## 

[**1. Цели и задачи управления процессом разработки ПО.**](#_heading=h.fw2g0vk0tyn9) **2**

[**2. Жизненный цикл программных систем.**](#_heading=h.mj5pucq0zyv2) **3**

[**3. Баг-трекинг системы.**](#_heading=h.5ytpsze9kdh) **4**

[**4. Системы управления версиями (обзор систем).**](#_heading=h.dd5j32wetpqm) **5**

[**5. Система управления версиями и поиска ошибок Redmine.**](#_heading=h.hp6ep1lw4y6) **6**

[**6. Системы управления версиями Git.**](#_heading=h.b5bkhrhjwbpr) **7**

[**7. Универсальная среда программирования Eclipse.**](#_heading=h.7jpepm5o6we1) **9**

[**8. Платформы и инструменты для создания веб-приложений.**](#_heading=h.qfpf9xnentfh) **10**

[**9. Создание кроссплатформенных приложений.**](#_heading=h.su8ytoek4hx1) **14**

[**10. Характеристики и возможности открытой среды юнит-тестирования приложений NUnit.**](#_heading=h.6qoxoqlgmf0t) **15**

[**11. Системы автоматизированного тестирования.**](#_heading=h.1lmwmvu0knga) **16**

[**12. Selenium – библиотека для тестирования веб-приложений.**](#_heading=h.bk77xu5k4hv) **18**

[**13. Программы для создания инсталляторов.**](#_heading=h.riy2jx6fnny6) **20**

[**14. Рефакторинг кода.**](#_heading=h.b60502hc25hk) **21**

[**15. Рефакторинг кода на Java и Python.**](#_heading=h.qaunr3j0b6ie) **23**

[**16. Автоматизированный рефакторинг в Java IDE.**](#_heading=h.7d3bda7z0cuj) **27**

[**17. Система управления базами данных Redis.**](#_heading=h.fqxq9g1dh4e) **29**

[**18. Платформа API Postman.**](#_heading=h.qdi7t4n06oq3) **31**

[**19. Программное обеспечение для автоматизации развёртывания и управления приложениями Docker.**](#_heading=h.2652imlghlwg) **32**

[**20. Инструменты оптимизации работы высоконагруженных систем.**](#_heading=h.7m504wdw00c5) **33**

## 

## 1. Цели и задачи управления процессом разработки ПО.

***Управление разработкой программного обеспечения*** — особый вид управления проектами, в рамках которого происходит планирование, отслеживание и контроль за проектами по разработке программного обеспечения. Ключевым моментом в управлении проектом по разработке программного обеспечения является правильный выбор метода разработки.

***Целью*** управления процессом разработки ПО является определение объема и содержания работ, необходимых для успешного осуществления проекта, оценка затрат и составление графика работ. Планирование прежде всего начинается с анализа требований, определяющих свойства и функции создаваемого программного обеспечения. Затем определяются задачи, выполнение которых приведет к успешному завершению проекта.

***Основная задача*** заключается в поддержании соответствия действий команды текущему состоянию проекта. В случае отклонения проекта от плана управляющий проектом может оперативно исправлять выявленные ошибки. Отслеживание состояния проекта включает в себя регулярные встречи с командой для обсуждения текущего состояния проекта.

## 

## 2. Жизненный цикл программных систем.

В соответствии со стандартом СТБ ИСО/МЭК 12207-2003 под ***жизненным циклом*** (ЖЦ) программного средства или системы подразумевается совокупность процессов, работ и задач, включающая в себя разработку, эксплуатацию и сопровождение ПС или системы, охватывающая их жизнь от формулирования концепции до прекращения использования. В соответствии с данным стандартом жизненный цикл программных средств состоит из процессов. Каждый процесс ЖЦ разделен на набор работ. Каждая работа разделена на набор задач.

Процессы ЖЦ ПС делятся на следующие группы:

* основные;
* вспомогательные;
* организационные.

К основным процессам ЖЦ относятся следующие процессы:

* заказ;
* разработка;
* поставка;
* эксплуатация;
* сопровождение.

***Процесс разработки*** состоит из работ и задач ,выполняемых разработчиком, и включает работы по анализу требований, проектированию, программированию, сборке, тестированию, вводу в действие и приемке ПС.

***Модель жизненного цикла*** – это совокупность процессов, работ и задач ЖЦ, отражающая их взаимосвязь и последовательность выполнения.

1. Waterfall (каскадная модель, или «водопад»)
2. V-образная модель (разработка через тестирование)
3. Incremental Model (инкрементная модель)
4. Iterative Model (итеративная модель)
5. Spiral Model (спиральная модель)

## 

## 3. Баг-трекинг системы.

***Система отслеживания ошибок*** — прикладная программа, разработанная с целью помочь разработчикам программного обеспечения учитывать и контролировать ошибки (баги), найденные в программах, а также следить за процессом устранения этих ошибок.

Главный компонент такой системы — база данных, содержащая сведения об обнаруженных ошибках.

Эти сведения могут включать в себя:

* кто сообщил о проблеме;
* дата и время, когда была обнаружена проблема;
* серьёзность(критичность) проблемы;
* описание неправильного поведения программы;
* кто занимается устранением (разрешением) проблемы;
* состояние ошибки.

Как правило, система отслеживания ошибок использует тот или иной вариант «жизненного цикла» ошибки, стадия которого определяется текущим состоянием, или статусом, в котором находится ошибка.

Система может предоставлять администратору возможность настроить, какие пользователи могут просматривать и редактировать ошибки в зависимости от их состояния, переводить их в другое состояние или удалять. В корпоративной среде система отслеживания ошибок может использоваться для получения отчетов, показывающих продуктивность программистов при исправлении ошибок. Однако, часто такой подход не даёт достаточно точных результатов, потому что разные ошибки имеют различную степень серьезности и сложности. При этом серьезность проблемы не имеет прямого отношения к сложности устранения ошибки

Примерами таких систем являются - Jira, Trello, Redmine, YouTrack, Mantis.

## 

## 4. Системы управления версиями (обзор систем).

***Система управления версиями***— программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы.

***Git*** — система управления версиями с распределенной архитектурой. В Git каждая рабочая копия кода сама по себе является репозиторием. Это позволяет всем разработчикам хранить историю изменений в полном объеме.

Разработка в Git ориентирована на обеспечение высокой производительности, безопасности и гибкости распределенной системы. Она превосходно работает под управлением различных операционных систем и может применяться со множеством интегрированных сред разработки (IDE).

***Mercurial*** была выпущена одновременно с Git. Это также распределенная система контроля версий. Система контроля версий Mercurial отличается от других систем контроля версий тем, что главным образом она написана на Python (а не С). Однако, некоторые части выполнены в качестве модулей-расширений на C. Поскольку система децентрализованная и написана на Python, многие Python-программисты склоняются к переходу на Mercurial.

Mercurial сохранила некоторые характеристики SVN, являясь при этом распределенной системой, и благодаря этой схожести, порог вхождения у нее ниже для тех, кто уже знаком с SVN. Также документация по Mercurial более полная, что помогает быстрее освоиться с различиями.

## 

## 5. Система управления версиями и поиска ошибок Redmine.

Система отслеживания ошибок (англ. bug tracking system) — прикладная программа, разработанная с целью помочь разработчикам программного обеспечения (программистам, тестировщикам и др.) учитывать и контролировать ошибки (баги), найденные в программах, а также следить за процессом устранения этих ошибок. Сокращенно BTS.

Redmine - открытое приложение для управления проектами, включающее в себя систему отслеживания ошибок. Функциональность решения такова, что оно подойдёт достаточно крупным компаниям. Некоторые из основных особенностей Redmine:

* Поддержка нескольких проектов.
* Контроль доступа на основе ролей.
* Гибкая система отслеживания задач.
* Диаграмма Ганта.
* Календарь.
* Новости, документы и управление файлами.
* Ленты и уведомления по электронной почте.
* Вики и форумы по проекту.
* отслеживание времени.
* Пользовательские поля для вопросов, временных записей, проектов и пользователей.
* Интеграция SCM (SVN, CVS, Git, Mercurial, Bazaar и Darcs).
* Отслеживание выпусков.
* Создание задач по электронной почте.
* Поддержка нескольких аутентификаций LDAP.
* Самостоятельная регистрация пользователя.
* Многоязычный интерфейс.
* Поддержка нескольких баз данных.

Задачи в Redmine могут быть взаимосвязаны. Предусмотрены следующие варианты связей: дублирование, простая связка, блокировка, предшествование, следование. Это охватывает практически все возможные варианты и позволяет оптимизировать работу в том числе и по исправлению ошибок. Код программы опубликован на GitHub и распространяется по GPL v.2.

Плюсы:

* Вся настройка через админ-панель
* Множественные роли в проектах
* Вложенные проекты неограниченное глубины
* Пакетное редактирование задач
* Настройка видимостей полей
* Просмотр вложений
* Поддержка плагинов; миграторы с других БТС

Минусы:

* Определенная сложность установки (нужен Ruby + ряд зависимостей)
* Нет прав на отдельные типы в Workflow

## 6. Системы управления версиями Git.

Git - абсолютный лидер по популярности среди современных систем управления версиями. Это развитый проект с активной поддержкой и открытым исходным кодом. Система Git была изначально разработана в 2005 году Линусом Торвальдсом - создателем ядра операционной системы Linux. Git применяется для управления версиями в рамках колоссального количества проектов по разработке ПО, как коммерческих, так и с открытым исходным кодом. Система используется множеством профессиональных разработчиков программного обеспечения. Она превосходно работает под управлением различных операционных систем и может применяться со множеством интегрированных сред разработки (IDE).

Git - система управления версиями с распределенной архитектурой. В отличие от некогда популярных систем вроде CVS и Subversion (SVN), где полная история версий проекта доступна лишь в одном месте, в Git каждая рабочая копия кода сама по себе является репозиторием. Это позволяет всем разработчикам хранить историю изменений в полном объеме.

Разработка в Git ориентирована на обеспечение высокой производительности, безопасности и гибкости распределенной системы.

Основные команды Git:

Ежедневная работа:

* Обновляем репозиторий и рабочую копию: git pull
* Добавляем файл в проект: git add hello.cpp
* Делаем коммит: git commit -m “описание того, что сделали”
* Отдаем изменения во внешний репозиторий git push

Ветка в Git — это простой перемещаемый указатель на один из таких коммитов. По умолчанию, имя основной ветки в Git — master. Как только вы начнёте создавать коммиты, ветка master будет всегда указывать на последний коммит. Каждый раз при создании коммита указатель ветки master будет передвигаться на следующий коммит автоматически.

Создание новой ветки: git branch testing

Переключение веток: git checkout testing

HEAD- указатель на текущую ветку, которая, в свою очередь, является указателем на последний коммит, сделанный в этой ветки.

Теги - ссылки, указывающие на определенные точки в истории Git. Команда git tag обычно используется для захвата некой точки в истории, которая используется для релиза нумерованной версии. Для создания нового тега команда: git tag <tag\_name>

Достоинства Git:

* Почти все отрицательные черты CVS/SVN устранены;
* Высокая скорость работы распределенной системы контроля версий;
* Легкость проведения различных операций с ветками проекта;
* Пользователи могут получить доступ к полному дереву истории в режиме офлайн;
* Предлагает высоко распределенную одноранговую модель.

Недостатки Git:

* Высокий порог вхождения для пользователей SVN;
* Ограниченная поддержка Windows по сравнению с Linux.

## 

## 7. Универсальная среда программирования Eclipse.

Eclipse – это интегрированная среда разработки мультиплатформенных приложений с открытым исходным кодом. Главным образом это приложение работает как платформа для программирования, и оно может компилировать и отлаживать программы на различных языках, но наибольшей популярностью она пользуется у программистов на Java. Тем не менее модульность этой среды позволяет использовать ее для программирования на C, Python и на многих других языках.

Структура среды понятна, все функции разделены на несколько различных окон, и позволяют точную настройку. Кроме этого, несмотря на то, что редактор включает массу встроенных инструментов для отладки и анализа ошибок, он также позволяет работать с предпросмотром проектов, поэтому вы сможете увидеть, как будет выглядеть Java-приложение, даже не компилируя и не запуская его.

Стандартная версия Eclipse включает интегрированную среду разработки (IDE) для Java, а также инструменты разработки (JDT), которые являются необходимыми плагинами для программирования на этом языке.

## 

## 8. Платформы и инструменты для создания веб-приложений.

Существует множество платформ и инструментов для создания веб-приложений. Рассмотрим 5 наиболее часто используемых:

* Sublime Text
* Bootstrap
* React
* Postman
* Django

**Sublime Text:**

Sublime Text — кроссплатформенный текстовый редактор для написания программного кода на различных языках программирования (Groovy, Erlang, C+, Java и т.д.), а также верстки веб-документов. Это проприетарное, распространяемое по платной лицензии ПО, имеющее бесплатную версию для ознакомления. За счет широкой функциональности, которую можно расширять путем подключения плагинов, а также кроссплатформенности Sublime Text популярен среди программистов и веб-разработчиков.

Этот текстовый редактор (правильнее его назвать редактором кода) изначально создавался с ориентацией на программистов и веб-разработчиков. Отсюда и соответствующий функционал, включающий следующие ключевые возможности:

* Настраиваемый интерфейс
* Множественное выделение
* Автоматическое заполнение
* Подсветка синтаксиса и высокая контрастность
* Заготовки (сниппеты).
* Раздельное редактирование
* Подключение плагинов
* Поддержка систем сборки

Преимущества Sublime Text:

* Широкая функциональность
* Поддержка языков
* Быстродействие
* Простой и понятный интерфейс
* Кроссплатформенность
* Широкая поддержка

Недостатки Sublime Text

* Платная лицензия
* Время загрузки программы
* Ошибки в работе с плагинами

**Bootstrap:**

Bootstrap — это открытый и бесплатный HTML, CSS и JS фреймворк, который используется веб-разработчиками для быстрой вёрстки адаптивных дизайнов сайтов и веб-приложений.

Основная область его применения – это фронтенд разработка сайтов и интерфейсов админок. Среди аналогичных систем (Foundation, UIkit, Semantic UI, InK и др.) фреймворк Bootstrap является самым популярным.

Фреймворк Bootstrap – это набор набор CSS и JavaScript файлов. Чтобы его использовать эти файлы необходимо просто подключить к странице. После этого станут доступны инструменты данного фреймворка: колоночная система (сетка Bootstrap), классы и компоненты.

Преимущества Bootstrap:

* высокая скорость создания качественной адаптивной вёрстки даже начинающими веб-разработчиками
* кроссбраузерность и кроссплатформенность
* наличие большого количество готовых хорошо продуманных компонентов, протестированных огромным сообществом веб-разработчиков на различных устройствах
* возможность настройки под свой проект,
* низкий порог вхождения
* однородность дизайна и его согласованность между различными компонентами
* наличие огромного количества сообществ и обучающих материалов

Недостатки:

* более большой размер конечных css и js-файлов проекта, чем они получились, если бы мы всё это создавали самостоятельно
* сложность использования Bootstrap для вёрстки сайтов с уникальным дизайном, т.к. разработка в этом случае будет сопровождаться значительным переписыванием его кода и простой настройкой Bootstrap переменных тут уже не обойтись.

**React:**

React — это JavaScript-библиотека для создания пользовательских интерфейсов. React часто называют фреймворком, но это ошибка. Во-первых, его использование ни к чему не обязывает, не формирует «фрейм» проекта. Во-вторых, React выполняет единственную задачу: показывает на странице компонент интерфейса, синхронизируя его с данными приложения, и только этой библиотеки в общем случае недостаточно для того, чтобы полностью реализовать проект.

Преимущества:

* Популярность
* Огромное сообщество. Документация поддерживается и обновляется, существует в том числе на русском языке.
* Развитая экосистема. Сейчас сформировалась целая экосистема: десятки библиотек и других инструментов, которые можно использовать в разработке.
* Простота создания интерфейса. С помощью React-компонентов можно быстро и легко собрать интерактивный, чутко реагирующий на любые изменения интерфейс сайта любой сложности.
* Реактивность. Фреймворк реагирует на обновление компонента и автоматически отображает его изменения в дереве документа.
* Эффективность. Благодаря виртуальному DOM фреймворк экономит ресурсы. Чтобы изменить состояние элементов в интерфейсе, не нужно полностью перезагружать все DOM-дерево, лишний раз тратить трафик пользователя и нагружать браузер. Изменяются только конкретные элементы, это происходит через виртуальное DOM-дерево — сайты становятся «легче» и удобнее в работе.
* Высокая скорость работы. Сайты, написанные на React, быстрые и отзывчивые благодаря виртуальному DOM.
* Удобная отладка.

Недостатки React:

* Запутанность синтаксиса. Технология JSX удобная и широко используется, но сначала она может вызвать сложности в изучении.
* Неполное соответствие MVC. Существуют фреймворки, которые полностью реализуют модель MVC. React отвечает только за интерфейс, за часть View. Поэтому для реализации модели потребуется подбирать стек из разных технологий. React можно использовать практически с любым стеком.

**Postman:**

Postman — это HTTP-клиент для тестирования API. HTTP-клиенты тестируют отправку запросов с клиента на сервер и получение ответа от сервера.

Методы Postman:

Чаще всего в работе API используется архитектура RESTful. В этой архитектуре есть четыре стандартных метода запросов к серверам по HTTP:

* POST — создание объекта и отправка данных на сервер;
* GET — получение информации с сервера;
* PUT — обновление объекта;
* DELETE — удаление объекта.

Особенности:

* Простой в использовании API клиент
* Функциональный и приятный UI.
* Может использоваться как для ручного, так и для автоматизированного тестирования API.
* Поддерживает интеграции с другими инструментами (например, поддерживает Swagger и RAML)
* Может быть запущен в Windows, Linux, MacOS.
* Не требует знания языков программирования.
* Предоставляет возможность легко экспортировать коллекции запросов, наборы тестов.
* Интегрируется с CI/CD инструментами (например, с Jenkins, TeamCity и т.п.)
* API Posman-a подробно документирован.
* Позволяет выполнять API автотесты.

**Django:**

Django — это программный каркас с богатыми возможностями, подходящий для разработки сложных сайтов и веб-приложений, написанный на языке программирования Python.

Django — фреймворк для веб-приложений на языке Python. Один из основных принципов фреймворка — DRY (don't repeat yourself).

Плюсы Джанго:

* Принцип «Всё включено». Большинство инструментов для создания приложения — часть фреймворка, а не поставляются в виде отдельных библиотек.
  + ORM
  + Миграции базы данных
  + Аутентификация пользователя
  + Панель администратора
  + Формы
* Стандартизированная структура. Django как фреймворк задаёт структуру проекта. Она помогает разработчикам понимать, где и как добавлять новую функциональность.
* Приложения Django позволяют разделить проект на несколько частей. Приложения устанавливаются путём добавления в settings.INSTALLED\_APPS для лучшей регистрации готовых решений.
* Безопасный по умолчанию
* REST Framework для создания API. Django REST Framework, который часто сокращают до «DRF», является библиотекой для построения API.
* GraphQL фреймворк для создания API - это язык запросов, который позволяет обмениваться связанными данными гораздо проще.

Недостатки Джанго:

* Django ORM значительно уступает последней SQLAlchemy.
* Django развивается медленно

## 

## 9. Создание кроссплатформенных приложений.

Кроссплатформенная разработка приложений относится к разработке программного обеспечения, которое может работать на нескольких устройствах. Совместимость с несколькими платформами — очень желательная черта в сфере разработки программного обеспечения. С другой стороны, сложности разработки программного обеспечения часто означают снижение переносимости. То есть разработчики обычно не могут использовать одно и то же программное обеспечение в различных вычислительных средах.

Преимущества кроссплатформенной разработки:

* ***Скорость***  
  Скорость гарантирована при использовании кроссплатформенной разработки. Это гораздо более быстрый способ разработки. При разработке нативных приложений компании, которые хотят создавать мобильные, должны создавать два отдельных приложения с собственными технологиями для каждой платформы. Кросс-платформенная разработка устранит эти недостатки.
* ***Удобство***  
  Точно так же работа с единой кодовой базой резко повышает производительность на протяжении всего жизненного цикла разработки программного обеспечения. Разработчикам не придется писать уникальный код для каждой платформы.
* ***Расходы***  
  Дает финансовые выгоды. Менее напряженный бюджет. Разработка нативных приложений может быть ресурсоемкой для тех, кто хочет, чтобы их приложение было широко доступно на нескольких платформах.

Недостатки кроссплатформенной разработки приложений

* ***Производительность***  
  Дополнительный уровень абстракции, который должен реализовать подходы к разработке для обеспечения совместимости с широким спектром устройств, заметно снижает производительность.
* ***Согласованность***  
  Поскольку неродные компоненты стремятся интегрироваться с родными компонентами в кроссплатформенной разработке, будут возникать несоответствия. Кроссплатформенная разработка требует постоянной корректировки дизайна и функциональных возможностей кроссплатформенного приложения, чтобы гарантировать, что оно может адаптироваться к разнообразию платформ, на которых оно предназначено для работы.
* ***Пользовательский опыт***  
  Пользовательский опыт (UX), например, будет большей ценой недостатков, описанных выше. UX является важной составляющей в удовлетворении потребностей клиентов. Пользователи, у которых есть положительный опыт использования программного продукта, могут сыграть фундаментальную роль в развитии бизнеса.

## 

## 10. Характеристики и возможности открытой среды юнит-тестирования приложений NUnit.

NUnit - открытая среда юнит-тестирования приложений для .NET, которая позволяет создавать автоматические тесты. Даный вид тестов обладает рядом преимуществ:

* Высокое качество программы;
* Снижение стоимости;
* Безопасность регрессии сети

Чем выше качество программы, тем меньше средств затрачивается на устранение недостатков проекта. То есть, если найти недостатки в проекте на раннем этапе, решить их будет дешевле.

Возможности:

* Тесты можно запускать из средства выполнения консоли, в Visual Studio через адаптер тестирования или через сторонние средства выполнения.
* Тесты можно запускать параллельно.
* Сильная поддержка для тестов на основе данных.
* Поддерживает несколько платформ, включая .NET Core, Xamarin Mobile, Compact Framework и Silverlight.
* Каждый тестовый пример можно добавить в одну или несколько категорий, чтобы обеспечить выборочный запуск.

NUnit предоставляет средство запуска консоли (nunit3-console.exe), которое используется для пакетного выполнения тестов. Средство выполнения консоли работает через механизм тестирования NUnit Test Engine, который дает ему возможность загружать, исследовать и выполнять тесты. Когда тесты должны выполняться в отдельном процессе, движок использует программу nunit-agent для их запуска.

Средство запуска NUnitLite может использоваться в ситуациях, когда более подходящим является средство запуска. Он позволяет разработчикам создавать самовыполняющиеся тесты.

## 

## 11. Системы автоматизированного тестирования.

**Автоматизированное тестирование программного обеспечения** — часть процесса тестирования на этапе контроля качества в процессе разработки программного обеспечения. Оно использует программные средства для выполнения тестов и проверки результатов выполнения, что помогает сократить время тестирования и упростить его процесс.

**Selenium** — Один из наиболее популярных фреймворков для автоматизации тестирования сайтов и веб-приложений. Selenium помогает тестировщикам писать тесты на разных языках программирования, таких как Java, PHP, C #, Python, Groovy, Ruby и Perl. Selenium предлагает функции записи и воспроизведения для написания тестов без изучения Selenium IDE. Среди плюсов — гибкость, возможность написания сложных скриптов для автоматизации. Среди минусов — достаточно высокая квалификация тестировщика. Специалист по автоматизации должен не только обладать повышенными знаниями в разработке программного обеспечения, но и быть готовым ко времязатратному написанию специальных библиотек и фреймов, обеспечивающих выполнение необходимых функций в процессе автоматизированного тестирования.

**Watir** - инструмент для автоматизированного тестирования веб-приложений, использующий в своей работе библиотеки Ruby. Имеет открытый исходный код, есть возможность кроссбраузерной работы во многих современных браузерах: Opera, Firefox, IE, headless-браузерах. Watir поддерживает тестирование, управляемое данными. Также он интегрирован с инструментами BBD (Cucumber, RSpec, Test/Unit).

**HP QuickTest Professional** был переименован в HPE Unified Functional Testing. HPE UFT предлагает автоматизацию тестирования для функционального и регрессионного тестирования для программных приложений. Язык сценариев Visual Basic Scripting Edition используется этим инструментом для регистрации процессов тестирования и управления различными объектами и элементами управления при тестировании приложений.

QTP предлагает различные функции, такие как:

* Интеграция с Mercury Business Process Testing и Mercury Quality Center
* Уникальное распознавание смарт-объектов
* Механизм обработки ошибок
* Создание параметров для объектов, контрольных точек и таблиц, управляемых данными
* Автоматизированная документация

**TestComplete** — это функциональная платформа тестирования, которая предлагает различные решения для автоматизации тестирования настольных, мобильных приложений компанией SmartBear Software.

TestComplete предлагает следующие функции:

* Тестирование GUI
* Поддержка языка сценариев — JavaScript, Python, VBScript, JScript, DelphiScript, C ++ Script и C# Script
* Тестовый визуализатор
* Скриптовое тестирование
* Тестовая запись и воспроизведение

Если сравнивать TestComplete с UTF, то он схож с функцией распознавания объектов GUI. В результате производится автоматическое обнаружение и обновление элементов пользовательского интерфейса.

**Ranorex** Studio предлагает инструменты автоматизации тестирования, которые охватывают тестирование всех десктопных и мобильных приложений.

Ranorex предлагает следующие функции:

* Распознавание графического интерфейса пользователя
* Многоразовые тестовые коды
* Обнаружение ошибок
* Интеграция с различными инструментами
* Запись и воспроизведение

Очередной плюс — возможность создавать тестовые сценарии без написания кода. Поддерживается интеграция с Selenium.

## 

## 12. Selenium – библиотека для тестирования веб-приложений.

Selenium - это набор программ с открытым исходным кодом, которые применяют для тестирования веб-приложений и администрирования сайтов локально и в сети. Программы Selenium позволяют автоматизировать действия браузера.

Преимущества Selenium

* Бесплатный
* Selenium WebDriver это "стандартный" драйвер (W3C приняла WebDriver за основу при разработке стандарта интерфейса для управления браузером)
* Развитое сообщество
* Поддержка многих языков программирования

Selenium WebDriver

библиотека для управления браузерами, основной продукт комплекта Selenium. Представляет из себя семейство драйверов для разных браузеров (Firefox, Edge, Google Chrome/Chromium, Internet Explorer, Safari, Opera) и набор клиентских библиотек на разных языках программирования для работы с драйверами. WebDriver поддерживает работу с языками Java, .Net (C#), Python, Ruby, JavaScript.

Selenium IDE

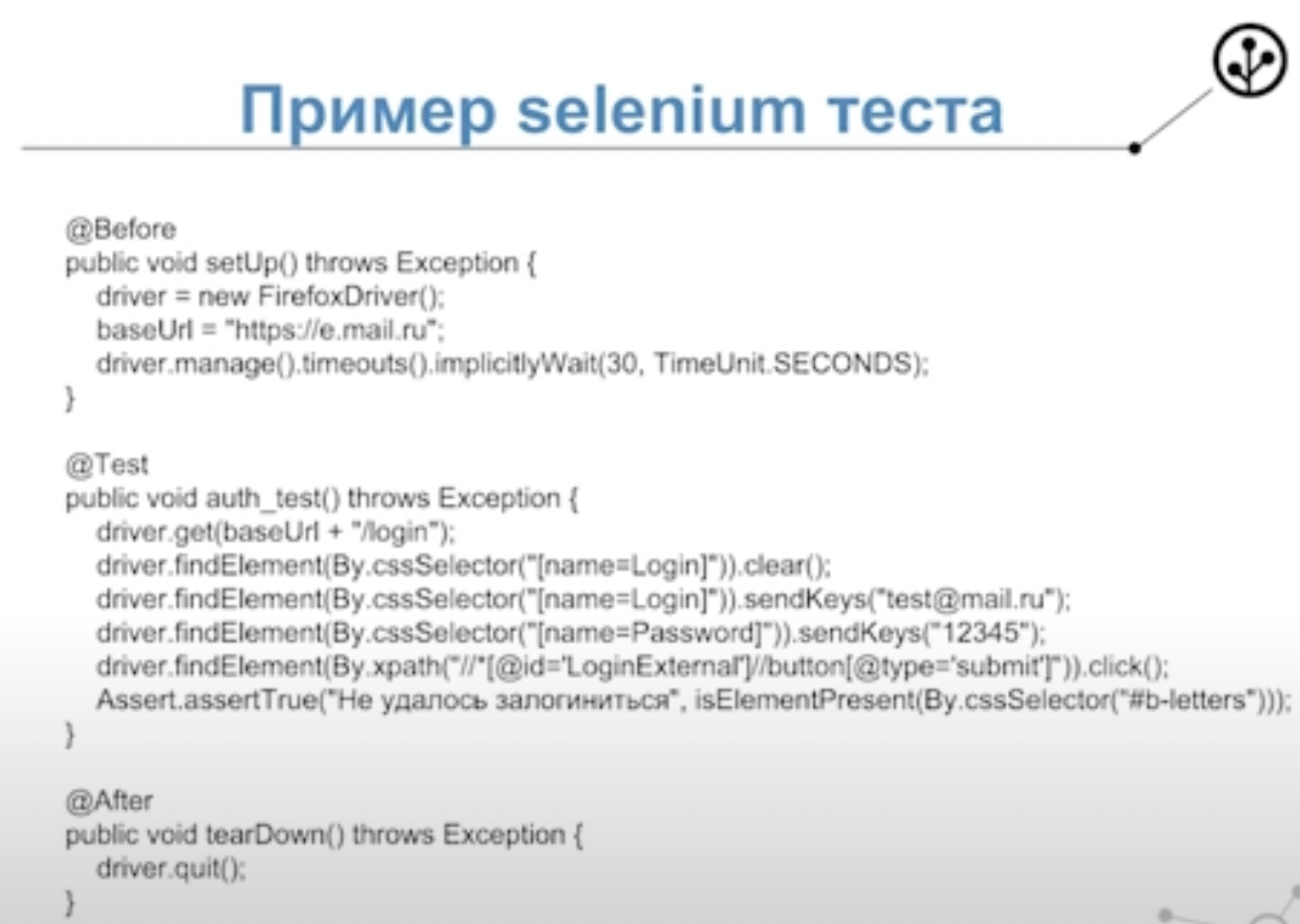
плагин для браузера Firefox для записи действий пользователя (тестировщика) и их воспроизведения для тестирования. Является библиотекой Selenium с графическим интерфейсом и возможностями для работы со сценариями тестирования веб-страниц. IDE генерирует код для Selenium RC или Selenium WebDriver, который повторяет записанные действия пользователей.

Selenium RC (remote control)

предыдущий основной продукт Selenium до появления WebDriver в 2007 году. Программа, называемая также Selenium 1.0, являлась средством удаленного управления браузером, но по функциональности сильно уступала WebDriver (Selenium 2.0). Selenium RC продолжает поставляться в дистрибутиве WebDriver, но продукт не развивается - при необходимости сложных тестов вне ограничений первой версии пользователям предлагают воспользоваться второй.

Selenium Grid

кластер из нескольких Selenium-серверов, которые позволяют управлять браузером удаленно по сети. Grid позволяет организовать сеть, в которой можно запускать большое количество браузеров на большом количестве компьютеров. Параллельное тестирование позволяет тестировщикам экономить время.



## 

## 13. Программы для создания инсталляторов.

Наиболее популярные программы для создания инсталляторов:

1. InstallShield - мощное и профессиональное средство для создания инсталляторов, используется многими крупными компаниями.
2. NSIS (Nullsoft Scriptable Install System) - бесплатное и открытое средство для создания инсталляторов, имеет простой скриптовый язык.
3. Inno Setup - бесплатное и открытое средство для создания инсталляторов для Windows, имеет простой синтаксис и множество готовых скриптов.
4. Advanced Installer - коммерческое средство, предлагает множество возможностей для создания инсталляторов и интеграции с системами управления лицензиями.
5. WiX - открытое средство для создания инсталляторов для Windows, использует язык XML для создания скриптов.

Существует несколько типов инсталляторов, которые можно классифицировать по различным критериям:

1. По типу установки:

* Инсталляторы пакетов: это инсталляторы, которые используются для установки определенного программного обеспечения. Они могут включать в себя несколько файлов и компонентов, которые необходимо установить на компьютер пользователя.
* Инсталляторы обновлений: это инсталляторы, которые используются для обновления уже установленного программного обеспечения.
* Инсталляторы деинсталляции: это инсталляторы, которые используются для удаления установленного программного обеспечения с компьютера пользователя.

2. По операционной системе:

* Инсталляторы Windows: это инсталляторы, которые специфичны для операционной системы Windows и используют технологии, такие как MSI или EXE.
* Инсталляторы Mac: это инсталляторы, которые специфичны для операционной системы Mac и используют технологии, такие как PKG или DMG.
* Инсталляторы Linux: это инсталляторы, которые специфичны для операционной системы Linux и используют технологии, такие как DEB или RPM.

3. По типу упаковки:

* Инсталляторы с установкой в системную папку: Это инсталляторы, которые устанавливают программное обеспечение в системные папки, такие как Program files или Application.
* Инсталляторы портативные: Это инсталляторы, которые устанавливают программное обеспечение на внешнюю носитель, такой как USB-накопитель, и могут быть запущены на любом компьютере без необходимости установки. Такие инсталляторы часто используются для программ, которые не требуют регистрации или установки в системную папку.

В зависимости от того, какой тип инсталлятора вы выберете, он может иметь различные особенности, плюсы и минусы, которые необходимо учитывать при создании инсталляционного пакета.

## 14. Рефакторинг кода.

Рефакторинг — это переработка исходного кода программы, чтобы он стал более простым и понятным.

Рефакторинг не меняет поведение программы, не исправляет ошибки и не добавляет новую функциональность. Он делает код более удобочитаемым.

В процессе работы даже изначально хорошо структурированный исходник становится беспорядочным и непонятным. Программисты знают, как легко завязнуть в этом хаосе. Причём неважно, чужой это код или собственный.

Чтобы решить все эти проблемы, делается рефакторинг программы. В новом проекте он нужен, чтобы:

* сохранить архитектуру проекта, не допустить потери структурированности;
* упростить будущую жизнь разработчиков, сделать код понятным и прозрачным для всех членов команды;
* ускорить разработку и поиск ошибок.

Но любое приложение со временем устаревает: язык программирования совершенствуется, появляются новые функции, библиотеки, операторы, делающие код проще и понятнее. То, что год назад требовало пятидесяти строк, сегодня может решаться всего одной.

Поэтому даже идеальная когда-то программа со временем требует нового рефакторинга, обновляющего устаревшие участки кода.

Признаки, показывающие, что назрела необходимость в рефакторинге:

* Программа работает, но даже небольшие доработки сильно затягиваются из-за того, что каждый раз приходится долго разбираться в коде.
* Разработчик постоянно не может точно сказать, сколько времени ему нужно на выполнение задачи, потому что “там надо вначале разбираться”.
* Одинаковые изменения приходится вносить в разные места текста программы.

Такой код нужно срочно рефакторить, иначе он будет тормозить реализацию проекта и затруднять внесение правок. Вообще рефакторинг нужно проводить постоянно. Чистить можно всё, но в первую очередь эти проблемы:

1. **Мёртвый код.** Переменная, параметр, метод или класс больше не используются: требования к программе изменились, но код не почистили. Такие элементы или участки текста нужно удалить.
2. **Дублирование**. Один и тот же код выполняет одно и то же действие в нескольких местах программы. Вынесите эту часть в отдельную функцию.
3. **Имена переменных, функций или классов не передают их назначение.** Имена должны сообщать, почему элемент кода существует, что он делает и как используется. Если видите, что намерения программиста непонятны без комментария, — рефакторьте.
4. **Слишком длинные функции и методы.** Оптимальный размер этих элементов — 2-3 десятка строк. Если получается больше, разделите функцию на несколько маленьких и добавьте одну общую. Пусть маленькие выполняют по одной операции, а общая функция их вызывает.
5. **Слишком длинные классы.** То же самое. Оптимальная длина класса — 20–30 строк. Разбейте длинный класс на несколько маленьких и включите их объекты в один общий класс.
6. **Слишком длинный список параметров функции или метода**. Они только запутывают, а не помогают. Если все эти параметры действительно нужны, вынесите их в отдельную структуру или класс с понятным именем, а в функцию передайте ссылку на него.
7. **Много комментариев**. Плохой код часто прикрывается обильными комментариями. Если почувствовали желание пояснить какой-то участок кода, попробуйте сначала его переписать, чтобы и так стал понятным. Бесполезные комментарии загромождают программу, а устаревшие и неактуальные вводят в заблуждение.

Несмотря на всю пользу рефакторинга, применять его стоит не всегда. К примеру, если вы разрабатываете маленький и несложный продукт, развитие которого идет очень медленно. В таком случае править код может и не понадобиться. Хотя, конечно, лучше соблюдать порядок в любых проектах. Также от рефакторингов можно отказаться при приближении дедлайна. Эффективность этой процедуры чувствуется лишь со временем, поэтому, если вы начнете рефакторить код перед самой сдачей проекта, заметного роста производительности не произойдет, а значит, вы впустую потратите время. Еще один пример — ситуация, когда проще переписать код с нуля, чем разбираться в нем. Рефакторинг — не панацея. Иногда распутывать безнадежный клубок из программных строк слишком долго и затратно.

## 

## 15. Рефакторинг кода на Java и Python.

Рефакторинг — это улучшения кода приложения (или архитектуры) за счет внесения внутренних изменений, без изменения внешнего функционала. Этими улучшениями могут быть увеличение стабильности, производительности или уменьшение сложности.

Python:

1. Включение списков, словарей и множеств

Есть три типичных изменяемых контейнерных типа данных в Python: списки, словари и множества. Если у нас есть итерируемый объект, то мы можем воспользоваться циклом for, чтобы перебрать этот объект для создания на его основе нового списка. Однако характерный для Python способ – использовать включение списка.

# Вместо этого:

squares0 = []

**for** number **in** numbers:

**if** number%2 == 0:

squares0.append(number\*number)

# Характерный для Python способ:

squares1 = [x\*x **for** x **in** numbers **if** x%2 == 0]

1. F-строки

Строки – привычный примитивный тип данных, который мы используем почти во всех проектах. В большинстве случаев для отображения строковых данных нам нужно сделать дополнительный шаг — определенным образом отформатировать их. Можно выполнить форматирование в стиле языка C, что подразумевает использование символа %, либо воспользоваться методом format, которым обладают строки в Python.

Однако в одном из недавних релизов Python был представлен новый метод форматирования строк. Он известен как «f-строки», что означает «форматируемые строковые литералы» – лаконичный и читабельный способ форматирования строк.

# Вместо этого:

print('1 USD = {rate:.2f} EUR'.format(rate=usd\_to\_eur))

print('1 USD = %.2f EUR' % usd\_to\_eur)

# Характерный для Python способ:

print(f'1 USD = {usd\_to\_eur:.2f} EUR')

1. Множественное присваивание и распаковка кортежей

При работе с переменными определение одной переменной на одной строке – устоявшаяся практика. Однако при объявлении нескольких переменных мы можем сделать это на одной строке.

# Вместо этого:

code = 404

message = "Not Found"

# Характерный для Python способ:

code, message = 404, "Not Found"

1. Выражение присваивания

В выражении присваивания используется :=, Как понятно из его имени, выражение присваивания позволяет присваивать значение переменной и, в то же время, оно может быть использовано в качестве выражения, например таком, как условный оператор.

# Вместо этого:

account\_number = get\_account("123-45-6789")

**if** account\_number:

withdraw\_money(account\_number)

**else**:

found\_no\_account()

# Характерный для Python способ:

**if** account\_number := get\_account("123-45-6789"):

withdraw\_money(account\_number)

**else**:

found\_no\_account()

1. Итерирование с использованием функции enumerate

Почти в каждом проекте нам неминуемо приходится заставлять нашу программу повторять определенные операции для всех элементов в списке, кортеже или каком-либо другом контейнерном типе данных. Добиться выполнения этих повторяющихся операций можно с помощью цикла for. Однако, при таком итерировании, если нам нужен подсчет текущей итерации цикла, лучше воспользоваться функцией enumerate.

# Создаем список студентов для итерирования

students = ['John', 'Jack', 'Jennifer', 'June']

# Вместо этого:

**for** i **in** range(len(students)):

student = students[i]

print(f"# {i+1}: {student}")

# 1: John

# 2: Jack

# 3: Jennifer

# 4: June

>>>

# Характерный для Python способ:

**for** i, student **in** enumerate(students, 1):

print(f"# {i}: {student}")

Java:

1. Неоправданно большие элементы

Существуют громоздкие классы и методы, с которыми невозможно эффективно работать именно из-за их огромного размера.

* Большой класс

У такого класса есть огромное количество строк кода и много различных методов. Как правило, функционал такого класса перегружен. В этом случае помогает выделение части функционала в отдельный класс.

* Большой метод

Это возникает, когда разработчик добавляет в метод новый функционал. Есть два правила рефакторинга большого метода:

Если при написании метода хочется добавить комментарий в код, необходимо

1. Выделить этот функционал в отдельный метод;
2. Если метод занимает более 10-15 строк кода, следует определить задачи и подзадачи, которые он выполняет, и попробовать вынести подзадачи в отдельный метод.

Несколько способов устранить большой метод:

1. Выделить часть функционала метода в отдельный метод;
2. Если локальные переменные не дают вынести часть функционала, можно передать весь объект в другой метод.

2. Использование множества примитивных типов данных

Обычно такая проблема возникает, когда с течением времени в классе растет количество полей для хранения данных. Например, если использовать примитивные типы вместо маленьких объектов для хранения данных (валюта, дата, телефонные номера и т.д.) или константы для кодирования какой-либо информации. Хорошей практикой в этом случае будет логическая группировка полей и вынос в отдельный класс. Также в класс можно включить методы для обработки этих данных.

3. Длинный список параметров

Достаточно распространенная ошибка, особенно в совокупности с большим методом. Обычно она возникает, если функционал метода перегружен, или метод объединяет несколько алгоритмов в себе. В длинных списках параметров очень трудно разбираться, и использовать такие методы неудобно. Поэтому лучше передать объект целиком. Если у объекта нет достаточно данных, стоит использовать более общий объект или разделить функционал метода, чтобы он обрабатывал логически связанные данные.

4. Группы данных

Часто в коде появляются логически связанные группы данных. Например, параметры подключения в БД (URL, имя пользователя, пароль, имя схемы и тд). Если из перечня элементов нельзя удалить ни одно поле, значит перечень — это группа данных, которую необходимо вынести в отдельный класс.

## 

## 16. Автоматизированный рефакторинг в Java IDE.

Некоторые способы рефакторинга для Java-файлов в IDEA:

«Refactor This» (Ctrl+Alt+Shift+T): данный способ используется для быстрого доступа к списку доступных способов рефакторинга. Заметьте, список зависит от места, где вы его вызываете.

«Rename» (Shift+F6): позволяет переименовать практически любой идентификатор в коде, будь то переменная или названия класса. Изменения распространяются по всему проекту, в некоторых случаях, включая и комментарии.

«Rename File»: переименовывает файл и ссылки на этот файл. В принципе можно вызывать через Shift+F6 если выделен файл.

«Change Signature» (Ctrl+F6): изменяет сигнатуру метода и вызывает обновление.

«Type Migration» (Ctrl + Shift + F6): позволяет изменить тип переменной, включая сигнатуры методов, возвращаемый тип переменной.

«Make Static» (Ctrl + Shift + F6): позволяет сконвертировать метод или внутренний класс в статический.

«Convert To Instance Method»: позволяет сконвертировать статический метод в нестатический (противоположность ”Make Static”). При этом можно указать к какому классу будет относиться новый метод.

«Move Classes» (F6): перемещает классы в проекте.

«Copy Classes» (F5): копирует Java-файл в указанный пакет.

«Variable» (Ctrl+Alt+V): создает новую переменную из выделенного фрагмента.

«Constant» (Ctrl+Alt+C): создает новую константу из выделенного фрагмента.

«Field» (Ctrl+Alt+F): cоздает новое поле класса из выделенного фрагмента.

«Parameter» (Ctrl+Alt+P): cоздает новый параметр (функции) из выделенного фрагмента.

«Parameter Object»: удобный способ, когда в функцию передается много аргументов и вам надо обернуть их в класс.

«Method» (Ctrl+Alt+M): извлекаем метод из выделенного фрагмента.

«Replace Method With Method Object»: оборачивает выделенный фрагмент в объект. Может использоваться, если надо вернуть несколько объектов из метода (возвращает объект-обертку).

«Delegate»: позволяет извлечь методы и поля в отдельный класс.

«Interface»: для заданного класса и его методов создает интерфейс.

«Superclass»: аналогично пункту «Interface», только теперь создается класс-родитель (Superclass).

«Find and Replace code duplicate»: ищет похожие фрагменты кода и предлагает заменить их, например, вызовом метода или константой.

«Invert Boolean»: позволяет инвертировать булевые переменные.

«Pull Member Up»: позволяет перемещать методы или поля по иерархии вверх.

«Pull Member Down»: выполняет обратную задачу пункта «Pull Member Up». Позволяет перемещать методы или поля по иерархии вниз.

«Use Interface Where Possible»: IDEA старается заменить, где это возможно, указания классов на указание интерфейсов.

## 

## 17. Система управления базами данных Redis.

Redis (Remote Dictionary Server) это опен-сорс система управления базами данных, которая использует оперативную память для хранения данных и предоставляет быстрое выполнение операций с данными. Она поддерживает несколько типов данных, включая строки, списки, карты, множества и отсортированные множества.

Используется в основном для:

1. Использование в качестве кеша: для этого у ключей есть параметр времени ttl и встроенный счетчик секунд для каждого ключа. Когда время заканчивается, то ключ автоматически удаляется.
2. Использование в качестве хранилища для очереди или стека задач для их асинхронного выполнения относительно основного кода
3. В качестве основной/дополнительной документоориентированной БД, подобно MongoDB: для этого включается синхронизация данных, хранящихся Redis с локальным диском (т.е. данные хранятся не только в оперативной памяти), но тогда производительность падает (и вообще это нетрадиционное использование Redis, я такое не приветствую)

Для работы с Redis из Python можно использовать клиентскую библиотеку redis-py. Ниже приведен пример работы с Redis используя redis-py:

import redis  
r = redis.Redis(host='localhost', port=6379, db=0)  
r.set('key', 'value')  
print(r.get('key'))

В Redis есть множество других операций, которые можно выполнять с помощью redis-py, например:

* Удаление ключа с помощью метода delete()
* Инкремент и декремент числового значения ключа с помощью методов incr() и decr()
* Работа со списками, множествами и отсортированными множествами с помощью методов rpush(), lpop(), sadd(), smembers() и т.д.
* Использование команд Lua скриптов с помощью метода eval()
* Выполнение транзакций с помощью метода pipeline()

Важно иметь в виду, что Redis хранит данные в оперативной памяти, поэтому его использование не подходит для хранения большого количества данных, которые не помещаются в память.

Redis предоставляет несколько политик вытеснения ключей, которые могут быть использованы для управления использованием памяти и ограничения размера базы данных. Эти политики могут быть установлены для всей базы данных или для отдельных ключей.

* volatile-lru: Эта политика вытесняет ключи, которые не использовались за определенное время (заданное с помощью параметра maxmemory-policy), из самой менее используемой группы ключей.
* volatile-ttl: Эта политика вытесняет ключи, у которых истек срок действия (TTL) из самой менее используемой группы ключей.
* volatile-random: Эта политика выбирает случайно ключ из самой менее используемой группы ключей для вытеснения.
* allkeys-lru: Эта политика вытесняет ключи из самой менее используемой группы ключей

## 

## 18. Платформа API Postman.

Postman - это платформа для тестирования и документирования API. Она позволяет пользователям легко создавать, отправлять и анализировать HTTP-запросы. Основные действия, которые можно выполнять с помощью Postman:

1. Создание и отправка HTTP-запросов (GET, POST, PUT, DELETE и т.д.
2. Добавление и редактирование заголовков запросов
3. Добавление и редактирование параметров запроса
4. Сохранение и повторное использование запросов
5. Просмотр и анализ результатов запроса (включая заголовки и тело ответа)
6. Автоматизация тестирования с помощью сценариев
7. Документирование API.

## 

## 19. Программное обеспечение для автоматизации развёртывания и управления приложениями Docker.

Docker — это платформа контейнеризации с открытым исходным кодом, с помощью которой можно автоматизировать создание приложений, их доставку и управление. Платформа позволяет быстрее тестировать и выкладывать приложения, запускать на одной машине требуемое количество контейнеров.

**Основные компоненты Docker**

Dockerfile. Текстовый файл с последовательно расположенными инструкциями для создания образа Docker..

Daemon. Фоновая служба на хосте, которая отвечает за создание, запуск и уничтожение контейнеров.

Image. Неизменяемый файл (образ), из которого можно неограниченное количество раз развернуть контейнер.

Client. Утилита командной строки в Docker для управления демоном. Любое взаимодействие с контейнером проходит через Daemon.

Container. Запущенное приложение, которое развернули из образа.

Registry. Служба в Docker, выполняющая функции репозитория.

Docker Hub. Популярный публичный репозиторий, используемый по умолчанию в Docker. Обеспечивает интеграцию с GitHub и BitBucket.

Docker Desktop. Приложение, позволяющее локально собирать, выполнять и тестировать контейнеры. Работает на Windows и macOS.

Docker volumes. Тома для постоянного хранения информации.

**Как устроен образ Docker**

Базовый образ — главный элемент контейнеризации в Docker. В нем содержатся процессы и зависимости, необходимые для нормальной работы приложения.На базовый образ в Docker один за другим накладываются доступные только для чтения слои, которые образуются после любых изменений в образе. Каждый новый слой — это актуальная версия образа. Получается, что финальный образ — это объединение всех слоев в один. Если образ — это набор доступных только для чтения слоев, то контейнер представляет собой тот же образ, но с еще одним слоем сверху — с возможностью записи. Информация записывается в контейнер, а когда он уничтожается, верхний слой и содержащиеся в нем данные пропадают. В случае необходимости создается новый (чистый) контейнер из старого образа.

## 

## 20. Инструменты оптимизации работы высоконагруженных систем.

Высоконагруженными (Highload) называют системы безостановочного доступа, то есть те структуры, запрос данных из которых позволяет получать информацию без длительной задержки при беспрерывной работе. Иными словами, высоконагруженные системы – это, в основном, те же веб-сайты (либо веб-приложения), только с очень большим количеством пользователей и как следствие с большой нагрузкой, требующей оптимизированной серверной части веб-сайта (либо веб-приложения).

Причины:

1. большое количество одновременных пользователей;
2. большой объем обрабатываемых данных;
3. наличие многочисленных сложных расчетов и вычислений.

Отличие высоконагруженных систем от обычных:

1. Жесткость. Высоконагруженное бизнес-приложение – это жесткая система, в которой предусмотрены варианты изменения лишь некоторых частей.
2. Быстрое время отклика. Это важное качество highload-приложений. Общение пользователя с системой происходит через запрос, и ответ на него должен приходить если не мгновенно, то через приемлемое время. Вряд ли вам понравится работать с программой, которая сутки будет производить необходимое вычисление.
3. Масштабируемость. Как много: людей могут одновременно пользоваться ресурсом; данных способна вместить база до того, как перестанет справляться с нагрузкой? Высоконагруженный проект обязательно должен быть масштабируемым.

Два основных метода оптимизации – статистический и алгебраический

Инструменты:

MapReduce — это подход к обработке данных, который имеет два серьезных преимущества по сравнению с традиционными решениями. Первое и самое главное преимущество — это производительность (он распараллелен, что позволяет обрабатывать огромные массивы данных на множестве ядер/процессоров/машин). Второе - возможность описывать обработку данных нормальным кодом.

Масштабируемые кластерные решения, как решение проблем реляционных СУБД (усложнение и перегрузка). Однако стоимость внедрения и сопровождения подобных решений зачастую не окупается.

NoSQL: Исключение излишнего усложнения; высокая пропускная способность; неограниченное горизонтальное масштабирование; консистентность в жертву производительности.