ЛАБ-4-Защита

ВАЖНО!
Написано @kkettch по лекциям Клименкова и иногда с помощью ChatGPT, использовалась для защиты ЛР. Информация может оказаться старой или недостоверной.

1. Тестирование системы целиком - системное тестирование

У тестирования системы в целом есть несколько этапов. Один из них - системное тестирование.

- Начинается после окончания интеграции
  Тестирование системы целиком включает следующие этапы:
- *Системное тестирование* - выполняется внутри организации-разработчика. Проводит разработчик - он сам управляет тесты, контролирует когда и сколько их управлять, окружение для тестов собирает самостоятельно.
- *Альфа- и Бета- тестирование* - выполняется пользователем под контролем разработчика. Здесь только заказчик работает с системой (альфа - под управление организации разработчика, они говорят какие тесты нужно выполнять, но их выполняет сам пользователь, бета - организация умывает руки и пользователь сам с ним работает)
- *Приемочное тестирование* - выполняется пользователем
  - Результат - платить или не платить

Системное тестирование

Цель - повышение пользовательского доверия, что система работает корректно.
Фокус -

- Проверяется функциональность системы с точки зрения пользователя (не программист)
- Нефункциональные характеристики - все что касается производительности и тд
- Проводится верификация (правильно ли работает система) и потом валидация (пригодна ли программа для пользователя)
  - В основном методом черного ящика

Тестирование - как научный эксперимент - необходимо повторение
Есть средства автоматизации, скрипты.

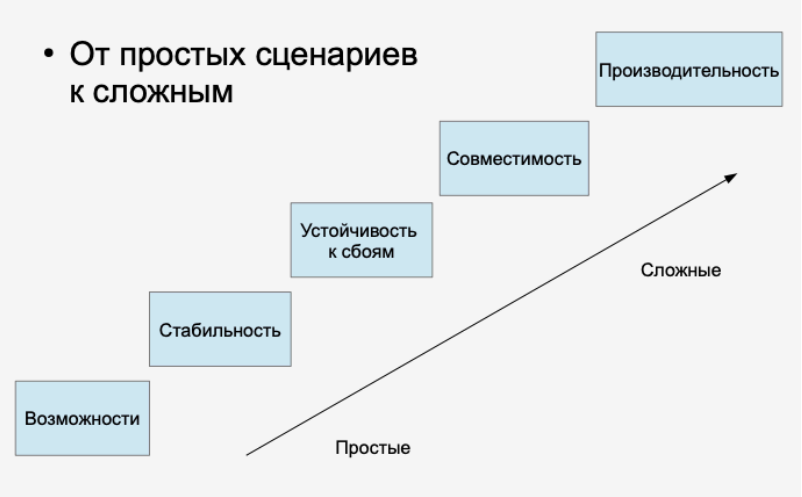
Тестовое окружение стараются максимально приблизить к пользовательскому, так как у разработчиком обычно мощность компьютеров завышена, по сравнению с обычными пользователями

- Необходимо то же самое оборудование
- ОС
- Сетевое окружение

Тестирование эмулирует реальные действия пользователей

2. Тестирование возможностей, стабильности, отказоустойчивости, совместимости

Начинать тестирование необходимо от простых сценариев, к более сложным:



1. **Возможности** - работают ли основные функции системы

- один пользователь выполняет одну транзакцию в единицу времени, вводит все корректно, без ошибок (идеальный пользователь)
- Проверяется, что нормальные переходы между состояниями
- "Среднее" аппаратное обеспечение для работы
- Систему не будут ломать

2. **Стабильность** - пробуется более реалистичная ситуация

- Несколько пользователей одновременно
- Запросы все еще с валидными данными
- Возможно несколько транзакций и появляется нагрузка, система запускается на длительный период времени (выявление утечек памяти, переполнения диска и др.)

3. **Устойчивость к сбоям** - проверка на выносливость

- Пользователя вводят данные некорректно
- Возможно проведение Ddos-атак и проверка, что система сможет самостоятельно вернуться в рабочее состояние
- Намеренные ошибки подсистем безопасности

4. **Совместимость** - проверка функционирования в разных средах

- Проверка функционирования с различными версиями:
- Библиотек
- Браузеров
- Операционных Систем

5. **Производительность** - CARAT

- Capacity - нефункциональные возможности
- Accuracy - точность
- Response Time - время ответа
- Availability - готовность
- Thoroughput - пропускная способность

3. Тестирование производительности - CARAT

CARAT включает в себя все виды тестов, которые необходимы для тестирования производительности во время системного тестирования.

Системное тестирование включает в себя этапы:

- Воможности ПО. Обычно те же сценарии из требований как в функицональном тестировании. Случаи приводящие к ошибкам не используются
- Стабильность. Например, во время одновременного обращения нескольких пользователей, или несколько запросов от одного пользователя.
- Устойчивость к сбоям. Вводятся заведомо неверные данные для анализа их обработки системой.
- Совместимость. Например, кроссбраузерность или возможность использования вместе с программой стороннего ПО.
- Производительность. Проверяется по принципу CARAT.

C - Capacity - Нефункциональные возможности. Доведение до предела каждого из имеющихся параметров и наблюдением за поведением системы.

A - Accuracy - Точность. Определение точности математических рассчетов с заданной константной погрешностью. ПО должно обеспечить определенную точность в заданный промежуток времени.

R - Response Time - Время ответа. Время ответа системы на запрос пользователя. Не должно быть слишком большим или слишком маленьким. В интерактивных системах должно попадать в диапозон от 1 до 5 секунд при нормальной (определенной в требованиях) нагрузке.

A - Availability - Готовность. Выражется в коэффициенте готовности. Высчитывается как: (MTBF - MTTR) / MTBF. Например, 0.99999 - система может простаивать только 5 минут в год.

T - Throughput - Пропускная способность. Проверяет сколько клиентских запросов система может обработать за единицу времени.

4. Альфа и Бета тестирование. Приемочное тестирование

- ***Альфа-*** и ***Бета-тестирование*** - выполняется пользователем под контролем разработчика
- ***Приёмочное тестирование*** - выполняется пользователем

Альфа- и ***Бета-тестирование*** - выполняется пользователем под контролем разработчика. Кроме того, так как тестирование выполняют не разработчики, то тестирующие могут пойти нестандартными путями, которые разработчики не учли. При этом ***альфа-тестирование*** проводится на окружении разработчиков, а ***бета- -*** в реальном пользовательском окружении(но все равно под контролем разработчика).

Приемочное тестирование - выполняется пользователем в его собственном окружении без контроля разработчиков.

5. Нагрузочное тестирование - виды, цели и решаемые задачи.

Нагрузочное тестирование — это вид нефункционального тестирования, направленный на проверку поведения программного обеспечения при ожидаемой или повышенной нагрузке. Оно помогает определить, как система работает при увеличении числа одновременных пользователей, транзакций или объемов данных.

Цели нагрузочного тестирования:

1. **Определение производительности системы** при типичной и максимальной нагрузке.
2. **Проверка устойчивости и стабильности** работы приложения при длительном использовании.
3. **Выявление узких мест (bottlenecks)**: медленные компоненты, перегруженные ресурсы, медленные запросы к БД и т.д.
4. **Оценка масштабируемости**: как система справляется с ростом нагрузки.
5. **Проверка соответствия SLA** (Service Level Agreement) — например, время отклика не должно превышать 2 секунд при 1000 одновременных пользователях.

Решаемые задачи:

- Выяснение максимального количества пользователей, которое может обслужить система.
- Определение времени отклика и скорости обработки данных.
- Анализ использования ресурсов: CPU, RAM, диск, сеть.
- Проверка корректности работы системы под нагрузкой (отсутствие сбоев, ошибок, утечек памяти и т.д.).

Виды:

1. **Тестирование при нормальной нагрузке** – симуляция среднего числа пользователей и операций.
2. **Тестирование при пиковой нагрузке** – проверка поведения системы при максимальной ожидаемой нагрузке.
3. **Тестирование при нарастающей нагрузке** – нагрузка увеличивается по шагам до предельной.
4. **Тестирование при длительной нагрузке (soak testing)** – проверка стабильности в течение длительного времени.

6. Принципы реализации нагрузочного тестирования ПО.

Ключевые принципы:

1. **Определение целей и метрик:**
   - Что именно мы хотим узнать? (время отклика, предел пользователей, использование ресурсов)
   - Какие показатели считать успешными или критичными?
2. **Реалистичность сценариев:**
   - Моделирование поведения реальных пользователей (например, 80% читают, 10% пишут, 10% — регистрируются).
   - Использование данных, приближенных к боевым.
3. **Постепенное увеличение нагрузки:**
   - Позволяет выявить момент, когда система перестает справляться.
4. **Изоляция среды:**
   - Желательно проводить тесты в среде, идентичной продуктивной, но изолированной от неё.
5. **Мониторинг системы:**
   - Использование инструментов для отслеживания ресурсов (CPU, память, диск, сеть, БД).
   - Фиксация времени отклика, ошибок, логов.
6. **Повторяемость тестов:**
   - Один и тот же тест можно воспроизвести несколько раз с одинаковыми условиями.
7. **Анализ результатов:**
   - После тестов следует анализировать графики, логи и отчеты, выявлять узкие места, возможные сбои, причины деградации.

7. Инструменты для реализации нагрузочного тестирования.

Их очень много - "как грязи": для веба, для обычных приложений, для отдельных компонентов (протестировать производительность диска).

Некоторые инструменты для веб-приложений:

- HP Load Runner
- LoadComplete
- IBM Rational Performance Tester
- Load UI Pro
- Apache JMeter
- The Grinder
- Tsung

8. Apache JMeter - архитектура, поддерживаемые протоколы, особенности конфигурации.

Является бесплатным инструментом, интерфейс пользователя простой, может строить графики (примитивные)

- Имеет возможность создания распределенной нагрузки
- Эмуляция одновременной работы пользователей
- Снятие метрик
- Возможность разрабатывать плагины
- Планы тестирования в XML - просто контролировать версии

Может быть использован для записи запросов.
JMeter это не браузер!

Основным элементом конфигуарции является разработка тестового плана

- Первый раздел - Thread Group - описывает пул пользователей для выполнения теста (количество, возрастание и пр.)
- Семплеры - формируют запросы и генерируют результат
  - Большой набор встроенных протокол - TCP, HTTP, FTP и др.
- Логические контроллеры - определяют порядок вызова семплеров
  - Конструкции управления (if, loop...)
  - Управление потоком
- Слушатели - получают ответы
  - Осуществляют операции с результатами
  - Не обрабатывают данные!
- Таймеры - добавляют задержку между запросами
  - Постоянные или в соответствии с законами
- Assertion - проверяют результаты
- Элементы конфигурации - Сохраняют предустановленные значения для семплеров
- Препроцессоры - изменяют семплеры в их контексте
- Постпроцессоры - применяются ко всем семплерам в одном контексте

Порядок выполнения:

1. Конфигурационные элементы
2. Препроцессоры (Pre-processors)
3. Таймеры (Tmers)
4. Семплеры (Sampler)
5. Постпроцессоры (Post-Processors)
6. Assertions
7. Слушатели (Listeners)

Дополнительные возможности

- Автоматическая генерация CSV-файла
- Bandwidth Throtting - ограничение полосы пропускания (статические, динамические)
- IP Spoofing - замена src_ip, для разделения клиентов
- Поддержка теста TPC-C

Распределенное тестирование

9. Стресс-тестирование - основные понятия, виды стресс-сценариев.

Стресс-тестирование (англ. *Stress Testing*) — это тип тестирования, целью которого является проверка поведения системы в условиях экстремальной (за пределами нормальной) нагрузки. Оно показывает, как приложение реагирует на перегрузку, недостаток ресурсов, сбо́и или пиковые нагрузки.

Основные цели:

- Оценка предельной устойчивости системы.

- Проверка способности к восстановлению (resilience) после отказов.
- Проверка отказоустойчивости при нехватке ресурсов (низкая память, сбои в сети, зависание БД).

Виды стресс-сценариев:

1. **Пиковая нагрузка:** резкое увеличение числа запросов в короткое время.
2. **Утечка ресурсов:** моделирование утечек памяти или блокировки потоков.
3. **Отключение компонентов:** внезапное отключение БД, кэша, сервисов.
4. **Сетевые сбои:** высокий latency, потеря пакетов, нестабильное соединение.
5. **Недостаток ресурсов:** искусственное ограничение CPU, RAM, дискового пространства.

Типовые последствия стресс-сценариев:

- Ошибки при запросах (500 Internal Server Error, Timeout).
- Сбои в сервисах.
- Утечки памяти, рост времени отклика.
- Крах или зависание приложения.

10. Стресс-тестирование ПО. Виды стресс-тестов ПО. Тестирование ёмкости.

1. Стресс-тестирование ПО:

Цель — выяснить, как система ведет себя за пределами своего нормального диапазона работы. Это помогает выявить:

- Слабые места в архитектуре.
- Поведение при сбоях.
- Границы масштабируемости.

---

2. Виды стресс-тестов:

Тип стресс-теста	Описание
Тест резкой нагрузки (Spike test)	Моментальный всплеск количества запросов (например, в 10 раз выше обычного).

Тип стресс-теста	Описание
Тест длительной перегрузки (Soak Stress)	Долгая работа под экстремальной нагрузкой для выявления деградации.
Сценарии деградации	Проверка, как система работает при отключении кэш-сервера, БД и т.п.
Тест на отказоустойчивость	Проверка реакции системы на сбои — например, падение сервиса или перегрузка сервера.
Инфраструктурные сбои	Проверка поведения при нехватке ресурсов: переполнение диска, нехватка RAM и т.д.

---

3. Тестирование ёмкости (Capacity Testing):
   Определение:
   Тестирование ёмкости определяет максимальное количество пользователей, транзакций или объема данных, которое система может обрабатывать до снижения производительности.

Цели:

- Найти **точку насыщения** — при каком количестве пользователей система перестаёт быть эффективной.
- Выявить **границы масштабируемости** и определить, как масштабировать систему горизонтально или вертикально.
- Определить, какие ресурсы становятся ограничивающим фактором при росте нагрузки.