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

Управление процессом разработки программного обеспечения (ПО) направлено на достижение определенных целей и решение различных задач. Основные цели управления разработкой ПО включают в себя:

1. Соблюдение сроков: Обеспечение выполнения проекта в установленные сроки, что позволяет своевременно предоставить продукт заказчику или внедрить его в предприятии.
2. Соблюдение бюджета: Эффективное распределение ресурсов и финансов для минимизации затрат и максимизации прибыли.
3. Высокое качество продукта: Обеспечение соответствия разрабатываемого ПО требованиям заказчика, стандартам качества и безопасности.
4. Эффективное управление рисками: Минимизация возможных проблем и рисков, связанных с техническими, организационными или другими аспектами процесса разработки.
5. Контроль изменений: Управление изменениями в проекте, чтобы избежать негативных последствий и обеспечить согласованность с требованиями заказчика.
6. Максимизация участия заинтересованных сторон: Взаимодействие с заказчиком, конечными пользователями и другими заинтересованными сторонами для максимального понимания их потребностей и ожиданий.

Задачи управления процессом разработки ПО могут включать в себя:

1. Планирование проекта: Определение целей, задач, ресурсов и сроков для эффективного выполнения проекта.
2. Оценка рисков: Идентификация и анализ возможных проблем и рисков, а также разработка стратегий их управления.
3. Контроль качества: Организация процессов тестирования и обеспечение соответствия продукта установленным стандартам.
4. Управление командой: Координация работы команды разработчиков, обеспечение их мотивации и эффективного взаимодействия.
5. Мониторинг и отчетность: Отслеживание хода выполнения проекта, анализ результатов и предоставление отчетов заинтересованным сторонам.
6. Управление изменениями: Контроль над внесением изменений в проект, их анализ и влияние на сроки и бюджет.

Существует множество инструментов для разработки программного обеспечения, и выбор зависит от типа проекта, используемых технологий и предпочтений команды разработчиков. Вот несколько ключевых категорий инструментов для разработки ПО:

1. **Интегрированные среды разработки (IDE):**
   * **IntelliJ IDEA (JetBrains):** IDE для Java с поддержкой множества фреймворков и языков.
   * **PyCharm (JetBrains):** IDE для Python-разработки.
   * **Visual Studio Code (Microsoft):** Легковесный редактор кода с обширной экосистемой расширений.
2. **Системы управления версиями:**
   * **Git:** Распределенная система управления версиями.
3. **Баг-трекинг и управление проектами:**
   * **Jira (Atlassian):** Инструмент управления проектами и баг-трекинг.
4. **Тестирование и QA:**
   * **JUnit (Java) и pytest (Python):** Фреймворки для модульного тестирования.
5. **Документация и коммуникация:**
   * **Confluence (Atlassian):** Платформа для создания и совместного использования документации.
6. **Инструменты для CI/CD:**
   * **Docker:** Платформа для контейнеризации приложений.
7. **Жизненный цикл программных систем.**

Жизненный цикл программной системы (ЖЦ ПС) представляет собой концептуальную модель, описывающую все этапы жизни программного продукта от его концепции и создания до вывода из эксплуатации. Этот процесс включает в себя различные фазы, задачи и процессы, необходимые для успешного разработки, внедрения, поддержки и, в конечном итоге, устаревания программного продукта.

Жизненный цикл ИС определяется как период времени, который начинается с момента принятия решения о необходимости ИС и заканчивается в момент ее полного изъятия из эксплуатации.

Этапы жизненного цикла программных систем:

1. Инициация (Conception): Этот этап связан с формулированием идеи создания программной системы, выявлением потребностей и определением основных целей и задач продукта.
2. Планирование (Planning): На этой стадии определяются ресурсы, сроки и бюджет проекта. Разрабатывается план работы, который включает в себя распределение задач, определение рисков и установку метрик успеха.
3. Разработка (Development): Фаза, в которой осуществляется проектирование и создание кода программного продукта. Включает в себя создание архитектуры, написание кода, тестирование и документирование.
4. Тестирование (Testing): Этап, на котором проверяется работоспособность и соответствие программы заявленным требованиям. Включает юнит-тестирование, интеграционное тестирование, системное тестирование и при необходимости приемочное тестирование.
5. Внедрение (Deployment): Процесс развертывания программной системы на реальном оборудовании или в рабочей среде.
6. Эксплуатация и сопровождение (Operation and Maintenance): Фаза, в которой происходит реальное использование программного продукта. Включает в себя поддержку, решение проблем, обновления и улучшения.
7. Устаревание (Decommissioning): Этап, на котором программная система выводится из эксплуатации. Включает в себя архивацию данных, уведомление пользователей и оценку эффективности системы.

Виды жизненных циклов:

1. Классический (Waterfall): Линейная модель, где каждая фаза выполняется последовательно.
2. Итеративный и инкрементальный (Iterative and Incremental): Разработка происходит частями (итерациями), с возможностью внесения изменений в ходе процесса.
3. Спиральный (Spiral): Процесс разработки, комбинирующий элементы итераций и инкрементов с анализом рисков на каждой фазе.
4. Гибкий (Agile): Гибкий подход, ориентированный на частые изменения требований и активное взаимодействие с заказчиком.
5. **Обзор информационных систем.**

Информационная система (ИС) — это комплекс взаимосвязанных компонентов, совместно использующих ресурсы для сбора, обработки, хранения и предоставления информации с целью поддержки принятия решений в организации или достижения определенных целей. Классификация информационных систем:

1. **Управление бизнес-процессами (BPM):** концепция процессного управления организацией, рассматривающая бизнес-процессы как особые ресурсы предприятия, непрерывно адаптируемые к постоянным изменениям.

Конструктор для моделирования процессов. Средства отладки. Инструменты для разработки приложений. Инструменты для конфигурирования самой BPM-системы. Пользовательский интерфейс процессов и задач. Системы мониторинга и сервисы уведомлений. Модули и шлюзы для интеграции BPM-системы с другими сервисами Примеры: Creatio. ELMA. Bitrix24.

1. **Системы управления ресурсами предприятия (Enterprise Resource Planning - ERP):**

Концепция ERP сформулирована в 1990 году аналитиком Gartner как видение развития методик MRP II. ERP – это создание модели управления всеми потоками; установка и поддержание в рабочем состоянии оборудования для хранения; подключение правильного ПО; обеспечение полноценного IT-отдела; обучение пользователей всему необходимому. Модульный принцип организации позволяет внедрять ERP-системы поэтапно, последовательно переводя в эксплуатацию один или несколько функциональных модулей. Примеры: 1С:ERP, Галактика ERP, ERP решения Парус

1. **САПР:** системы инженерной графики (CAD), системы инженерных расчетов (CAE), системы автоматизации подготовки и управления производства (CAM)

В современные CAD-системы входят модули моделирования трехмерной объемной конструкции и оформления чертежей и текстовой конструкторской документации (спецификаций, ведомостей и т. д.). САЕ-системы — это класс систем, каждая из которых позволяет решать определенную расчетную задачу, начиная от расчетов на прочность, анализа и моделирования тепловых процессов до расчетов гидравлических систем и машин, расчетов процессов литья. CAM-системы предназначены для проектирования обработки изделий на станках с числовым программным управлением (ЧПУ) и выдачи программ для этих станков. В настоящее время CAM-системы являются одним из основных способов изготовления сложнопрофильных деталей и сокращения цикла их производства. Примеры: Autodesk, NX, Creo.

1. **CRM-система**

CRM-система собирает данные о каждом клиенте в одной панели управления. С помощью этой информации ваша команда может отслеживать путь покупателя и делать релевантные предложения на каждом этапе. CRM помогает определить интересы и предпочтения ваших клиентов. Консолидация данных клиентов. Отслеживание взаимодействий и активности. Измерение производительности и продуктивности. Автоматизация рутинных процессов. Пример: Битрикс24, Microsoft Dynamics, SAP CRM

1. **EAM (Enterprise Asset Management) - Системы управления активами предприятия**

техническое обслуживание и ремонт; материально-техническое снабжение; управление складскими запасами (запчасти для технического обслуживания); управление финансами, качеством и трудовыми ресурсами в части технического обслуживания, ремонтов и материально-технического обеспечения. Автоматизированное управление процессом эксплуатации и тех. обслуживания. Лучше предупреждать, чем ликвидировать последствия. Примеры: EAMcloud, 1С:ТОИР, IBS EAM

1. **СЭД/EDMS**

Система EDMS разрабатывалась для электронного архивирования и управления динамическими документами, представляет собой платформу для хранения и обмена документацией, сотрудничества, благодаря которой можно осуществлять управление миллионами документов компании посредством единой системы, а пользователи по всему миру могут быстро и легко получить к ним доступ, а все операции, производимые с документами на протяжении всего жизненного цикла, записывается и контролируются.

Примеры: Watermark, PC DOCS, Open Text, iManage, Tower Technology, IBM и Identitech

1. **Системы управления производством MES:** Системы управления производством – ПО, специально разработанное для повышения эффективности планирования и управления производством. MES система «1С: Производственная логистика». **Примеры:** 1С:MES, Easy95 ODS, ФОБОС
2. **Баг-трекинг системы.**

Баг-трекинг системы, также известные как системы управления ошибками или дефект-трекеры, представляют собой инструменты, используемые в процессе разработки программного обеспечения для отслеживания, управления и регистрации дефектов, ошибок или недочетов (багов) в программном коде или функциональности продукта. Они играют важную роль в обеспечении качества и поддержке жизненного цикла программного продукта. Основные функции баг-трекинг систем:

1. Регистрация багов: Возможность зарегистрировать новый баг в системе, предоставив подробную информацию о проблеме, включая шаги воспроизведения, ожидаемое поведение и фактическое поведение.
2. Отслеживание статуса: Предоставление информации о текущем статусе каждого зарегистрированного бага, например, "новый", "в обработке", "исправлен", "проверен" и т.д.
3. Приоритизация: Возможность устанавливать приоритеты для различных багов в соответствии с их важностью и воздействием на функциональность продукта.
4. Назначение ответственных: Присвоение ответственных сотрудников или разработчиков для решения конкретных багов.
5. Комментарии и обсуждение: Возможность добавления комментариев и обсуждения к каждому багу, что обеспечивает прозрачность и обмен информацией между членами команды.
6. История изменений: Фиксация всех изменений, связанных с багом, таких как статус, ответственные, комментарии, даты изменений и т.д.
7. Отчетность и аналитика: Генерация отчетов о статусе багов, динамике их решения, времени на решение и других параметрах для анализа эффективности процесса управления багами.

Примеры популярных баг-трекинг систем:

1. Jira: Разработана компанией Atlassian, предоставляет широкие возможности для управления проектами, включая баг-трекинг.
2. Bugzilla: Бесплатная и открытая система, разработанная Фондом Свободного Программного Обеспечения (Free Software Foundation).
3. Redmine: Открытая платформа управления проектами, которая включает в себя функциональность баг-трекинга.
4. Trello: Легкая в использовании система, предоставляющая возможности для организации задач, включая трекинг багов.

Преимущества использования баг-трекинг систем:

* Улучшение коммуникации и совместной работы в команде разработки.
* Ускорение процесса обнаружения, регистрации и решения проблем.
* Повышение прозрачности и контроля над процессом управления качеством.

Недостатки:

* Внедрение и обучение персонала может потребовать времени и ресурсов.
* Неэффективность при неправильной настройке и непродуманном использовании.

1. **Системы управления версиями (обзор систем).**

Системы управления версиями (СУВ), также известные как системы контроля версий, представляют собой инструменты, используемые разработчиками программного обеспечения для отслеживания изменений в исходном коде, документации и других артефактах проекта. Эти системы записывают и хранят изменения, позволяя команде разработчиков восстанавливать предыдущие версии кода, сравнивать изменения, а также эффективно совместно работать над проектом. **Основные типы СУВ:**

1. **Локальные СУВ (Local VCS):** Они хранят изменения в файловой системе на локальном компьютере. Пример: RCS (Revision Control System).
2. **Централизованные СУВ (Centralized VCS):** Все файлы и история изменений хранятся на центральном сервере, доступном всем разработчикам. Пример: SVN (Apache Subversion).
3. **Распределённые СУВ (Distributed VCS):** Каждый разработчик имеет полную копию репозитория, что обеспечивает возможность работы офлайн и улучшенную масштабируемость. Примеры: Git, Mercurial.

**Популярные СУВ:**

1. **Git:** **Преимущества:**
   * + Высокая скорость и эффективность.
     + Распределённая архитектура.
     + Большое сообщество и поддержка.
   * **Используется в:** Множество открытых и коммерческих проектов, включая Linux, Android, и многие другие.
2. **Subversion (SVN):** **Преимущества:**
   * + Простота в освоении.
     + Централизованная архитектура.
   * **Используется в:** Многие коммерческие проекты, включая Apache Software Foundation.
3. **Mercurial:** **Преимущества:**
   * + Простота использования.
     + Распределённая архитектура.
   * **Используется в:** Mozilla, Facebook, среди других.
4. **Perforce (Helix Core):** **Преимущества:**
   * + Высокая производительность с большими проектами.
     + Централизованная и распределённая архитектуры.
   * **Используется в:** Крупные корпорации, такие как Salesforce, Electronic Arts.

**Основные функции СУВ:**

1. **Запись изменений:** Фиксация изменений в исходном коде или других файлах.
2. **Откат к предыдущей версии:** Возможность вернуться к предыдущим состояниям проекта.
3. **Ветвление и слияние (Branching and Merging):** Создание веток для независимой разработки и их последующее объединение.
4. **Отслеживание изменений (Diff):** Возможность просмотра различий между версиями файлов.
5. **История изменений:** Ведение детальной истории изменений с указанием авторов и времени.
6. **Коллаборация:** Возможность совместной работы нескольких разработчиков над одним проектом.

**Преимущества использования СУВ:** 1) **Отслеживание изменений:** Легкость отслеживания истории изменений в коде. 2) **Коллаборация:** Улучшенная командная работа, возможность параллельной разработки. 3) **Откат к предыдущим версиям:** Возможность быстро вернуться к стабильной версии проекта.

**Недостатки использования СУВ:** 1) **Изучение:** Некоторые СУВ могут потребовать времени для освоения. 2) **Размер репозитория:** С ростом проекта размер репозитория может значительно увеличиваться.

1. **Инструменты для мобильной разработки (обзор систем).**

Для мобильной разработки существует множество инструментов и платформ, позволяющих разработчикам создавать приложения под различные операционные системы, такие как Android и iOS. Выбор инструментов зависит от множества факторов, таких как тип приложения, целевая платформа, опыт разработчика и требования проекта. Некоторые разработчики предпочитают использовать кроссплатформенные фреймворки, чтобы избежать дублирования работы для разных платформ, в то время как другие предпочитают нативные инструменты для максимальной производительности и доступа к уникальным функциям устройств. Перечислим несколько популярных инструментов:

1. Android Studio:

Язык программирования: Java, Kotlin.

Официальная среда разработки для Android. Android Studio предоставляет все необходимые инструменты для создания, тестирования и отладки приложений под Android. Поддерживает как Java, так и Kotlin.

1. Xcode:

Язык программирования: Swift, Objective-C.

Официальная среда разработки от Apple для создания приложений под iOS, macOS и watchOS. Включает в себя множество инструментов для проектирования интерфейса, отладки и профилирования.

1. Flutter:

Язык программирования: Dart.

Фреймворк от Google для создания кроссплатформенных мобильных приложений. Использует один код для разработки приложений под Android и iOS.

1. React Native:

Язык программирования: JavaScript, React.

Фреймворк от Facebook для создания кроссплатформенных мобильных приложений с использованием JavaScript и React. Позволяет использовать общий код для обеих платформ.

1. **Система управления версиями и поиска ошибок Redmine.**

**1.** Redmine — это веб-приложение с открытым исходным кодом, предоставляющее интегрированные инструменты для управления проектами, включая систему управления версиями (СУВ) и баг-трекинг. Это мощный инструмент для организации работы команды разработчиков и улучшения процессов разработки программного обеспечения.

**2. Основные функции:**

**Система управления версиями:**

* Создание репозиториев: Redmine интегрируется с различными СУВ, такими как Git, Mercurial, Subversion.
* Отслеживание изменений: Пользователи могут просматривать историю изменений в коде, смотреть различия между версиями и следить за коммитами.
* Ветвление и слияние: Поддерживает ветвление кода и последующее слияние изменений.

**Система управления ошибками (Баг-трекинг):**

* Регистрация багов: Возможность создавать задачи по обнаруженным ошибкам, указывая детали, прикреплять файлы и устанавливать приоритеты.
* Отслеживание статуса: Предоставление информации о текущем статусе каждого бага - "новый", "в обработке", "решен", "закрыт" и др.
* Аналитика и отчетность: Встроенные отчеты и статистика по багам для анализа эффективности процесса управления ошибками.

**3. Преимущества:**

* **Интеграция с СУВ:** Возможность интегрировать Redmine с различными системами управления версиями, что обеспечивает гибкость в выборе инструментов разработки.
* **Пользовательский интерфейс:** Интуитивно понятный интерфейс, обеспечивающий легкость использования для разработчиков и других членов команды.
* **Расширяемость:** Возможность добавления плагинов и модулей для расширения функциональности в соответствии с требованиями проекта.
* Множественные роли в проектах, вложенные проекты неограниченной глубины, пакетное редактирование задач, просмотр вложений.

**4. Недостатки:**

* **Сложность настройки:** Внедрение и настройка Redmine может потребовать определенных технических навыков.
* **Система не всегда актуальна:** Некоторые пользователи отмечают, что не все функции всегда обновляются и могут быть не так актуальными, как в других системах.
* Нет прав на отдельные типы переходов Workflow

**5. Пример использования:** Предположим, команда разработчиков использует Redmine для управления проектом. Они могут создавать задачи, привязанные к конкретным версиям проекта, отслеживать изменения в коде через интеграцию с СУВ, а также регистрировать и отслеживать баги встроенной системой баг-трекинга.

**6. Резюме:** Redmine предоставляет разработчикам и командам удобный инструмент для управления проектами, включая систему управления версиями и баг-трекинг. Его гибкость и расширяемость делают его популярным выбором в различных командах разработки программного обеспечения.

1. **Системы управления версиями Git.**

**Система контроля версий** – 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. **Система управления версий**– это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определенной версии. Git — это распределенная система управления версиями (VCS), которая широко используется разработчиками для отслеживания изменений в исходном коде проекта.

Ключевые особенности Git:

* Поддерживается автономная работа; локальные фиксации изменений могут быть отправлены позже.
* Каждое рабочее дерево в Git содержит хранилище с полной историей проекта.
* Ни одно хранилище Git не является по своей природе более важным, чем любое другое.
* Скорость работы, ветвление делается быстро и легко.

Основные понятия и команды Git:

* + 1. Репозиторий:
* *Локальный репозиторий* находится на компьютере и содержит полную историю проекта.
* *Удаленный репозиторий* расположен на внешнем сервере (например, GitHub, GitLab, Bitbucket) и используется для совместной работы и обмена кодом.
  + 1. Коммит:

Коммит фиксирует изменения в коде.

git commit -m "Ваше сообщение" - создает коммит с указанным сообщением.

* + 1. Ветка:

Ветка — это отдельная линия разработки.

git branch - показывает список веток.

git branch <название> - создает новую ветку.

git checkout <название> - переключается на указанную ветку.

git merge <название> - объединяет ветку в текущую.

* + 1. Слияние:

Объединение изменений из одной ветки в другую.

git merge <название ветки> - объединяет указанную ветку в текущую.

* + 1. Конфликты слияния:

Возникают, когда Git не может автоматически объединить изменения. Решаются вручную, после чего происходит продолжение слияния.

* + 1. Pull Request:

Используется для предложения изменений в удаленный репозиторий. Обычно сопровождается обсуждением и проверкой кода.

* + 1. Клонирование:

Создание копии удаленного репозитория на локальной машине.

git clone <URL репозитория> - клонирует репозиторий.

* + 1. Статус:

git status - показывает состояние рабочей директории и индекса.

* + 1. Откат и сброс:

git revert <хеш коммита> - создает новый коммит, отменяющий изменения указанного коммита.

git reset <опции> - изменяет историю коммитов (осторожно, используйте с осторожностью).

1. **Интегрированные среды разработки (обзор систем).**

**Интегрированные среды разработки (IDE, integrated development environment)** – комплекс средств, поддерживающих полный жизненный цикл программного продукта (ПП).

**Простая ИСР** содержит минимальный набор компонентов:

* Текстовый редактор
* Компилятор
* Редактор связей
* Отладчик

**Продвинутая ИСР:**

* Автоматизация редактирования кода
* Выделение синтаксиса
* Интеллектуальное завершение кода
* Поддержка рефакторинга
* Автоматизация локальной сборки
* Компиляция
* Тестирование
* Отладка

**Интегрированные среды разработки:**

1. **Microsoft Visual Studio** – интегрированная среда разработки является творческой стартовой площадкой, которую можно использовать для редактирования, отладки и сборки кода, а также для публикации приложения.

Поддерживаемые языки: Ajax, ASP.NET, DHTML, ASP.NET, JavaScript, Visual Basic, Visual C#, Visual C++, Visual F#, XAML и другие.

Преимущества:

* Огромная коллекция всевозможных расширений, которая постоянно пополняется.
* Технология автодополнения IntelliSense.
* Возможность кастомизировать рабочую панель.
* Поддержка разделенного экрана (split screen).

Недостатки:

* тяжеловесность IDE. Для выполнения даже небольших правок могут потребоваться значительные ресурсы, поэтому если нужно выполнить какую-то простую и быструю задачу, удобнее использовать более легкий редактор.

1. **PyCharm** – среда разработки для Python и Django с умным автодополнением кода, проверкой ошибок на лету, быстрыми исправлениями и другими возможностями.

Поддерживаемые языки: Python, Jython, Cython, IronPython, PyPy, AngularJS, Coffee Script, HTML/CSS, Django/Jinja2 templates, Gql, LESS/SASS/SCSS/HAML, Mako, Puppet, RegExp, Rest, SQL, XML, YAML и т.д.

Преимущества:

* Поддержка Google App Engine; IronPython, Jython, Cython, PyPy wxPython, PyQt, PyGTK и др.
* Поддержка Flask-фреймворка и языков Mako и Jinja2.
* Редактор Javascript, Coffescript, HTML/CSS, SASS, LESS, HAML.
* Интеграция с системами контроля версий (VCS).
* UML диаграммы классов, диаграммы моделей Django и Google App Engine.

Недостатки:

* Иногда встречаются баги, которые, как правило, не вызывают сильных неудобств.

1. **IntelliJ IDEA** – ведущая IDE для разработки на Java и Kotlin.

Поддерживаемые языки: Java, AngularJS, Scala, Groovy, AspectJ, CoffeeScript, HTML, Kotlin, JavaScript, LESS, Node JS, PHP, Python, Ruby, Sass,TypeScript, SQL и другие.

Преимущества:

* Инструменты для анализа качества кода, удобная навигация, расширенные рефакторинги и форматирование для Java, Groovy, Scala, HTML, CSS, JavaScript, CoffeeScript, ActionScript, LESS, XML
* Интеграция с серверами приложений, включая Tomcat, TomEE, GlassFish, JBoss, WebLogic, WebSphere, Geronimo, Resin, Jetty и Virgo.
* Инструменты для работы с базами данных и SQL файлами.
* Интеграция с коммерческими системами управления версиями Perforce, Team Foundation Server, ClearCase, Visual SourceSafe.
* Инструменты для запуска тестов и анализа покрытия кода, включая поддержку всех популярных фреймворков для тестирования.

Недостатки:

* Придется потратить время для того, чтобы разобраться в этой IDE, поэтому начинающим программистам она может показаться сложноватой.

1. **Платформы и инструменты для создания веб-приложений.**

**Web-приложение** – программа с определенным набором функционала, использующая в качестве клиента браузер.

Различные типы веб-инструментов:

* Внешние библиотеки (содержат предварительно написанные фрагменты кода и шаблоны, которые можно повторно использовать и вспоминать в своем коде).
* Редакторы кода (имеют встроенные функции, такие как подсветка синтаксиса, отладчик и автозавершение кода, что упрощает процесс написания и редактирования кода).
* Системы контроля версий (предоставляют центральную платформу для управления и совместного написания кода).
* Системы баз данных (помогают создавать, редактировать и поддерживать базу данных).
* Выполнители задач (автоматизируют и выполняют повторяющиеся задачи в вашем коде, чтобы вам не приходилось контролировать каждую задачу).
* Локальное окружение (создают среду на компьютере для тестирования и запуска веб-приложений и решений, не требуя подключения к Интернету или инвестиций в веб-хостинг).
* Менеджеры пакетов (помогают загрузить и установить зависимости ваших фреймворков и библиотек правильным образом).

**Инструменты фронтенд-разработки:**

* **Visual Studio Code** — это бесплатный редактор кода с открытым исходным кодом, созданный компанией Microsoft для операционных систем Windows, macOS и Linux.
* **Bootstrap** — это широко используемая библиотека HTML, CSS и JavaScript с открытым исходным кодом.
* **Figma** — онлайн-сервис для разработки интерфейсов и прототипирования с возможностью организации совместной работы в режиме реального времени.

**Платформы:**

* 1. Django (Python) — это фреймворк для создания веб-приложений на языке Python. Он предоставляет множество встроенных функций и инструментов для быстрой разработки.
  2. Express.js (JavaScript (Node.js)) — это минималистичный фреймворк для создания веб-приложений с использованием языка JavaScript в среде Node.js.
  3. Angular (TypeScript) — это фреймворк, разработанный и поддерживаемый Google, для создания масштабируемых веб-приложений. Он использует TypeScript, что является надмножеством JavaScript.

**Инструменты:**

1. Flutter

Преимущества:

* Быстрая разработка
* Низкая конкуренция на рынке
* Отсутствие багов из прошлого
* Открытость
* Современный интерфейс

Недостатки:

* Трудность поиска ошибок, несмотря на отличную документацию
* Мало сторонних библиотек
* Непростой язык Dart

1. React Native

Преимущества:

* Общий код на JavaScript
* Быстрая разработка. Готовые компоненты
* Использование нативных возможностей

Недостатки:

* Производительность
* Ограничения платформенных возможностей

1. Kotlin Multiplatform

Преимущества:

* Общий код на Kotlin
* Доступные платформы: Android, JVM, IOS, JavaScript
* Гибкость

Недостатки:

* Возраст фреймворка
* Ограничения платформенных возможностей

1. Xamarin

Преимущества:

* Общий код для нескольких платформ
* Нативные интерфейсы и высокая производительность
* Поддержка C# и .NET
* Активное сообщество разработчиков и сторонних библиотек

Недостатки:

* Ограничения в доступе к нативным API
* Возможное увеличение размера приложения

1. **Создание кроссплатформенных приложений.**

**Кроссплатформенная разработка** — это подход к разработке приложений, который позволяет создавать приложения, работающие на разных платформах, используя общий код. Это означает, что разработчик может написать код один раз и использовать его для создания приложений для разных операционных систем, таких как Android, iOS, Windows и другие.

**Инструменты для мобильной разработки:**

* 1. Flutter

Преимущества:

* Быстрая разработка
* Низкая конкуренция на рынке
* Отсутствие багов из прошлого
* Открытость
* Современный интерфейс

Недостатки:

* Трудность поиска ошибок, несмотря на отличную документацию
* Мало сторонних библиотек
* Непростой язык Dart
  1. React Native

Преимущества:

* Общий код на JavaScript
* Быстрая разработка. Готовые компоненты
* Использование нативных возможностей

Недостатки:

* Производительность
* Ограничения платформенных возможностей
  1. Kotlin Multiplatform

Преимущества:

* Общий код на Kotlin
* Доступные платформы: Android, JVM, IOS, JavaScript
* Гибкость

Недостатки:

* Возраст фреймворка
* Ограничения платформенных возможностей
  1. Xamarin

Преимущества:

* Общий код для нескольких платформ
* Нативные интерфейсы и высокая производительность
* Поддержка C# и .NET
* Активное сообщество разработчиков и сторонних библиотек

Недостатки:

* Ограничения в доступе к нативным API
* Возможное увеличение размера приложения

1. **Характеристики и возможности юнит-тестирования приложений.**

**Модульное тестирование (юнит-тестирование)** применяется для исследования каждого отдельного элемента или объекта системы. Чтобы найти баги, применяя модульное тестирование, нужно знать, как устроена программа в целом и какой функционал каждого отдельного модуля. Этот уровень тестирования используется больше программистами, нежели тестировщиками. Они создают специальные тест-коды, с помощью которых можно проверить, выполняет ли программное обеспечение свое предназначение.

Юнит-тестирование приложений представляет собой метод, ориентированный на проверку отдельных компонентов программного кода, таких как функции, методы или классы. Оно осуществляется в изоляции от других частей приложения, обеспечивая целенаправленную проверку функциональности и избегая влияния внешних факторов. Автоматизированные тесты позволяют быстро и эффективно выявлять дефекты на ранних этапах разработки, предотвращать регрессии после внесения изменений, и обеспечивать стабильность приложения. Оценка покрытия кода и минимизация зависимостей способствуют созданию надежных и легко поддерживаемых кодовых баз. Юнит-тестирование также играет ключевую роль в контексте непрерывной интеграции, поддерживая быстрое и автоматизированное тестирование после каждого изменения кода, что в конечном итоге повышает качество и устойчивость приложений.

Unit tests исключают регрессию отлаженных модулей, позволяют избежать накопления ошибок в будущем при работе программного продукта. Они решают такие задачи как:

* Поиск и исправление имеющихся ошибок в тестируемом приложении на ранних этапах разработки. Данный прием позволяет снизить расходы.
* Понимание разработчиками базового кода программного обеспечения. Способствует упрощению читабельности и корректировки проекта.
* Перенос в код (или его фрагмент) в другие приложения. Сюда же относят повторное его использование.
* Замена проектной документации. При помощи Unit-тестирования разработчики, не знающие проект, смогут разобраться в принципах его функционирования.

Юнит-тестирование приложений обладает рядом возможностей, среди которых выделяется способность к изоляции функциональности, что позволяет тестировать отдельные части кода независимо друг от друга. Автоматизированные тесты обеспечивают быстрое и многократное выполнение тестовых сценариев, что повышает эффективность процесса разработки. Регрессионное тестирование дает уверенность в том, что новые изменения не нарушают стабильность существующего функционала. Возможность оценки покрытия кода позволяет определить, насколько хорошо код охвачен тестами, обеспечивая более полное тестирование. Минимизация зависимостей и поддержка непрерывной интеграции делают юнит-тестирование мощным инструментом для раннего выявления ошибок, обеспечивая высокую надежность и устойчивость приложений.

Существует множество инструментов для юнит-тестирования в различных языках программирования. Некоторые из популярных инструментов включают в себя:

* JUnit: Для языка Java, JUnit является стандартным фреймворком для написания и выполнения юнит-тестов.
* pytest: В мире Python, pytest предоставляет простой и мощный фреймворк для тестирования.
* JUnit и TestNG: Для тестирования приложений, написанных на языке Java, TestNG также является популярным фреймворком, предоставляющим дополнительные возможности в сравнении с JUnit.
* Mocha и Jasmine: Для JavaScript, Mocha и Jasmine предоставляют инструменты для тестирования кода на стороне клиента и сервера.

1. **Тестирование ПО. Системы автоматизированного тестирования.**

Можно выделить 2 **вида** **тестирования**: **1)** Ручное **2)** Автоматизированное

Ручное тестирование представляет собой процесс выполнения тест-кейсов человеком с применением вспомогательного ПО. Автоматизированное тестирование представляет собой процесс выполнения тестовых случаев инструментами и программами самостоятельно.

|  |  |  |
| --- | --- | --- |
| Параметр | **Ручное тестирование** | **Автоматизированное тестирование** |
| Скорость выполнения тестирования | Медленная | Быстрая |
| Исследовательское тестирование | Возможны случайные комбинации | Случайности невозможны |
| Надежность | Высокая надежность | Возможны человеческие ошибки |
| Объемы тестирования | Для малого кол-ва | Эффективен при больших объемах тестирования |

Главным преимуществом автоматизированного тестирования является возможность быстро повторить тестовый случай. Виды автоматизированного тестирования:

* **Модульное тестирование** применяется для исследования каждого отдельного элемента или объекта системы.
* **Интеграционное тестирование** проверяет, как отдельные модули взаимодействуют вместе, то есть интегрируясь друг с другом.
* **Системное тестирование** проверяется вся система целостно на наличие в ней багов. В системном тестировании тестировщик проверяет взаимосвязь между всеми аппаратными и программными компонентами системы и потом тестирует уже методику работы всей системы.
* **Предварительное тестирование (приемо-сдаточные испытания)** Этот уровень тестирования используют уже почти перед непосредственной передачей программного обеспечениязаказчику. Его используют, чтобы проверить соответствует ли разработанный продукт тем требованиям, которые выдвигалзаказчик.
* **Тестирование методом черного ящика** **(black-box testing)**

Оно подразумевает исследование работы программного обеспечения с точки зрения пользователя и не предполагает знаний о том, как устроена и работает внутренняя система.

* **Тестирование методом белого ящика (white-box testing)**

Этот метод тестирования уже базируется на знаниях внутреннего функционирования системы. Тестировщик должен знать,как работает код, чтобы выявить, где находятся баги.

* **Тестирование методом серого ящика (grey box testing)**

Для выполнения этого метода тестирования предполагает понимание о внутреннем устройстве программногообеспечения, но тестирование проводится с точки зрения конечного пользователя.

Системы автоматизированного тестирования (примеры):

1. **Selenium** — это среда тестирования для тестирования веб-приложений в различных браузерах и платформах, таких как Windows, Mac и Linux. 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.
2. **Behave** – это behavior-driven (BDD) фреймворк для тестирования. В Behave есть полная поддержка Gherkin, поэтому он является одним из самых популярных BDD-фреймворков для Python.
3. **Jasmine** – популярный фреймворк для тестирования веб-приложений в стиле BDD. Инструменте тестирования приложений на JavaScript, но вы вполне можете использовать его для автоматизации тестирования на Python.
4. **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.

Selenium включает в себя:

* **Selenium IDE** – плагин для браузера Firefox для записи действий пользователя
* **Selenium RC** – устаревшая библиотека для управления браузерами
* **Selenium WebDriver** – библиотека для управления браузерами
* **SeleniumGrid** – кластер Selenium-серверов для управления браузерами на разных компьютерах в сети
* **Selenium WebDriver**

**Selenium WebDriver.** Библиотека для управления браузерами. Представляет из себя семействодрайверов 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.

Плюсы:

* бесплатный продукт с открытым исходным кодом
* гибкий инструмент тестирования
* разработка с 2004 года

Минусы:

* можно тестировать только веб-приложения
* необходимость продвинутых навыков

Термины и базовые методы:

* **Локаторы** – метки для поиска определенных элементов на веб-странице

(**By.ID** –  по ID , **By.Classname** – по имени класса, **By.Xpath** – по прямому пути DOM, **By.Name** – по имени)

* **driver = webdriver.Chrome()** – инициализация браузера
* **driver.get("https://www.example.com")** – открытие веб-страницы
* **el = driver.find\_element(By.ID, 'element\_id\*')** – поиск элемента с использованием локатора
* **driver.execute\_script(arguments[0].click|', element)** – клик по элементу
* **pg=driver.page\_source** – получение HTML страницы
* **driver.quit()** – закрытие браузера и отчистка ресурсов

1. **Программы для создания инсталляторов (обзор систем).**

**Необходимость создания инсталляции:**Надежный, профессиональный установщик программного обеспечения является чрезвычайно важным для вашей продукции. Создание инсталляции программного продукта позволяет автоматизировать процесс установки программного продукта на компьютеры пользователей, предоставляя им при этом возможность выбора различных сценариев установки и обеспечивая корректность его дальнейшей работы.

**Инсталятторы:**

**СВОБОДНЫЕ ПРОГРАММЫ ДЛЯ СОЗДАНИЯ ИНСТАЛЛЯТОРОВ**

* **NSIS (Nullsoft Scriptable Install System)** - один из самых популярных инсталляторов. Система создания установочных программ для Microsoft Windows с открытым исходным кодом, разработанная компанией Nullsoft — автором плеера Winamp. NSIS был задуман как альтернатива InstallShield, предназначенного для коммерческих продуктов.

NSIS допускает расширение с помощью плагинов, которые могут быть написаны на C++, С и Delphi. Плагины применяются для улучшения функциональности и интерфейса установщика и могут быть вызваны в любой части кода NSIS-скрипта.

С пакетом NSIS поставляются несколько плагинов, позволяющих добавлять новые страницы, заменять фоновые изображения, скачивать файлы из Интернета, выполнять математические вычисления, обновлять файлы и др.

* **IzPack** - java-разработчик. Это универсальный разработчик, способный создавать дистрибутивы для Unix, Linux, FreeBSD, Mac OS X и Windows 2000, XP. Позволяет создавать как обычные пакеты для аналитики, так и Веб инсталляторы, которые подгружают необходимые файлы по мере необходимости. Данная возможность позволяет свести к минимуму количество загружаемых файлов в зависимости от требуемой конфигурации установки;
* **Inno Setup** — система создания инсталляторов для Windows-программ с открытым исходным кодом. Впервые выпущенный в 1997 году, Inno Setup сегодня конкурирует и даже превосходит многие коммерческие установщики по функциональности и стабильности.
* **WiX (Windows Installer XML)** - специализированный продукт от Microsoft для создания MSI и MSM инсталляционных пакетов.

**КОММЕРЧЕСКИЕ ПРОГРАММЫ ДЛЯ СОЗДАНИЯ ИНСТАЛЛЯТОРОВ**

* **InstallShield** - система создания инсталляторов и пакетов программного обеспечения для Microsoft Windows и Linux с закрытыми исходными кодами. Программный инструмент используется главным образом для установки программного обеспечения для Microsoft Windows, как на десктопные, так и серверные платформы, а также может быть использован для управления приложениями на различных портативных и мобильных устройствах.
* **WISE** - простой в освоении с богатыми возможностями генератор инсталляторов;
* **VISE** - профессиональный инсталлятор для Windows, Mac OS X и Macintosh;
* **СreateInstall** это универсальный, гибкий и мощный инсталлятор как для профессиональных разработчиков, так и для начинающих. С помощью этой программы Вы можете создать полнофункциональные инсталляционные программы для Ваших приложений, а также самораспаковывающиеся архивы с высокой степенью сжатия и многое другое;
* **Advanced Installer** - Позволяет создавать инсталляторы для java приложений. Создает дополнительный исполняемый файл.

1. **Создание справочной системы.**

Справочная система программы, состоящая из одно или нескольких файлов справок, предназначена для предоставления пользователю программы полной и исчерпывающей информации, о том, как работать с программой и для чего данный программный продукт нужен. Справочная система должна удовлетворять следующим требованиям:

* Давать полное описание по вопросам использования программы.
* Иметь графические материалы по вопросам использования программы.
* Быть доступной для вызова из любой формы программы.
* Иметь контекстные описания и удобную систему поиска информации.
* Иметь минимально возможный размер.

Разработка *справочной системы*требует решения следующих основных задач.

1. **Планирование системы справок.**На этом этапе составляется перечень разделов справочной службы и необходимых перекрестных ссылок.
2. **Создание текстовых файлов,**содержащих описания справочных разделов. Текстовые файлы готовятся с помощью любого текстового редактора, поддерживающего расширенный текстовый формат *RTF.*В них включаются специальные управляющие символы для создания перекрестных ссылок и подключения растровых изображений.
3. **Разработка проектного файла,**содержащего специальные команды для компилятора файлов контекстной справки. Проектный файл описывает структуру справочной службы в целом, в нем каждому разделу присваивается уникальный целочисленный идентификатор.
4. **Разработка файла содержания.**Содержание активизируется при запуске *файла справки,*атакже после щелчка по кнопке *Содержание*в окне справочной службы.
5. **Компиляция***файлов справок из RTF файлов.* Файлы контекстной справки могут быть двух форматов: \*.hlp и \*.chm. В качестве компилятора *файлов* *справки*поставлялся стандартный компилятор *Microsoft Help Workshop*(файл hcw.exe из каталога HELP – TOOLS каталога размещения *Delphi)* для \*.hlp файлов или *Microsoft HTML Help Workshop* *(hhw.exe)* для \*.chm файлов*.*
6. **Тестирование и отладка***справочной системы.*
7. **Связь программы**с разделами*справочной службы.*

Оба упомянутых формата файлов контекстной справки разработаны корпорацией Microsoft. Формат HLP (WinHelp) является устаревшим, хотя и встречается до сих пор, но в настоящий момент наиболее распространенным форматом справки в среде *Windows* является CHM (HTMLHelp). *Файл* CHM является сжатым набором *HTML*-страниц, своего рода, электронной книгой, и может содержать весь набор Web-страниц: текст, таблицы, рисунки, ссылки, медиа-файлы, и прочее, что делает такую справочную систему мощным инструментом.

1. **Рефакторинг кода. Обзор Сode Smell и метрик оценки сложности кода.**

**Рефакторинг** – это контролируемая техника совершенствования структуры существующего кода. Суть рефакторинга заключается во внесении серии мелких изменений (с сохранением функциональности приложения), каждое из которых «слишком мелкое, чтобы тратить на него время». Тем не менее эффект от внесения всех этих изменений достаточно ощутимый.

**Что дает рефакторинг?**

* Улучшение композиции программного кода
* Облегчение понимания программного кода
* Упрощение поиска ошибок
* Ускорение написания кода
* Сокращение времени на добавление нового функционала
* Снижение затрат на поддержку

**Сode Smell (код с запахом)** – термин, обозначающий код с признаками проблем в системе. Свидетельствует о необходимости проводить рефакторинг.

**Общие «запахи» объектно-ориентированного кода**

* **Дублирование кода — это** использование одинаковых структур кода в нескольких местах. Объединение этих структур позволит улучшить программный код.
* **Длинный метод –** более 10–12 строк (условно). Чем длиннее процедура, тем труднее её понять.
* **Большой класс:** если класс реализует слишком обширную функциональность, стоит подумать о вынесении некоторой части кода в подкласс.
* **Длинный список параметров.** В таких списках параметровтрудно разбираться, они становятся противоречивыми и сложными в использовании.
* **Расходящиеся модификации –** ситуация, когда есть один класс, в котором производится много различных изменений.
* **Стрельба дробью –** ситуация, при которой одно изменение, затрагивающее много классов.
* **Завистливые функции (Feature Envy) -** метод обращается к данным другого объекта чаще, чем к собственным данным.
* **Группы данных,** если встречаются совместно, нужно превращать в самостоятельный класс.
* **Одержимость элементарными типами –** использование элементарных типов вместо маленьких объектов для небольших задач.
* **Switch-case,** когда один и тот же Switch-case разбросан по всему коду, и при добавлении нового условия необходимо потом долго искать все эти свитчи.
* **Ленивый класс –** класс, затраты на существование которого не окупаются выполняемыми им функциями.
* **Цепочка вызовов –** ситуация, когда клиент запрашивает у одного объекта другой объект, другой объект запрашивает ещё один объект и т. д. Любые изменения промежуточных связей означают необходимость модификации клиента.
* **Посредник возникает** при чрезмерном использовании делегирования, приводящем к появлению классов, у которых большинство методов состоит только из вызова метода другого класса.
* **Неуместная близость,** возникающая, когда классы чаще, чем следовало бы, погружены в закрытые части друг друга
* **Альтернативные классы с разными интерфейсами –** ситуация, при которой два класса, в которых часть функциональности общая, а методы, реализующие её, имеют разные параметры.
* **Классы данных –** это классы, которые содержат только поля и методы для доступа к ним, это просто контейнеры для данных, используемые другими классами.
* **Излишние комментарии.** Хороший и лаконичный код (почти всегда) понятен и без пояснений.

**Рефакторинг: метрики для измерения сложности кода**

1. **Количество строк кода**
2. **Цикломатическая сложность**

Цикломатическое число МакКейба показывает требуемое количество проходов для покрытия всех контуров сильно связанного графа или количества тестовых прогонов программы, необходимых для исчерпывающего тестирования по принципу «работает каждая ветвь».

Показатель цикломатической сложности может быть рассчитан для модуля,метода и других структурных единиц программы.

1. **Метрики Холстеда**

Основу метрики Холстеда составляют четыре измеряемые характеристики программы на основании этих характеристик рассчитываются оценки:

1. *Словарь программы*
2. *Длина программы*
3. *Объем программы*
4. *Сложность программы*
5. **Индекс поддерживаемости**

*Этот индекс говорит нам о том, насколько сложно будет поддерживать или редактировать кусок программы. Этот параметр рассчитывается на основе чисел, полученных из метрик, посчитанных выше*

1. **Рефакторинг кода на Java и Python.**

Рефакторинг в Java имеет свои особенности, связанные с языковыми особенностями Java и обилием инструментов, предназначенных для упрощения этого процесса. Многие Java-разработчики используют мощные интегрированные среды разработки, такие как IntelliJ IDEA, Eclipse или NetBeans, которые предоставляют обширный набор инструментов для автоматизации рефакторинга. Эти среды позволяют выполнять различные рефакторные операции с минимальными усилиями.

Вот несколько особенностей рефакторинга в Java:

* Структура проекта: Java IDE предоставляют возможность переименовывать пакеты целиком, автоматически обновляя все ссылки в проекте. Это упрощает реорганизацию структуры проекта.
* Интерфейсы и абстрактные классы: Java активно использует интерфейсы и абстрактные классы. Рефакторинг может включать в себя выделение интерфейсов, вынос общей логики в абстрактные классы, а также улучшение иерархии классов.
* Аннотации: Введение аннотаций в Java позволяет использовать их для маркировки классов и методов. Рефакторинг может включать в себя выделение общей функциональности, ранее осуществляемой через аннотации.
* Лямбда-выражения: В Java 8 и более поздних версиях появились лямбда-выражения, которые предоставляют более компактный и выразительный способ написания анонимных классов. Рефакторинг может включать в себя преобразование старого кода в использование лямбда-выражений.
* Работа с исключениями: В Java рефакторинг может включать в себя более эффективное использование исключений, включая перехват более конкретных исключений, вынос общей логики обработки исключений в отдельные методы и т.д.

**Пример**

**До рефакторинга**

**public** **boolean** max(**int** a, **int** b) {

**if**(a > b) {

**return** true;

    } **else** **if**(a == b) {

**return** false;

    } **else** {

**return** false;

    }

}

**После рефакторинга**

**public** **boolean** max(**int** a, **int** b) {

**return** a > b;

}

Рефакторинг в Python обычно осуществляется с учетом динамической природы языка и его особенностей. Существуют инструменты, такие как Rope, PyCharm, и др., предоставляющие автоматизацию определенных видов рефакторинга. Они могут предложить изменения, подсказки и даже автоматически выполнить определенные операции рефакторинга.

Ниже приведены основные особенности и приемы рефакторинга в Python:

* Интерпретируемость и динамическая типизация: Python является интерпретируемым языком с динамической типизацией. Это означает, что многие ошибки, связанные с типами данных, могут проявиться только во время выполнения программы. При рефакторинге важно учесть этот аспект и проверить корректность типов данных.
* Простота и читаемость кода: В Python уделяется особое внимание простоте и читаемости кода. Рефакторинг включает в себя улучшение структуры кода, избегание сложных конструкций и использование понятных имен переменных и функций.
* List Comprehensions и генераторы: В Python часто используются List Comprehensions и генераторы для создания списков и последовательностей. Эти конструкции делают код более кратким и выразительным.
* Использование функциональных возможностей: С появлением функциональных возможностей в Python, таких как lambda-функции, функции высшего порядка и функциональные инструменты (например, map, filter, reduce), рефакторинг может включать в себя применение более функционального стиля программирования.
* Деструктуризация и множественное присваивание: В Python удобно использовать деструктуризацию для присвоения значений нескольким переменным одновременно.
* Использование контекстных менеджеров: Контекстные менеджеры в Python позволяют более безопасно и читаемо работать с ресурсами, такими как файлы или соединения с базой данных. Рефакторинг может включать в себя использование контекстных менеджеров для улучшения обработки ресурсов.

**Пример**

**До рефакторинга**

numbers = [1, 2, 3, 4, 5]  
squared\_numbers = []  
for num in numbers:  
 squared\_numbers.append(num \*\* 2)  
print(squared\_numbers)

**После рефакторинга**

numbers = [1, 2, 3, 4, 5]  
squared\_numbers = [num \*\* 2 for num in numbers]  
print(squared\_numbers)

1. **Автоматизированный рефакторинг в IDE.**

Большинство современных IDE предоставляют встроенные инструменты для автоматизации рефакторинга. Эти инструменты обеспечивают удобный интерфейс для проведения рефакторинга, а также предоставляют набор правил и шаблонов для общих операций рефакторинга. Также некоторые среды разработки предоставляют функциональность автоматического обнаружения потенциальных улучшений в коде. Например, они могут выделять дублированный код или предлагать изменения для повышения читаемости.

Хорошие инструменты рефакторинга обеспечивают безопасность внесенных изменений, предоставляя возможность отмены операций рефакторинга или предварительного просмотра изменений перед их фактическим внесением. Также при рефакторинге может потребоваться обновление всех связанных элементов, таких как ссылки на методы, классы и переменные. Инструменты автоматизированного рефакторинга обычно обеспечивают эту функциональность. Еще некоторые IDE интегрируются с системами контроля версий, позволяя отслеживать изменения и вносить соответствующие правки в код в рамках рабочего процесса с использованием этих систем.

В современных IDE предоставляется широкий набор операций рефакторинга, включая объединение/разделение методов, внедрение/извлечение интерфейсов, оптимизацию импортов и многое другое. Это позволяет разработчикам проводить разнообразные изменения в коде с минимальными усилиями.

PyCharm - это популярная интегрированная среда разработки (IDE) для языка программирования Python, разработанная компанией JetBrains. Вот некоторые из функций автоматизированного рефакторинга в PyCharm:

1. **Переименование переменных и символов:** PyCharm позволяет изменять имена переменных, функций, классов и других символов в коде с автоматическим обновлением всех ссылок на них. Это делается с использованием функции "Rename" (переименование), и PyCharm самостоятельно просматривает код, обеспечивая безопасное изменение имени.
2. **Выделение кода в функцию или метод:** Разработчики могут выделить фрагмент кода и автоматически превратить его в функцию или метод, используя функцию "Extract Method" (выделение метода) или "Introduce Variable" (введение переменной).
3. **Удаление неиспользуемого кода:** PyCharm предоставляет возможность автоматического удаления неиспользуемых импортов, переменных, функций и классов, что помогает поддерживать код в аккуратном состоянии.
4. **Реорганизация кода (Code Reformat):** Инструменты реформатирования кода в PyCharm позволяют автоматически привести код к соответствующему стилю оформления, что улучшает его читаемость.
5. **Инспекции кода и автоматические исправления:** PyCharm предоставляет инспекции кода, выявляющие потенциальные проблемы и стилистические ошибки. Разработчик может использовать автоматические исправления, чтобы быстро устранить эти проблемы.
6. **Поддержка тестирования:** PyCharm помогает рефакторить код, связанный с тестированием, например, вынос кода в setUp и tearDown методы, создание заглушек и многое другое.
7. **Интеграция с системами контроля версий:** PyCharm интегрируется с популярными системами контроля версий (например, Git) и предоставляет возможность проводить рефакторинг кода в рамках рабочего процесса с использованием этих систем.

IntelliJ IDEA, разработанный также компанией JetBrains, предоставляет мощные инструменты автоматизированного рефакторинга для языков программирования, таких как Java, Kotlin, Scala и другие. Вот несколько функций автоматизированного рефакторинга в IntelliJ IDEA:

1. **Переименование (Rename):** Это одна из базовых функций рефакторинга. Вы можете изменить имя переменной, метода, класса или пакета, и IntelliJ IDEA автоматически обновит все ссылки на это имя в вашем коде.
2. **Выделение метода (Extract Method):** Эта функция позволяет выделить фрагмент кода в новый метод. IntelliJ IDEA автоматически вставляет вызов нового метода вместо выделенного кода.
3. **Выделение переменной (Extract Variable):** Аналогично Extract Method, но выделение переменной позволяет создать новую переменную для фрагмента выделенного кода.
4. **Перемещение кода (Move):** Позволяет перемещать классы, методы, поля и другие элементы между пакетами или классами.
5. **Изменение сигнатуры метода (Change Signature):** Это позволяет изменять параметры метода, добавлять новые параметры или изменять типы параметров. IntelliJ IDEA автоматически обновляет все вызовы этого метода.
6. **Поиск дублированного кода (Find Duplicates):** IntelliJ IDEA способен выявлять дублированный код в проекте и предоставлять возможности для его рефакторинга.
7. **Реорганизация кода (Code Cleanup):** Инструмент Code Cleanup в IntelliJ IDEA автоматически приводит ваш код к соответствующему стилю оформления и правилам.
8. **Поддержка Javadoc и комментариев:** IntelliJ IDEA помогает рефакторить даже комментарии, обновляя их в соответствии с изменениями в коде.
9. **Интеграция с системами контроля версий:** Поддержка систем контроля версий, таких как Git, позволяет проводить рефакторинг кода в среде с учетом изменений в истории проекта.
10. **Генераторы документации для Java и Python (обзор систем).**

**Генератор документации** – программа или пакет программ, позволяющая получать документацию, предназначенную для программистов (документация на API) и/или для конечных пользователей системы, по особым образом комментированному исходному коду и, в некоторых случаях, по исполняемым модулям (полученным на выходе компилятора).

Обычно генератор анализирует исходный код программы, выделяя синтаксические конструкции, соответствующие значимым объектам программы: типам, классам и их членам/свойствам/методам, процедурам/функциям и т. п. В ходе анализа также используется мета-информация об объектах программы, представленная в виде документирующих комментариев. На основе всей собранной информации формируется готовая документация, как правило, в одном из общепринятых форматов — HTML, HTMLHelp, PDF, RTF и других.

**Javadoc** — это генератор документации в HTML-формате из комментариев исходного кода Java. Javadoc, обрабатывая файл с исходным текстом программы, выделяет помеченную документацию из комментариев и связывает с именами соответствующих классов, методов и полей. Таким образом, при минимальных усилиях создания комментариев к коду, можно получить хорошую документацию к программе.

Документ Javadoc не влияет на производительность в Java, поскольку все комментарии удаляются во время компиляции. Написание комментариев и документации Javadoc предназначено для лучшего понимания кода и, следовательно, лучшего его обслуживания.

**Doxygen** является отличным инструментом для генерации документации из исходного кода. Инструмент нацелен на документирование программного обеспечения, написанного на языке C++, однако на самом деле данная система поддерживает гораздо большое число других языков, включая C, C#, PHP, Java, Python и другие. С помощью Doxygen, можно создать онлайн HTML документацию. Doxygen — консольная программа в духе классической Unix. Она работает подобно компилятору, анализируя исходные тексты и создавая документацию. Выходными данными при использовании этого генератора будут CHM, HTML, RTF, LaTex, Doc book, XML, Man pages.

Самым большим преимуществом использования Doxygen является то, что будет иметься последовательность всей документации исходного кода. Она также может помочь создавать структуру кода с использованием недокументированных исходных файлов. Все, что нужно сделать, это настроить его соответствующим образом.

**Sphinx** это популярный инструмент позволяющий создавать текстовые документы и преобразовывать их в различные форматы. Это удобно при использовании систем управления версиями, предназначенных для отслеживания изменений. Он доступен по лицензии BSD и поддерживает несколько языков программирования, таких как Python, C, C++, PHP, JavaScript. Он может быть использован как для проектной документации так и для документации кода. При генерировании документации из кода в данном случае доступны следующие выходные форматы: HTML, CHM, LaTex, Man pages, ePub.

Sphinx избавляет от рутинных действий и предлагает автоматическую функциональность для решения типовых проблем, например, индексирования заголовков и специального выделения кода (например, при включении в документ фрагментов кода) с соответствующим выделением синтаксиса.

***Swagger*** *— инструмент для подготовки документации к API и проведения тестов API. Инструмент поддерживает спецификацию OpenAPI, которая набор формальных плавил, которые нужно использовать при описании API в формате JSON или YAML. Swagger позволяет автоматически генерировать клиентский код для всех популярных языков программирования, включая Java, C++, C#, Kotlin, PHP, Python и другие. Выходными данными при использовании этого генератора будут HTML и Confluence.*

*Swagger стал популярнее других фреймворков и сервисов для создания документации, потому что он предлагает удобочитаемый формат. В нём без проблем разберётся как человек, так и машина. Swagger — это многофункциональная платформа, которая сокращает время реализации проекта.*

1. **UML диаграммы в разработке ИС.**

UML обеспечивает поддержку всех этапов жизненного цикла ИС и предоставляет для этих целей ряд графических средств – диаграмм:

* На этапе создания *концептуальной модели* для описания бизнес-деятельности используются модели бизнес-прецедентов и диаграммы видов деятельности, для описания бизнес-объектов – модели бизнес-объектов и диаграммы последовательностей.
* На этапе создания *логической модели* ИС описание требований к системе задается в виде модели и описания системных прецедентов, а предварительное проектирование осуществляется с использованием диаграмм классов, диаграмм последовательностей и диаграмм состояний.
* На этапе создания *физической модели* детальное проектирование выполняется с использованием диаграмм классов, диаграмм компонентов, диаграмм развертывания.

Ниже приводятся определения и описывается назначение перечисленных диаграмм и моделей применительно к задачам проектирования ИС (в скобках приведены альтернативные названия диаграмм, использующиеся в современной литературе):

*Диаграммы прецедентов* (диаграммы вариантов использования, use case diagrams) – это обобщенная модель функционирования системы в окружающей среде.

*Диаграммы прецедентов используют* чтобы исключить дублирование функциональных возможностей системы в рамках прецедента. Глядя на схему, разработчик понимает, что определенные элементы интерфейса и кода можно использовать повторно. Это экономит время и ресурсы при разработке.

*Диаграммы деятельности* (диаграммы активности, activity diagrams) – модель бизнес-процесса или поведения системы в рамках прецедента.

*Диаграмму действий используют* чтобы понять, какие действия приводят к нужному результату. Ведь систему проектируют для достижения целей, а не просто так.

*Диаграммы последовательностей* (sequence diagrams) – модель взаимодействия объектов информационной системы во времени и обмена сообщения между ними.

*Диаграмма последовательности нужна* чтобы детализировать сценарии использования функциональных возможностей, а также уточнить время жизни объектов. Схема помогает экономить вычислительные мощности и точно проектировать сценарии использования на уровне интерфейсов.

*Диаграммы состояний* (statechart diagrams) – модель динамического поведения системы и ее компонентов при переходе из одного состояния в другое.

*Диаграммы классов* (class diagrams) – логическая модель базовой структуры системы, отражает статическую структуру системы и связи между ее элементами.

*Диаграммы классов активно используются* для проектов, опирающихся на объектно-ориентированное программирование.

*Диаграммы компонентов* (component diagrams) – модель иерархии подсистем, отражает физическое размещение баз данных, приложений и интерфейсов ИС.

*Диаграмма компонентов* *позволяет* разбить комплексную систему на более мелкие составляющие и наглядно продемонстрировать установленные между ними связи.

*Диаграммы развертывания* (диаграммы размещения, deployment diagrams) – модель физической архитектуры системы, отображает аппаратную конфигурацию ИС.

*Чаще всего* *такими диаграммами пользуются* проектировщики систем — для наглядной оценки производительности, масштабируемости, надежности и портативности.

1. **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.

Огромную роль играет компонент под названием Docker Container, который дает возможность отделения программы от инфраструктуры. Docker позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер. Он будет запускаться в изолированной среде, которая не оказывает никакого влияния на основную операционную систему. Поэтому его можно запускать на различных системах. В своем ядре docker позволяет запускать практически любое приложение, безопасно изолированное в контейнере.

Основные концепции Docker:

* Каждый контейнер Docker начинается с Dockerfile – это Текстовый файл с последовательно расположенными инструкциями для создания образа Docker. Файл создаётся по принципу «одна строка — одна команда».
* После конфигурации Dockerfile используется утилита docker build для создания образа на его основе.
* Docker image (образ) – это переносимый файл, содержащий спецификации программных компонентов, выполняемых в контейнере.
* Команда Docker run ищет образ и запускает контейнер на его основе. Каждый контейнер основан на образе. На одном образе может быть основано несколько контейнеров.
* Docker Hub – популярный публичный репозиторий, используемый по умолчанию в Docker. Обеспечивает интеграцию с GitHub и BitBucket.
* Docker Daemon – фоновая служба на хосте, которая отвечает за создание, запуск и уничтожение контейнеров.
* Docker Client – утилита командной строки в Docker для управления демоном. Любое взаимодействие с контейнером проходит через Daemon.
* Docker Registry – служба в Docker, выполняющая функции репозитория. Позволяет следить за версиями образов, создавать приватные репозитории.

Платформа и средства контейнерной виртуализации могут быть полезны в следующих случаях:

* упаковывание приложения (и так же используемых компонент) в docker контейнеры;
* раздача и доставка этих контейнеров командам для разработки и тестирования;
* выкладывания этих контейнеров на продакшены, как в дата центры так и в облака.

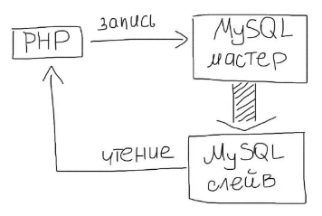
1. **Инструменты оптимизации работы высоконагруженных систем.**

Высоконагруженные системы - системы, в которых из-за неправильной настройки при выполнении машинных операций происходит большая нагрузка на определённые компоненты серверов. 5 качеств высоконагруженных систем:

* Это система с огромной аудиторией
* Это высокоресурсная система
* Это распределенная система
* Это интерактивная система
* Это система с позитивной динамикой

Инструменты:

1. *Сервисно-ориентированная архитектура* (service-oriented architecture, SOA) — модульный подход к разработке программного обеспечения, базирующийся на обеспечении удаленного по стандартизированным протоколам использования распределённых, слабо связанных легко заменяемых компонентов (сервисов) со стандартизированными интерфейсами.
2. *Вертикальное масштабирование* – увеличение производительности каждого компонента системы с целью повышения общей производительности. Масштабируемость в этом контексте означает возможность заменять в существующей вычислительной системе компоненты более мощными и быстрыми по мере роста требований и развития технологий. Это самый простой способ масштабирования, так как не требует никаких изменений в прикладных программах, работающих на таких системах.
3. *Горизонтальное масштабирование* – разбиение системы на более мелкие структурные компоненты и разнесение их по отдельным физическим машинам (или их группам), и (или) увеличение количества серверов, параллельно выполняющих одну и ту же функцию. Масштабируемость в этом контексте означает возможность добавлять к системе новые узлы, серверы для увеличения общей производительности. Этот способ масштабирования может требовать внесения изменений в программы, чтобы программы могли в полной мере пользоваться возросшим количеством ресурсов.
4. *Отложенные вычисления* (lazy evaluation) – применяемая в некоторых языках программирования стратегия вычисления, согласно которой вычисления следует откладывать до тех пор, пока не понадобится их результат.
5. *Асинхронная обработка* – формирование очереди, когда мгновенное выполнение задачи невозможно (например, уведомление миллиона пользователей)
6. *Использование толстого клиента* (Rich-клиент) – в архитектуре клиент — сервер — это приложение, обеспечивающее (в противовес тонкому клиенту) расширенную функциональность независимо от центрального сервера. Часто сервер в этом случае является лишь хранилищем данных, а вся работа по обработке и представлению этих данных переносится на машину клиента
7. *Кеширование* – промежуточный буфер с быстрым доступом к нему, содержащий информацию, которая может быть запрошена с наибольшей вероятностью. Доступ к данным в кэше осуществляется быстрее, чем выборка исходных данных из более медленной памяти или удалённого источника, однако её объём существенно ограничен по сравнению с хранилищем исходных данных. (Система должна работать и без кеширования!, Кешируйте данные, а не готовые решения)
8. *Функциональное разделение:* Например, форум и новости на разных серверах…
9. *Репликация* позволяет создать полный дубликат базы данных. Так, вместо одного сервера у вас их будет несколько:

Репликация позволяет использовать два или больше одинаковых серверов вместо одного. Операций чтения (SELECT) данных часто намного больше, чем операций изменения данных (INSERT/UPDATE). Поэтому, репликация позволяет разгрузить основной сервер за счет переноса операций чтения на слейв.

1. *Шардинг* – это другая техника масштабирования работы с данными. Суть его в разделении (партиционирование) базы данных на отдельные части так, чтобы каждую из них можно было вынести на отдельный сервер. Этот процесс зависит от структуры Вашей базы данных и выполняется прямо в приложении в отличие от репликации
2. *Виртуальные шарды:* Физически сервер еще один, но логически вы уже шардите, а затем по необходимости разносите. Заранее написан алгоритм позволяющий делить данные (базы данных) по серверам. Разбиение заранее предопределено.
3. *Партицинионирование* – разбиение таблиц, содержащих большое количество записей, на логические части по неким выбранным администратором критериям. Партиционирование таблиц делит весь объем операций по обработке данных на несколько независимых и параллельно выполняющихся потоков, что существенно ускоряет работу СУБД. Для правильного конфигурирования параметров партиционирования необходимо, чтобы в каждом потоке было примерно одинаковое количество записей.
4. *Денормализация* – хранение избыточных данных для быстрого предоставления их по запросу. Например, отдельное хранение данных каждого из участников виртуального диалога.
5. *Введение избыточности –* это не результат кривых рук. Это не недоделанная нормализация, намеренное приведение структуры базы данных в состояние, не соответствующее критериям нормализации, обычно проводимое с целью ускорения операций чтения из базы за счет добавления избыточных данных.
6. *Параллельное выполнение:* Например, поисковое обращение ищется одновременно на огромном количестве машин. Яндекс запрос обрабатывается на нескольких десятках серверов…
   1. Цели и задачи управления процессом разработки ПО.
   2. Жизненный цикл программных систем.
   3. Обзор информационных систем.
   4. Баг-трекинг системы.
   5. Системы управления версиями (обзор систем).
   6. Инструменты для мобильной разработки (обзор систем).
   7. Система управления версиями и поиска ошибок Redmine.
   8. Системы управления версиями Git.
   9. Интегрированные среды разработки (обзор систем).
   10. Платформы и инструменты для создания веб-приложений.
   11. Создание кроссплатформенных приложений.
   12. Характеристики и возможности юнит-тестирования приложений.
   13. Тестирование ПО. Системы автоматизированного тестирования.
   14. Selenium – библиотека для тестирования веб-приложений.
   15. Программы для создания инсталляторов (обзор систем).
   16. Создание справочной системы.
   17. Рефакторинг кода. Обзор Сode Smell и метрик оценки сложности кода.
   18. Рефакторинг кода на Java и Python.
   19. Автоматизированный рефакторинг в IDE.
   20. Генераторы документации для Java и Python (обзор систем).
   21. UML диаграммы в разработке ИС.
   22. Программное обеспечение для автоматизации развёртывания и управления приложениями Docker.
   23. Инструменты оптимизации работы высоконагруженных систем.