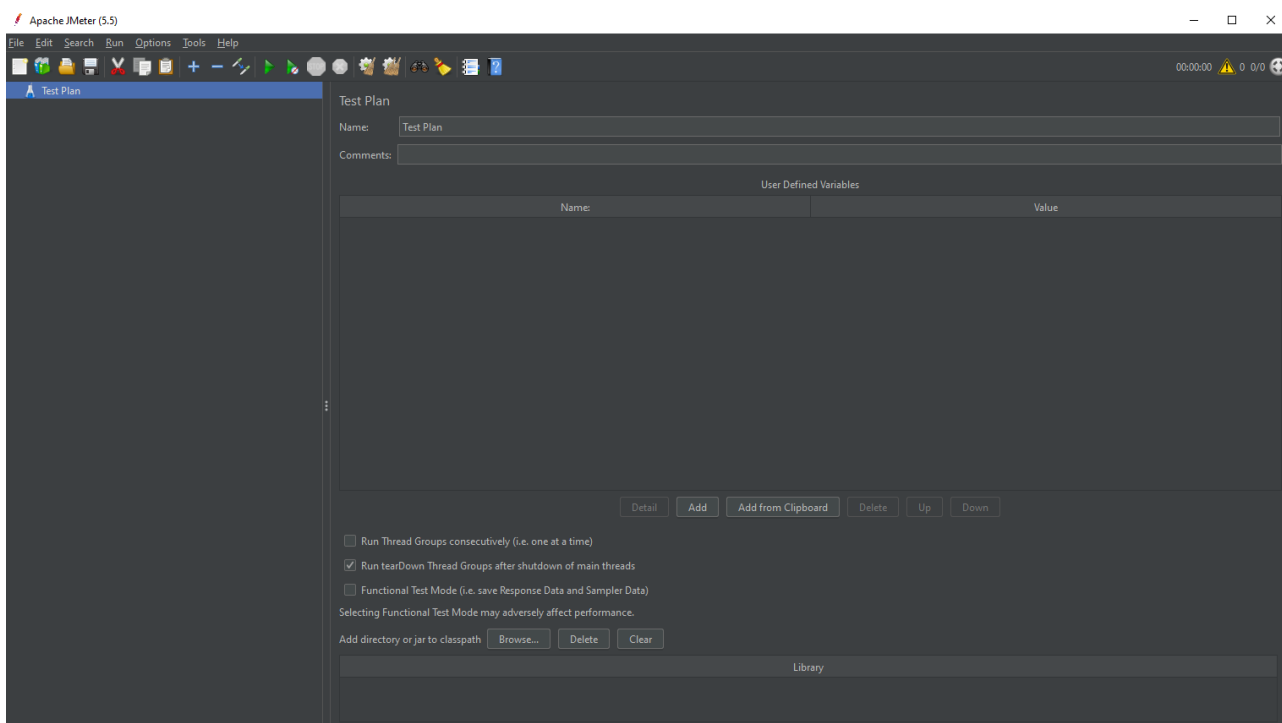
Самый частый сценарий нагрузочного тестирования — проверить, что бэкенд приложения выдержит нагрузку большим количеством пользователей. Сегодня мы рассмотрим инструмент для нагрузочного тестирования — Jmeter.

JMeter — это инструмент с открытым исходным кодом, который можно использовать для тестирования, анализа и измерения производительности сервисов. Это один из лучших инструментов тестирования производительности, который в основном используется в качестве инструментов нагрузочного тестирования.

Для установки Jmeter нужно установить Java и скачать приложение с официального сайта — [Download Apache JMeter](#). Затем — сохранить архив и распаковать в удобном месте, зайти в папку bin и запустить bat-файл **jmeter.bat**.

jaas.conf	01-Feb-80 12:00 AM	CONF File	2 KB
jmeter	01-Feb-80 12:00 AM	File	9 KB
jmeter.bat	01-Feb-80 12:00 AM	Windows Batch File	9 KB
jmeter.jar	01-Feb-80 12:00 AM	Java Archive File	576 KB

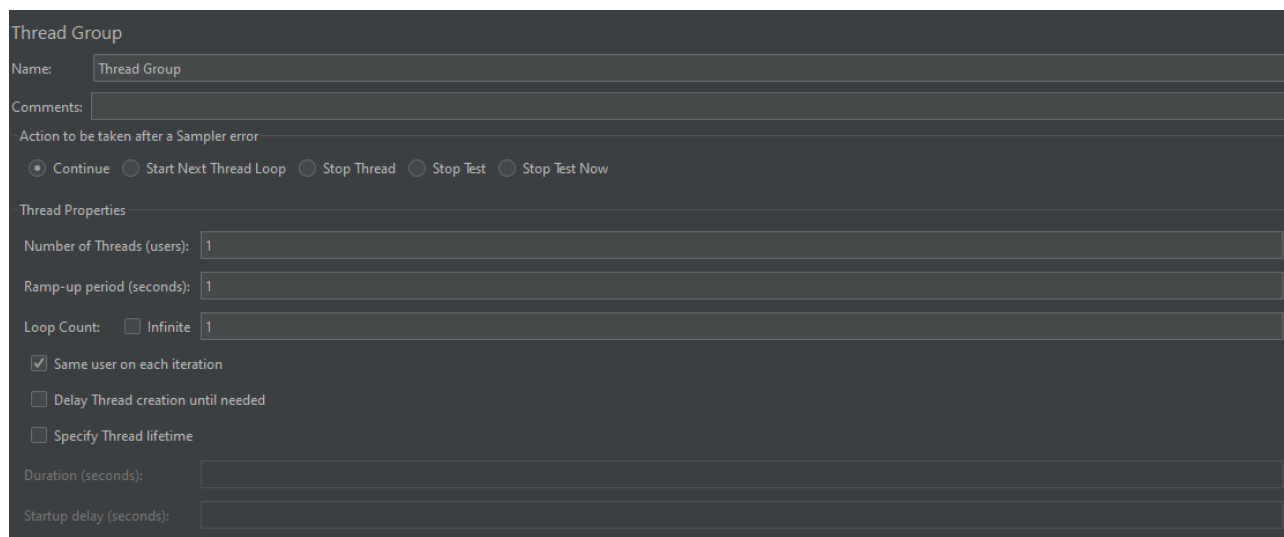
Запустится Jmeter:



На скриншоте мы видим стартовый экран программы. В левой части есть созданный по умолчанию пустой тест-план. Здесь хранится настоящий план тестирования, то есть расписанный сценарий тестирования.

План тестирования можно представить как сценарий JMeter для запуска тестов. Он состоит из элементов тестирования: групп потоков, логических контроллеров, контроллеров, элементов конфигурации.

Первое, с чем нужно познакомиться, — группы потоков.



The screenshot shows the 'Thread Group' configuration window in Apache JMeter. It includes fields for 'Name' (set to 'Thread Group'), 'Comments', and 'Action to be taken after a Sampler error' (with radio buttons for Continue, Start Next Thread Loop, Stop Thread, Stop Test, and Stop Test Now). The 'Thread Properties' section contains fields for 'Number of Threads (users)' (1), 'Ramp-up period (seconds)' (1), and 'Loop Count' (1, with an 'Infinite' checkbox). There are also checkboxes for 'Same user on each iteration' (checked), 'Delay Thread creation until needed', and 'Specify Thread lifetime'. At the bottom, there are fields for 'Duration (seconds)' and 'Startup delay (seconds)'.

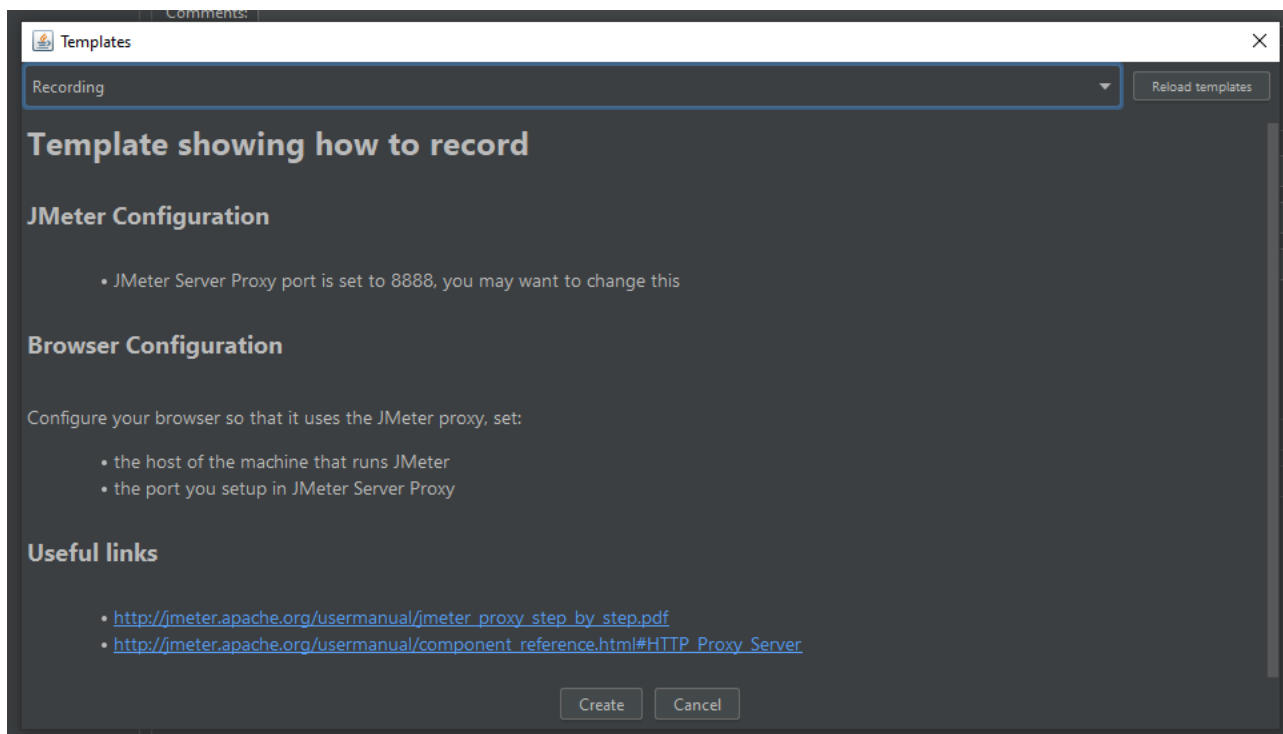
Группы потоков — это набор потоков. Каждый поток представляет одного пользователя, использующего тестируемое приложение. По сути, каждый поток имитирует один реальный пользовательский запрос к серверу.

Элементы управления позволяют установить количество потоков для каждой группы. Например, если вы установите количество потоков как 100, JMeter создаст и смоделирует 100 пользовательских запросов к тестируемому серверу.

Чуть позже мы рассмотрим сами запросы. А сейчас сделаем настройку подключения к мобильному устройству. Будем использовать подключение к эмулятору в Android Studio, но принцип будет одинаковым для всех мобильных устройствах.

Нам нужно создать шаблон: File → Templates. Будем использовать шаблон, в котором настроена запись наших действий для последующего воспроизведения. То есть мы будем выполнять определенные действия и записывать запросы, которые идут от мобильного устройства. После этого сможем перезапустить эти запросы, но уже моделируя необходимую нагрузку.

Выбираем шаблон Recording и нажимаем Create.



Стрессовое тестирование

Стрессовое тестирование направлено на определение эффективности производительности приложения в условиях повышенной нагрузки.

Разберем тестовые условия, которые следует учитывать при стрессовом тестировании.

Высокая загрузка центрального процессора. Для проверки нагрузки мы можем использовать разные приложения, например Geekbench. Для моделирования стрессовых условий мы должны активно использовать мобильное устройство: загружать много файлов, запускать много приложений, имитировать плохую пропускную способность.

Нехватка памяти. Мы должны запустить большое приложение одновременно с нашим тестовым приложением и смотреть показатели.

Нехватка места на диске. Мы можем вручную заполнить телефон большим количеством тяжелых файлов.

Загрузка батареи. Нам также нужно запускать большое количество приложений и качать большие файлы. Чтобы проверить загрузку батареи, можем использовать AnTuTu Benchmark или подобные приложения.

Низкая пропускная способность сети. Для этого в настройках сети указываем минимальные значения пропускной способности.

Большое количество взаимодействий пользователя с приложением. Помимо ручного взаимодействия, могут помочь monkey-тесты.

При проведении стресс-тестов нам нужно симитировать все эти условия: мы забиваем память телефона большим объемом данных, запускаем большое количество приложений, настраиваем низкую пропускную способность сети.

Тестирование локализации и глобализации

Тестирование глобализации приложения включает в себя тестирование приложения для различных форматов дат, чисел и валюты.

Тестирование локализации включает тестирование приложения с локализованными строками и изображениями для конкретного региона. Например, русские, немецкие и французские слова могут быть длиннее, чем слова на других языках:

Английский:	BUY NOW
Русский:	КУПИТЬ СЕЙЧАС
Немецкий:	JETZT KAUFEN
Французский:	ACHETER MAINTENANT

Частая причина отображения иероглифов — некоторые символы теряются (не добавлены изначально) и вместо них отображаются квадраты.

Поскольку у мобильных устройств разные размеры и разрешения экрана, могут случаться проблемы с переведенными строками. Такие моменты должны быть включены в чек-листы локализации/глобализации.

Например, на скриншоте надпись перекрывает экран с объяснением навигации в игре:



Источник: [medium](https://medium.com/@davidmiller/doom-how-to-play-screen-1e1e1e1e1e1e)

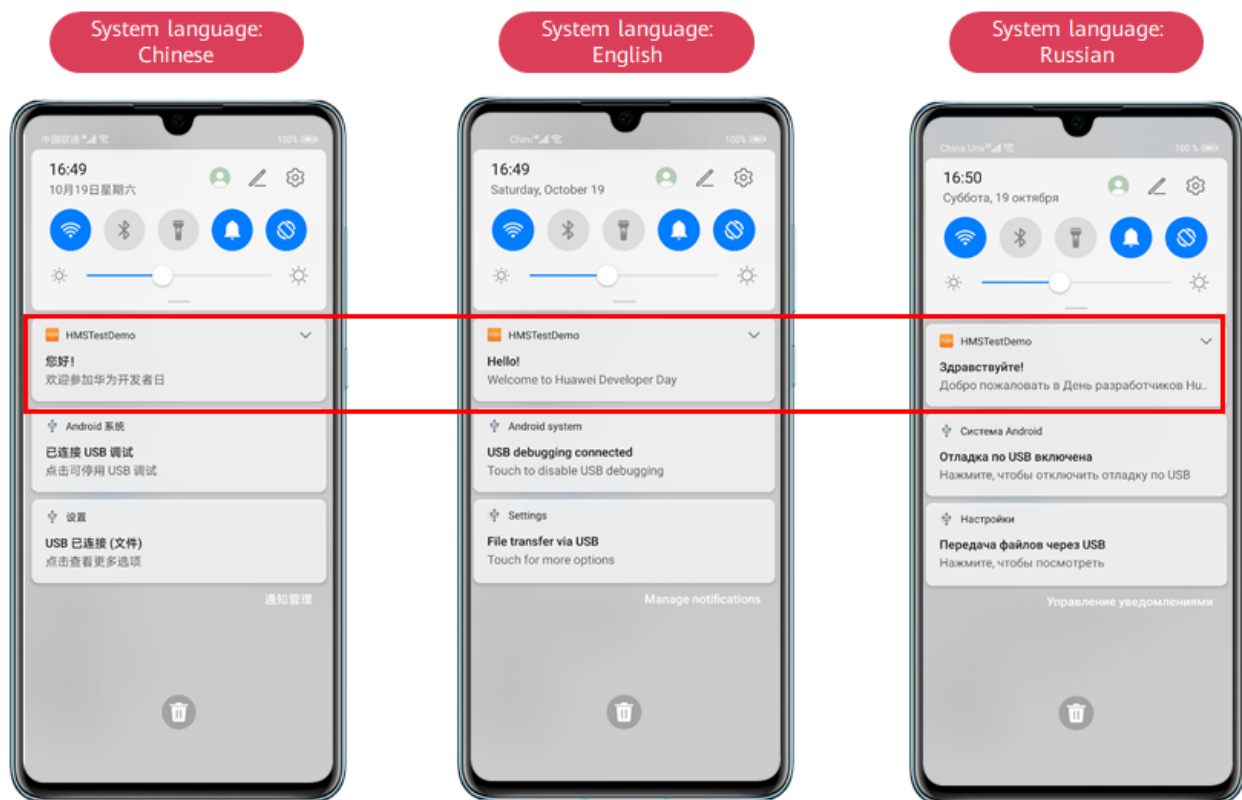
Важный аспект, который нужно проверить, — формат даты: год-месяц-день или день/месяц/год.

	A	B	C	D	E	
1	No	Country	Date Format in Google Sheets	Example	Meaning	
2	1	Australia	d/MM/yyyy	3/04/2000	3 April 2000	
3	2	Canada	yyyy-MM-dd	2000-04-03	3 April 2000	
4	3	Germany	dd.MM.yyyy	03.04.2000	3 April 2000	
5	4	Philippines	M/d/yyyy	4/3/2000	3 April 2000	
6	5	United Kingdom	dd/MM/yyyy	03/04/2000	3 April 2000	
7	6	United States	MM/dd/yyyy	04/03/2000	3 April 2000	
8						

На скриншоте примеры того, как могут отображаться форматы в разных регионах. Важно учитывать это при выставлении языка на устройстве.

Когда проверяем тексты, важно не забывать про уведомления. Они не появляются непосредственно в приложении, поэтому про них часто забывают.

На скриншоте видим пример реализации текста в уведомлениях, когда он полностью не помещается в поле:



Источник: developer.huawei.com

Как правило, мы не пишем много тест-кейсов для проверки локализации, а, например, делаем один-два кейса на каждый язык. И в шагах проходим все основные экраны приложения, где есть текст и изображения.

Подведение итогов

Сегодня мы разобрали разницу между функциональными и нефункциональными требованиями, познакомились с видами нефункциональных тестов и узнали, что нужно тестировать в каждом из них. Посмотрели Jmeter. Разобрались, как смоделировать плохое интернет-соединение и плохую пропускную способность.

Домашнее задание

Возьмите любое мобильное приложение почты и составьте проверки для ручного тестирования безопасности и установочного тестирования.

💡 Задание является необязательным, но крайне полезным для получения практического опыта и подготовки к семинару. Сдавать задание не нужно.

Используемая литература

1. [Лаги, джиттер и потеря пакетов: откуда берутся проблемы с неткодом и как их решать](#)
2. [Тест на потерю пакетов – лучшие инструменты для тестирования и полное бесплатное руководство](#)

Дополнительные материалы

1. «Как тестируют в Google», Арбон Джейсон, Каролло Джефф, Уиттакер Джеймс