*Преподаватель: Прилепская Татьяна Владимировна*

07.09.2023

**Отладка и тестирование информационных систем**

**Введение. Организация тестирования в команде разработчиков**

**Стадии тестирования в процессе разработке ПО:**

1. Автономное тестирование компонентов программного обеспечения;
2. Комплексное тестирование разрабатываемого программного обеспечения;
3. Системное или оценочное тестирование на соответствие основным критериям качества.

**Основные принципы**, соблюдаемые для повышения качества тестирования:

1. Предполагаемые результаты должны быть известны до тестирования;
2. Следует избегать тестирования программы автором;
3. Досконально изучать результаты каждого теста;
4. Необходимо проверять работу программы на неверных данных;
5. Вероятность наличия необнаруженных ошибок в части программы пропорциональна числу ошибок, уже обнаруженных в этой части.

**2 подхода к формированию тестовых наборов:**

1. Структурный – базируется на том, что известны алгоритмы работы программы. В основе структурного тестирования лежит концепция максимально полного тестирования всех маршрутов программы.
2. Функциональный – основывается на том, что алгоритм работы программного обеспечения не известен.

**Отладка** – процесс поиска, локализация и исправления ошибок в программе

1. Тестирование – процесс выполнения ПО системы или компонента в условиях анализа или записи получаемых результатов с целью проверки (оценки) некоторых свойств тестируемого объекта
2. Процесс анализа пункта требований к ПО с целью фиксации различий между существующим состоянием ПО и требуемым при экспериментальной проверке соответствующего пункта требований
3. Контролируемое выполнение программы на конечном множестве текстовых данных и анализ результатов этого выполнения для поиска ошибок

Цели и область тестирования

**Целью проектирования тестовых вариантов** является систематическое обнаружение различных классов ошибок при минимальных затратах времени и стоимости

**Тестирование обеспечивает:**

1. Обнаружение ошибок;
2. Демонстрацию соответствия функций программы ее по назначению;
3. Демонстрацию реализации требований к характеристикам программы;
4. Отображение надежности как индикатора качества программы.

**Организация тестирования в команде разработчиков**

Коммуникация и взаимодействие в команде тестирования:

* Аккаунт менеджер;
* Менеджер проекта ();
* Системный аналитик (аналитик);
* Системный архитектор (архитектор);
* Программист (разработчик);
* Ведущий программист (технический лидер);
* QA-специалист;
* SDET-специалист (контроль качества, автоматизация тестирования);
* QA lead (ведущий специалист по управлению и контролю качества);
* Тимлид.

Методология тестирования

**Виды тестирования:**

* Блочное тестирование
* Тестирование компонента
* Интеграционное тестирование
* Регрессивное тестирование
* Тестирование системы

Фазы тестирования

**Этапы реализации тестирования:**

1. Создание тестового набора (test suite) путем ручной разработки или автоматической генерации для конкретной среды тестирования (testing environment).
2. ﻿﻿﻿Прогон программы на тестах, управляемый тестовым монитором (test monitor, test driver) с получением протокола тестирования (test log).
3. ﻿﻿﻿Оценка результатов выполнения программы на наборе тестов с целью принятия решения о продолжении или остановке тестирования.

Тестовые среды

**Среда тестирования** – это настройка программного и аппаратного обеспечения для групп тестирования для выполнения тестовых случаев.

*Испытательный стенд или тестовая среда настраиваются в соответствии с требованиями тестируемого приложения. В некоторых случаях испытательный стенд может представлять собой комбинацию тестовой среды и тестовых данных, которые он использует.*

Документированность процесса тестирования

**Тестовый план** – это документ, или набор документов, который содержит тестовые ресурсы, перечень функций и подсистем, подлежащих тестированию, тестовую стратегию, расписание тестовых циклов, фиксацию тестовой конфигурации (состава и конкретных параметров аппаратуры и программного окружения), определение списка тестовых метрик, которые на тестовом цикле необходимо собрать и проанализировать (например, метрик, оценивающих степень покрытия тестами набора требований)

* Тесты разрабатывают на основе спецификаций как вручную, так и с помощью автоматизирующих средств.
* Для оценки качества тестов используют различные метрики, связанные с количеством найденных дефектов, покрытием кода, функциональных требований, множества сценариев.
* Вся информация об обнаруженных в процессе тестирования дефектах (тип, условия обнаружения, причина, условия исправления, время, затраченное на исправление) заносятся в базу дефектов.
* Информация о тестовом плане, тестах и дефектах используется в конце каждого цикла тестирования для генерации тестового отчета и корректирования системы тестов для следующей итерации

**Типы тестирования по виду подсистемы или продукта таковы:**

* Тестирование основной функциональности, когда тестированию подвергается собственно система, являющаяся основным выпускаемым продуктом.
* ﻿﻿Тестирование инсталляции включает тестирование сценариев первичной инсталляции системы, сценариев повторной инсталляции (поверх уже существующей копии), тестирование деинсталляции, тестирование инсталляции в условиях наличия ошибок в инсталлируемом пакете, в окружении или в сценарии и т.п.

﻿﻿Тестирование пользовательской документации включает проверку полноты и понятности описания правил и особенностей использования продукта, наличие описания всех сценариев и функциональности, синтаксис и грамматику языка, работоспособность примеров и т.д.

**Методология тестирования сложных систем**

**Сложная система** – система, состоящая их множества взаимодействующих составляющих, вследствие чего сложная система приобретает новые свойства, которые отсутствуют на подсистемном уровне и не могут быть сведены к свойствам подсистемного уровня.

Эффективно начинать тестирование комплексных систем на ранних стадиях разработки ПО.

Тестирование организует цикл: тестирование – исправление – повторное тестирование.

Тестирование выполняется всегда для всех объектов системы вне зависимости от её критичности.

05.10.2023

**Принципы тестирования:**

1. Полное тестирование невозможно – ограничены ресурсы и время. Фокус на наиболее критических аспектах;
2. Раннее тестирование – тестирование начинается на ранних этапах разработки, что экономит ресурсы и время;
3. Исчерпывающее тестирование – важно покрыть все функциональные и нефункциональные области. Тестовые случаи должны покрывать их все.

**Виды тестирования:**

* Функциональное (работоспособность) и нефункциональное (удобство);
* Статическое (документация) и динамическое (практическая проверка).

**Другие виды тестирования:**

* Нагрузочное;
* Тестирование UX;
* Конфигурационное (совместимость).

**Классификация по знанию внутренней системы:**

* Чёрный ящик (поведенческий) – тестировщик не знает как устроен виртуальный мост (?). Идеи идут от предполагаемых паттернов поведения пользователей;
* Белый ящик – тестировщик знает устройство логику программы;
* Серый ящик – смешанное тестирование.

**Критерии тестирования:**

1. Функциональная корректность – проверка на выполнение всех функций и соответствие ожидаемому поведению;
2. Надёжность – оценка стабильности продукта в различных условиях, выявление и устранение сбоев.
3. Производительность – оценка скорости и эффективности работы продукта, проверка на высокие нагрузки и скорость отклика;
4. Совместимость – проверка работы продукта на разных платформах, браузерах, устройствах. Обеспечение работоспособности в различных окружениях;
5. Безопасность – проверка на уязвимости и угрозы безопасности, защита данных и конфиденциальности;
6. Удобство использования – оценка уровня удовлетворения пользователей, удобство навигации и интерфейса.

**Требование** – описание того, какие функции и с соблюдением каких условий должно выполнять приложение в процессе решения полезной для пользователя задачи.

**Требования:**

* Позволяют понять, что и с соблюдением каких условий система должна делать;
* Предоставляют возможность оценить масштаб изменений и управлять изменениями;
* Является основной для формирования плата проекта (в том числе плана тестирования);
* Помогают предотвращать или разрешать конфликтные ситуации;
* Упрощают расстановку приоритетов в наборе задач;
* Позволяют объективно оценить степень прогресса в разработке проекта.

19.10.2023

**Организация тестирования в команде разработчиков**

**Тестирование ПО** – процесс исследования ПО с целью получения информации о качестве продукта;

**Баг** – отклонение фактического результата от ожидаемого результата;

**Тестировщик** – человек, в профессиональные обязанности которого входит обнаружение, локализация и отслеживание ошибок в программе, описание их самих, а также шагов для воспроизведения. Специалист, способный рассмотреть проблему с точки зрения пользователя.

**Составляющие качества ПП:**

* Наличие ошибок в продукте;
* Соответствие требованиям;
* Пригодность к использованию;
* Составляющие цены:
  + Цена сбоев – затраты на выявление и исправление ошибок и выходов из строя;
  + Цена оценки качества – издержки на контроль качества: издержки на тестирование продукта;
  + Цена превентивных усилий – затраты на обучение, проектирование процесса, планирование качества.

**Цели тестирования:**

* Повышения качества продукта;
* Оценка статуса продукта;
* Сократить затраты на разработку;
* Сократить затраты на поддержку выпущенного продукта и т.д.

Тестирование само по себе не повышает качество продукта.

**Кому нужно тестирования:**

* Тестировщику?
* Разработчику:
  + Качественное ведение дефектов;
  + Своевременное ведение дефектов;
  + Предоставление требуемой информации;
* РМ (программный менеджер):
  + Реальная картина готовности проекта;
  + Точное планирование;
  + Приемлемые проекту сроки;
* Заказчику:
  + Следование бюджету;
  + Следование срокам исполнения проекта;

**Требуемое тестовое покрытие:**

1. Понимание пользователя, use-case’ов, приоритетов;
2. Следование бюджетов.

**Кому помогает тестирование:**

* Помогает разработчикам исправлять баги;
* Руководству принимать решения;
* Бизнесу не пропускать критичные баги.

Результат работы тестировщика – помощь другим участникам проекта.

**Почему тестировщик не может отвечать за качество проекта:**

* Тестировщик не отвечает за работу других участников;
* Не вносит изменения в код;
* Не может организационно повлиять на решения об исправлении ошибок;
* Не управляет ресурсами проекта;
* Не управляет бюджетом проекта.

**Кто отвечает за качество:**

* В проекте – менеджер проекта;
* В подразделении разработчиков – руководитель подразделения;
* На уровне компании – первое лицо компании.

**Тестирование** – контроль качества (поиск ошибок, дефектов);

**Контроль качества** (Quality Control - QA) – измерение качества продукта;

**Обеспечение качества** (Quality Assurance QA) – измерение и управление качеством процесса, который используется для создания качественного продукта.

**Уровни тестирования:**

* Модульное тестирование (Unit testing) – проверяет функционирование отдельно взятого элемента системы. Что считать элементом (модулем) системы зависит от контекста;
* Интеграционное тестирование (Integration testing) – процесс проверки взаимодействия между программными компонентами/модулями;
* Системное тестирование (System testing) – тестирование всей системы целиком.

**Функциональное тестирование:**

* Ручное – основный вид тестирования, проверяет все требования;
* Регрессивное – проверка что изменения исправляющие ошибки не породили новые.

**Нефункциональное тестирование:**

* Тестирование производительности – определение скорости работы системы под нагрузкой;
* Нагрузочное тестирование – тестирование работоспособности при больших нагрузках (потеря работоспособности/неоправданный уровень производительности);

**Тестовые артефакты:**

* План тестирования – документ, описывающий объём работ по тестированию, начиная с описания объекта, стратегии, используемые для его тестирования, методики и необходимые ресурсы, с графиком выполнения запланированных работ, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения;
* Набор тест кейсов и тестов – последовательность действий, по которой можно проверить соответствует ли тестируемая функция установленным требованиям;
* Чек лист – список пунктов, которые нужно обязательно проверить;
* Дефекты/баг репорты – документы, описывающие ситуацию или последовательность действий, приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.

02.11.2023

**Виды и методы тестирования**

Функциональное в основном проверяется метод белого ящика.

**Сравнение функционального и нефункционального тестирования:**

|  |  |
| --- | --- |
| Функциональное тестирования | Нефункциональное тестирование |
| Выполняется с использованием функциональной спецификации, предоставленной клиентом, и проверяет систему на соответствие функциональным требованиям; | Проверяет производительность, надёжность, масштабируемость и другие нефункциональные аспекты программной системы |
| Выполняется в первую очередь | Проводится после функционального |
| Можно использовать ручное и автоматическое тестирование | Автоматическое тестирование более эффективно |
| Бизнес-требования – исходные данные для функционального тестирования | Параметры производительности (скорость, масштабируемость и т.д.) – входные данные для нефункционального тестирования |
| Описывает, что делает продукт | Описывает, как хорошо работает продукт |
| Легко тестировать вручную | Трудно тестировать вручную |
| Примеры:   1. Модульное тестирование; 2. Дымное тестирование; 3. Санитарное тестирование; 4. Интеграционные тестирование; 5. Белый ящик; 6. Чёрный ящик; 7. Приёмочное тестирование пользователей; 8. Регрессивное тестирование. | Примеры:   1. Тестирование производительности; 2. Нагрузочное тестирование; 3. Объёмное тестирование; 4. Стресс тестирование; 5. Тестирование безопасности; 6. Тестирование установки; 7. Проверка на проницаемость; 8. Тестирование совместимости; 9. Миграционное тестирование. |

**Методы модульного тестирования:**

1. Покрытие операторов – метод белого ящика, при котором все исполняемые операторы в исходном коде выполняются хотя бы один раз;
2. Покрытие решений – метод белого ящика, сообщающий об истинных или ложных результатах каждого логического выражения исходного кода;
3. Покрытие ветвей – метод белого ящика, проверяющий каждый результат модуля кода (оператора, цикла и т.д.);
4. Покрытие условий/выражений – проверяет и оценивает переменные или подвыражения в условном операторе;
5. Покрытие конечного автомата – работает на поведении дизайна. Наиболее сложные тип метода.

**Smoke Testing (Дымовое тестирование)** – процесс тестирования ПО, определяющий, является ли развёрнутая сборка ПО стабильной или нет. Дымовое тестирование является подтверждением того, что команда QA может приступить к дальнейшему тестированию ПО. Состоит из минимального набора тестов, запускаемых в каждой сборке для проверки функциональных возможностей ПО. Другие названия – Тестирование проверки сборки, Тестирование доверия.

**Sanity Testing** – тестирование, выполняемое после получения сборки ПО с изменениями, чтобы убедиться, что ошибки были исправлены и новых не возникло. Проверка программы, работает ли она так, как ожидалось. Не проверяет новую функциональность, но проверяет рациональность разработчика при написании.

**Сравнение дымового и санитарного тестирования:**

|  |  |
| --- | --- |
| Дымное тестирование | Санитарное тестирование |
| Выполняется для проверки правильной работоспособности критически важных функций программы; | Проверка того, что новые функции/ошибки исправлены; |
| Проверка стабильности системы; | Проверка рациональности системы; |
| Выполняется разработчиками или тестировщиками; | Обычно выполняется тестировщиками; |
| Документируется или записывается в сценарии; | Не документируется и не записывается; |
| Часть приёмочного тестирования; | Подмножество регрессивного тестирования; |
| Проверяет всё систему; | Проверяет определённый компонент системы; |
| «Общая проверка здоровья» | «Специализированная проверка здоровья» |

**Системное интеграционное тестирование (SIT)** – тип тестирования ПО, проводимого в интегрированной аппаратно-программной среде для проверки поведения всей системы. Проводится на полной интегрированной системе для оценки соответствия системы заданным требованиям. Проверяет взаимодействие модулей ПО, сосуществование программной системы с другими, интерфейс между модулями. Тестируются модуль по-отдельности, а затем как единая система.

**Критерии входа и выхода для интеграционного тестирования:**

Обычно при выполнении интеграционного тестирования используется стратегия ETVX (критерии входа, задачи, проверки критерии выхода).

**Критерии входа:**

* Завершение модульного тестирования;

**Входы:**

* Данные о требованиях к ПО;
* Документ по дизайну ПО;
* План проверки ПО;
* Документы по интеграции ПО.

**Мероприятия**:

* На основе требований высокого и низкого уровня создание тестовых примеров и процедур;
* Объединение низкоуровневых сборок модулей, реализующих общий функционал;
* Разработка тестовых обвязок;
* Тестирование сборки;
* После прохождения теста сборка объединяется с другими сборками и тестируется до тех пор, пока система не будет интегрирована в целом;
* Повторное выполнение всех тестов на платформе с целевым процессором с получением результатов;

**Критерии выхода:**

* Успешное завершение интеграции программного модуля на целевом аппаратном обеспечении;
* Корректная работа ПО в соответствии с указанными требованиями;

**Выходы:**

* Отчёты об интеграционных испытаниях;
* Примеры и процедуры тестирования ПО (SVCP).

**Методы тестирования чёрного ящика:**

* Тестирование класса эквивалентности – используется для минимизации количества возможных тестовых случаев до оптимального уровня при сохранении разумного тестового покрытия;
* Тестирование граничных значений – определяет, является ли определённых диапазон значений приемлемым для системы. Сокращает тестовые случаи, подходит для ситуаций с диапазоном;
* Тестирование таблицы решений – таблицы решений помещает причины и их последствия в матрицу. В каждом столбце есть уникальная комбинация.

05.02.2024

**Автоматизация тестирования**

**Основная идея автоматизированного тестирования** заключается в использовании автотестов - записанных на специальных скриптовых языках действий по проверке качества программ.

**Преимущества автоматизации:**

* Экономия времени;
* Исключение человеческого фактора;
* Отсутствие необходимости в GUI;
* Наличие инструментария фиксации ошибок и результатов;
* Возможность эмулировать многопользовательскую работу.

**Недостатки автоматизации:**

* Временные затраты на создание, поддержку и тестирование тестов;
* Неприменимость к некоторым объектам, оцениваемым субъективно;
* Необходимость программистских навыков у тестировщика;
* Чувствительность к среде, программному и аппаратному окружению тестируемого приложения.

**Типы автоматизирующегося тестирования:**

* Функциональное (в т.ч. модульное или unit-testing);
* Регрессивное (проверка работоспособности старого функционала и отсутствия ранее исправленных дефектов в новых версиях);
* Нагрузочное (поведение приложения под рабочей и стрессовой нагрузкой, влияние работающего приложения на системное окружение).

**Планирование функционального тестирования:**

В качестве основы для планирования автоматического типа тестирования используются явные и неявные функциональные требования к программному продукту.

Функциональные требования разделяют по степени критичности и начинают планирование тестов для самых критичных пользовательских бизнес-прецедентов.

**Цели планирования нагрузочного тестирования:**

* Убедиться, что при той или иной нагрузке в работе приложения не возникает сбоев, т.е. отсутствуют ошибки;
* Проверить, сохраняется ли с ростом нагрузки эргономичность приложения;
* Поиск опасных тенденций для системных ресурсов клиента и сервера.

**Уровни нагрузки:**

* Минимальная (один пользователь) – проверка общей работоспособности;
* Рабочая (штатное количество пользователей) – приложение должно работать без проблем;
* Стрессовая/пиковая – нагрузка, которую приложение в принципе должно выдержать.

**Модульное тестирования**

Для модульное тестирования применяются специальные утилиты, позволяющие сразу запустить все тесты и увидеть результат.

**Создание тестов для NUnit:**

* Для написания тестов используются расширение любого языка .NET;
* Основные элементы таких расширений: *утверждения (assertions)* и *директивы (directives)*;
* Тесты оформляются как методы *тестирующего* класса, который снабжается ссылкой на *тестируемый* класс.

**Утверждения** – гипотезы, высказываемые тестировщиков относительно результатов выполнения некоторого теста. Если гипотеза подтвердилась, то начинает выполняться следующий тест, иначе возникает ошибка.

**Модели для утверждений:**

* Классическая модель;
* Закрытая модель.

**Группы (виды) утверждений:**

* Утверждение равенства (Equality Asserts);
* Утверждение сравнения (Comparison Asserts);
* Утверждение о типах (Type Asserts);
* Утверждение о строках (String Asserts);

**Директивы (атрибуты)** – специальные предложения, используемые для структурирования тестовых заданий и описания дополнительных спецификаций теста. Директивы содержатся в пространстве имён nunit.framwork, которое должно быть включено в любой файл, содержащий тесты.

**Категории директив:**

* Идентифицирующие – показывают на классы и методы для проверки;
* Селектирующие – позволяют пометить тесты для специальных случаев;
* Модифицирующие – используется для тестов, где может возникнуть ошибки;
* Подготовки и очистки – если нужно после тестов очистить некоторые результаты;
* Параметризирующие – используются для передачи параметров в тестирующий метод.

**Юзабилити и интеграционное тестирование**

**Юзабилити** – степень, с которой продукт может быть использован определёнными пользователями при определённом контексте для достижения определённых целей с должной эффективностью, результативностью и удовлетворённостью. Отражает степень удобства использования программного продукта конечными пользователями программного продукта конечными пользователями.

**Основные подходы планирования и разработки программных продуктов:**

1. Экспертный подход – проверка соответствия принципам обеспечения удобства пользования и корректного визуального представления в контексте функциональных требований посредством экспертной оценки;
2. Пользовательский подход – изучение опыта взаимодействия пользователя с приложением через имитацию поведения пользователей.

**Преимущества экспертного подхода:**

* Быстрота применения;
* Эксперты гарантировано понимают общие задачи программного продукта.

**Недостатки экспертного подхода:**

* Субъективизм (эксперты не является реальными пользователями);

**Преимущества пользовательского подхода:**

* Объективные результаты (участвуют реальные пользователи);
* Процесс легко измерим;

**Недостатки пользовательского подхода:**

* Длительность по времени;
* Дорогой (если пользователей привлекают на платной основе);
* Большое внимание следует уделить подборку пользователей.

**Возможные измерения при юзабилити:**

* Время выполнения задачи;
* Успешность выполнения задачи;
* Эффект первого впечатления.

15.02.2024

**Тестирование производительности**

**Тестирование производительности** – тестирование, целю которого служит определение скорости работы вычислительной системы или её частей под определёнными нагрузками.

Служит проверкой или подтверждением атрибутов качества системы, например, надёжность системы или её масштабируемость.

**Масштабируемость** – способность системы справляться с увеличением рабочей нагрузки при добавлении ресурсов.

**Способы тестирования параметров систем в определённой среде:**

* Определение рабочего количества пользователей приложения;
* Измерение времени выполнения различных операций системы;
* Определение производительности приложения при различных степенях нагрузки
* Определение допустимых границ производительности программы при разных уровнях нагрузки.

Тестирование производительности проводится для обеспечения бесперебойной работы и снижения стоимости владения ПО на всех этапах жизненного цикла решения.

**Этапы тестирования производительности:**

1. **Анализ системы и рассмотрение требований.**

Главное условие этапа – полностью законченное приложение или система. Правильно настроенная конфигурация ПО.

**С помощью анализа системы можно:**

* 1. Воспроизводить максимально объективные шаблоны поведения пользователей и профиля нагрузки;
  2. Определить необходимое количество тестовых данных;
  3. Настраивать способы мониторинга системы;
  4. Выявить потенциально слабые места ПО.

Подобранные на начальных этапах требования необходимо сравнивать с полученными результатами чтобы оценить поведение ПО целиком и по модулям, для выявления и устранения слабых мест системы.

**Критерии для измеряемых метрик:**

1. Количество пользователей;
2. Время отклика;
3. Запросы в секунду;
4. Транзакции в секунду;
5. Процент ошибок;
6. Процессов;
7. Оперативная память;
8. Жёсткий диск.
9. **Подготовка стратегии.**

Стратегия разрабатывается на основе детального анализа ПО и описывает подходы к тестированию производительности. Данная стратегия включает всё собранную информацию на этапе анализа системы: требования к ПО, конфигурацию тестового стенда, мониторинг, сценарии поведения пользователей, профиле и модели нагрузки, инструментах тестирования производительности, планируемых запусках тестов, виде предоставления результатов.

Конфигурация тестового стенда: факторы, влияющие на результаты нагрузочного тестирования, загруженность сети, наполненность БД.

Чтобы получить достоверные результаты, тестирование необходимо проводить на отдельном окружении с параметрами и конфигурацией, которые максимально приближены к характеристикам реальной системы.

Во время тестирования собирается статистика использования ПО, для создания профиля нагрузки – модели поведения пользователей.

1. **Настройка генератора нагрузки.**

**Генератор нагрузки** – виртуальная или физическая машина, расположенная максимально близко к серверу приложения. Это снижает искажение при подаче нагрузки. Вызванные задержками сети.

1. **Мониторинг серверов и генератора нагрузки.**

Мониторинг можно проводить с помощью специальных утилит для отслеживания потребления аппаратных ресурсов ПО.

1. **Подготовка тестовых данных.**

4 способа подготовки данных:

1. Код;
2. SQL-запросы;
3. API-запросы;
4. Интерфейс.
5. **Разработка нагрузочных скриптов.**

Процесс разработки скриптов:

1. Изучение сценарием – анализ подготовленных сценариев для определения действия каждого;
2. Создание тест-кейсов – каждый сценарий записывается в шаблон без параметров;
3. Отладка сценариев – запуск каждого сценария.
4. **Предварительные запуски тестов.**
5. **Тестирование.**

**Стресс тест** – производит с постепенно увеличивающейся нагрузкой на сервер и возрастает до тех пор, пока не будет достигнут один из сценариев остановки теста:

1. Превышение требуемых значений отклика в несколько раз;
2. Достижение критического уровня использования аппаратных ресурсов();
3. Количество ошибок HTTP превышает 1% от общего числа запросов;
4. Сбой системного ПО.

**Нагрузочное тестирование** – происходит в течении длительного промежутка времени (около 10 часов). Если система не выдерживает, то нагрузка уменьшается до 80%. Нагрузка выдаётся быстрее

**Проверка стабильности** – нагрузка выдаётся сразу и проводится длительно.

**Объёмное тестирования**– используется для оценки производительности работы ПО при увеличении данных, которые хранятся в БД приложения. Для проведения такого теста необходимо заполнить базу определённым объёмом информации.

**Проверка масштабируемости** – позволяет оценить способность решения увеличивать производительность пропорционально добавлению аппаратных ресурсов системы.

**Модульное тестирование** – нагрузка подаётся на отдельные компоненты ПО.

**Тест «Часа пик»** - проводится, чтобы проверить реакцию системы как в момент наибольшей загруженности, так и непосредственно после снижения нагрузки.

**Тестирование конфигурации** оценивает работу программного продукта при разных конфигурациях окружения.

1. **Анализ результата и подготовка отчёта.**

Отчёт по тестированию содержит следующую информацию:

* 1. Цель проведения тестирования;
  2. Конфигурация тестового стенда и генератора нагрузки;
  3. Требования к ПО;
  4. Сценарии поведения пользователей, профиль нагрузки;
  5. Статистику по ключевым характеристикам производительности (время отклика, количество запросов в секунду, количество транзакций в секунду);
  6. Данные о максимально возможном количестве одновременно работающих пользователей, при котором решение будет справляться с нагрузкой;
  7. Сведения о количестве и типах ошибок HTTP;
  8. Графики, показывающие зависимость производительности системы от количества одновременно работающих пользователей;
  9. Выводы о производительности системы в целом и о слабых местах, если они будут обнаружены;
  10. Рекомендации по улучшению производительности ПО.

16.02.2024

**Регрессионное тестирование**

**Регрессонное тестирование** – тип тестирование ПО для подтверждения, что недавнее изменение программы не повлияло отрицательно на существующие функции. Полный или частичный выбор уже выполненных тестовых случаев, которые выполняются повторно, чтобы убедиться, что существующие функции работают нормально.

Проводится в самом конце.

Необходимость регрессионного тестирования возникает всякий раз, когда требуется изменить код, и нам нужно проверить, влияет ли изменённый код на другую часть программного приложения или нет. Необходимо при добавлении новой функции в программное приложение и для устранения дефектов, а также устранения проблем с производительностью.

**Обслуживание ПО** – деятельность, которая включает улучшения, исправления ошибок, оптимизацию и удаление существующих функций. Эти модификации могут привести к неправильной работе системы.

**Выбор регрессионного теста** – метод, при котором выполняются некоторые выбранные тестовые примеры из набора тестов, чтобы проверить, влияет ли изменённый код на программное приложение или нет.

Тестовые наборы бывают повторно используемые и устаревшие.

**Управление конфигурацией** во время регрессионного тестирования становится обязательным в гибких средах, где код постоянно модифицируется.

**Обеспечение эффективности регрессионных тестов:**

* Код, подвергаемый регрессионному тестированию, должен находиться под инструментом управления конфигурацией;
* На этапе регрессионного тестирования нельзя допускать никаких изменений в коде. Код регрессионного тестирования должен быть защищён от изменений, вносимых разработчиком;
* База данных, используемая для регрессионного тестирования, должна быть изолирована. Никакие изменения БД не должны быть разрешены.

**Повторное тестирование** – проверка, что тестируемый код исправен.

**Регрессионное тестирование** – проверка, что новый код не влияет на другие части ПО.

Основные проблемы регрессионного тестирования:

* При последовательных запусках регрессии наборы тестов становятся довольно большими. Из-за ограничений по времени и бюджету весь набор регрессионных тестов не может быть выполнен.
* Сведение к минимуму набора тестов при достижении максимального покрытия тестами остаётся сложной задачей.
* Определение частоты регрессионных тестов, т.к. после каждой модификации сборки…

Сравнительная характеристика утилит ля отслеживания потребления аппаратных ресурсов:

1. Nmon - легковесное Linux решение. Предоставляет детальную информацию о производительности системы Ограниченные возможности визуализации и масштабирования. Записывает данные в файл, который можно посмотреть с помощью nmonchar. Ограниченный функционал, отсутствие собственного графического интерфейса.
2. Perfomance Monitor - интегрирован в операционные системы Windows. Предоставляет детальный мониторинг ресурсов. Меньшая гибкость по сравнению с некоторыми другими утилитами
3. Zabbix - мониторинг с открытым исходным кодом. Позволяет отслеживать широкий спектр ресурсов. Требует усилий для настройки и поддержки, особенно в больших средах.
4. Grafana - платформа визуализации данных. Позволяет интегрировать данные из различных источников. Требует дополнительной интеграции с другими инструментами мониторинга для сбора данных
5. New Relic - Облачное решение. Предоставляет мониторинг производительности приложений и инфраструктуры. Требует подписки и доступа к интернету для использования. Более дорогое в сравнении с локальными решениями

Сравнение SQL server profiler и Postgre:

1. PostgreSQL – свободно распространяемая СУБД. Поддерживается как Linux, так и Windows.
2. SQL Server Profiler – считается устаревшей. Не СУБД. Позволяет создавать и отслеживать трассировки. Встроена в SSMS.