**1. Цели и задачи технологий разработки ПО. Особенности современных проектов разработки ПО.**

**Ответ:**

Цели:

1. Качество ПО: Обеспечить создание надежного, масштабируемого, легко поддерживаемого и безопасного программного продукта.
2. Удовлетворение потребностей пользователя: Создать продукт, максимально соответствующий ожиданиям конечного пользователя.
3. Оптимизация времени и ресурсов: Снизить затраты на разработку без ущерба для качества.
4. Инновации: Использование современных технологий и методов для повышения конкурентоспособности продукта.
5. Гибкость и адаптивность: Возможность быстрой адаптации ПО к изменяющимся требованиям бизнеса и рынка.

Задачи:

1. Системный анализ: Изучение требований заказчика, определение целей проекта, выявление ограничений и рисков.
2. Планирование: Разработка плана проекта, включая сроки, этапы, распределение ресурсов и бюджет.
3. Моделирование: Создание архитектуры и моделей системы для понимания структуры ПО.
4. Разработка: Написание и тестирование кода, разработка интерфейсов и функциональных модулей.
5. Тестирование: Проверка качества продукта на всех этапах (модульное, интеграционное, системное тестирование).
6. Внедрение и сопровождение: Развертывание готового продукта и его дальнейшая поддержка.

Особенности современных проектов разработки ПО

1. Гибкие методологии разработки:
   * Использование Agile, Scrum, Kanban и других подходов, которые позволяют быстрее реагировать на изменения требований.
   * Итеративная и инкрементальная разработка, обеспечивающая видимый прогресс.
2. Интеграция DevOps:
   * Объединение процессов разработки и эксплуатации для автоматизации тестирования, сборки и развертывания.
   * Инструменты CI/CD (Jenkins, GitLab CI/CD) и облачные платформы.
3. Фокус на UX/UI:
   * Современные проекты уделяют особое внимание пользовательскому опыту и удобству интерфейсов.
4. Микросервисы и контейнеризация:
   * Использование микросервисной архитектуры для модульности и масштабируемости систем.
   * Контейнеризация с помощью Docker, Kubernetes для упрощения развертывания и управления.
5. Использование ИИ и машинного обучения:
   * Включение технологий ИИ для анализа данных, автоматизации процессов и повышения функциональности продукта.
6. Кибербезопасность:
   * Современные проекты включают безопасность на всех этапах разработки (DevSecOps).
   * Учет требований GDPR, CCPA и других стандартов защиты данных.
7. Облачные технологии:
   * Миграция приложений в облако (AWS, Azure, Google Cloud).
   * Создание SaaS (Software-as-a-Service) решений.
8. Командная работа и удаленная разработка:
   * Использование средств коллективной работы (Slack, Jira, Trello, Confluence).
   * Распределенные команды и удаленная работа стали стандартом.
9. Быстрая смена технологий:
   * Регулярное обновление используемых инструментов и фреймворков.
   * Учет трендов, таких как Low-Code/No-Code разработки.
10. Ориентированность на Agile-тестирование:
    * Постоянное тестирование на всех стадиях разработки.
    * Автоматизация тестов для ускорения итераций.

**2. Основные определения: программа, программный продукт, программное обеспечение, программная инженерия, жизненный цикл ПО. Стандарты программной инженерии.**

**Ответ:**

* **Программа:** Набор инструкций, выполняемых компьютером для выполнения задач.
* **Программный продукт:** Программа, дополненная документацией, лицензией и поддержкой.
* **Программное обеспечение (ПО):** Совокупность программ и данных для выполнения вычислений на ЭВМ.
* **Программная инженерия:** Дисциплина, включающая методы, инструменты и процессы для создания и сопровождения ПО.
* **Жизненный цикл ПО:** Процесс от идеи до вывода продукта из эксплуатации, включая анализ, проектирование, реализацию, тестирование, сопровождение.

**Стандарты программной инженерии:**

* **ISO/IEC 12207:** Процессы жизненного цикла ПО.
* **ISO/IEC 25010:** Модель качества ПО.
* **PMBOK, PRINCE2:** Управление проектами.

**3. Классификация программного обеспечения. Определение и состав системы программирования. Примеры.**

**Ответ:**

Классификация ПО:

* **Системное ПО:** ОС, драйверы.
* **Прикладное ПО:** Пользовательские программы (MS Word, AutoCAD).
* **Инструментальное ПО:** Средства разработки (компиляторы, IDE).

**Состав системы программирования:**

* Компиляторы, трансляторы, редакторы, отладчики.
* Пример: IDE Visual Studio, Code::Blocks.

**4. Интегрированная среда разработки Visual Studio. Представление символьной информации в кодировке Windows-1251. Примеры.**

**Ответ:**

1. Инструменты для создания, отладки и тестирования приложений.

2. Поддержка символьной информации: кодировки Windows-1251 для русскоязычного текста.

**5. Компоненты классической системы программирования. Трансляторы, ассемблеры, интерпретаторы. Схема работы транслятора.**

**Ответ:**

Классическая система программирования:

* **Трансляторы:** Преобразуют код высокого уровня в объектный код.
* **Ассемблеры:** Генерируют машинный код из ассемблерного языка.
* **Интерпретаторы:** Выполняют код построчно.

**Схема работы транслятора:**

1. Лексический анализ.
2. Синтаксический анализ.
3. Генерация объектного кода.

**6. Структура классической системы программирования. Этапы обработки исходного кода программы. отладчик. Основные элементы языка программирования. Алфавит языка программирования, символы времени трансляции, символы времени выполнения.**

**Ответ:**

- **Этапы обработки кода:**

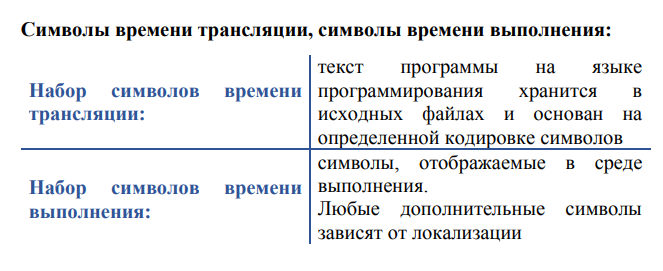
* Сборка, компиляция, отладка, исполнение.

- **Элементы языка программирования:**

* Лексемы, идентификаторы, операторы.

- **Модели памяти:**

* Кодовая, статическая, стековая, динамическая.



**7. Структура системы программирования. Модели памяти (классы памяти): код, статическая память, стек, динамическая память.**

**Ответ:**

- **Транслятор (компилятор):** преобразует исходный код в машинный код.

- **Редактор исходного кода:** инструмент для написания и редактирования программ.

- **Линковщик (линкер):** объединяет объектные модули в исполняемый файл.

- **Отладчик:** позволяет находить и устранять ошибки в коде.

- **Библиотеки:** набор готовых функций, используемых в разработке.

- **Средства сборки (build tools):** автоматизируют процесс компиляции и сборки.

- **Среда выполнения (runtime):** обеспечивает выполнение программы.

**Код (Code Segment или Text Segment)**

* Содержит исполняемый машинный код программы.
* Эта область памяти является статической и доступной только для чтения.
* Обычно загружается в память в момент запуска программы.

**Особенности:**

* Код неизменяем (Read-Only).
* Включает инструкции, такие как вызовы функций и переходы.

**Динамическая** **память**, называемая также "кучей", **выделяется** явно по запросу программы из ресурсов операционной системы и контролируется указателем. Она не инициализируется автоматически и должна быть явно освобождена.

**Статическая** **память** представляет блок **память** фиксированного размера, **выделение** которой происходит во время компиляции и сборки.

**2. Статическая память (Data Segment)**

* Используется для хранения глобальных переменных, статических переменных, а также констант.
* Данные в этой области инициализируются при запуске программы и остаются неизменными на протяжении её выполнения.

**Особенности:**

* **Инициализированная память:** содержит глобальные переменные с заданными значениями.
* **Неинициализированная память (BSS):** область для переменных, которые инициализируются по умолчанию (обычно нулями).

**3. Стек (Stack Segment)**

* Используется для локальных переменных, параметров функций и управления вызовами функций.
* Работает по принципу **LIFO** (Last In, First Out): последний добавленный элемент удаляется первым.
* Автоматически управляется процессором.

**Особенности:**

* Быстрая работа с данными.
* Память освобождается автоматически при выходе из функции.
* Размер ограничен (может вызвать переполнение стека).

**4. Динамическая память (Heap Segment)**

* Предназначена для выделения памяти во время выполнения программы.
* Управляется программистом с помощью таких функций, как malloc и free (в C) или new и delete (в C++).

**Особенности:**

* Размер неограничен (в пределах доступной памяти).
* Требует явного управления (могут возникать утечки памяти, если память не освобождена).

**8. Парадигмы программирования. Примеры.**

**Ответ:**

Импертивное, декларативное и визуальное.

Императивное делится на: Структурное и Объектно-ориентированное.

Декларативное делится на: Функциональное и Логическое

**9. Язык программирования: определение, назначение, примеры. Исходный код. Объектный код. Объектный модуль. Загрузочный модуль.**

**Ответ:**

Определение:

Язык программирования — это формальный язык, предназначенный для описания алгоритмов и данных, которые может выполнять компьютер. Он включает правила синтаксиса и семантики для написания программ.

**Назначение:**

* Разработка программного обеспечения.
* Управление работой компьютера.
* Обработка данных и автоматизация процессов.

**Примеры языков программирования:**

* **Низкоуровневые:** Ассемблер, машинный код.
* **Высокоуровневые:** Python, C++, Java, JavaScript.

**Основные термины:**

* **Исходный код (Source Code):** текст программы, написанный на языке программирования.
* **Объектный код (Object Code):** промежуточное представление программы после компиляции, содержащее машинные инструкции.
* **Объектный модуль (Object Module):** отдельный файл, содержащий объектный код, готовый для линковки.
* **Загрузочный модуль (Executable Module):** итоговый исполняемый файл, объединяющий объектные модули и готовый для выполнения.

**10. Кодирование информации: определение, назначение, данные, представление данных, кодировки. Примеры.**

**Ответ:**

**Определение:**

Кодирование информации — это процесс представления данных в форме, удобной для хранения, передачи или обработки.

**Назначение:**

* Обеспечение совместимости данных между системами.
* Уменьшение объёма данных.
* Защита информации.

**Данные и представление:**

* **Данные:** числа, текст, изображения, видео и т. д.
* **Представление:** в виде бинарных кодов (0 и 1).

**Кодировки:**

* **ASCII (American Standard Code for Information Interchange):** 7-битная кодировка для английского алфавита и специальных символов.
* **Windows-1251:** кодировка для кириллицы.
* **UTF-8 (Unicode Transformation Format):** универсальная кодировка, совместимая с ASCII, использует переменное количество байтов для символов.

**Пример:**

Символ А в разных кодировках:

* ASCII: отсутствует.
* Windows-1251: 0xC0.
* UTF-8: 0xD090.

**11. Спецификация системы программирования. Кодировка ASCII, кодировка Windows1251, стандарт кодирования символов Unicode.**

**Ответ:**

**Спецификация системы программирования:**

* Набор инструментов и стандартов, обеспечивающих разработку программ (язык программирования, трансляторы, библиотеки).

**Кодировки:**

* **ASCII:** 7-битная кодировка для 128 символов (латиница, цифры, служебные символы).
* **Windows-1251:** 8-битная кодировка для кириллицы (256 символов).
* **Unicode:** стандарт для представления символов всех языков мира. Основные форматы:
  + **UTF-8:** переменное количество байтов (от 1 до 4).
  + **UTF-16:** 16-битная кодировка.
  + **UTF-32:** 32-битная кодировка.

**12. Кодирование информации: определение, назначение, примеры. Стандарт кодирования UNICODE: назначение, структура, UCS, UTF. Прямой (LE) и обратный (BE) порядок байт. BOM: определение, назначение, примеры.**

**Ответ:**

Назначение:

Обеспечивает универсальное кодирование символов для всех языков и систем.

**Структура Unicode:**

* UCS (Universal Character Set): набор всех символов.
* Кодировка: UTF-8, UTF-16, UTF-32.

**Прямой (LE) и обратный (BE) порядок байт:**

* **LE (Little Endian):** младший байт записывается первым.
* **BE (Big Endian):** старший байт записывается первым.

**BOM (Byte Order Mark):**

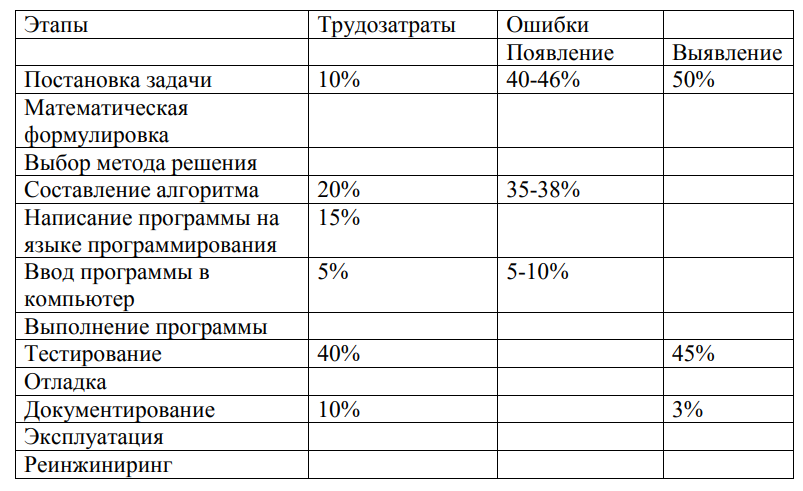
* Определяет порядок байтов (LE или BE) для Unicode файлов.
* Пример BOM для UTF-8: 0xEF 0xBB 0xBF.

**13. Этапы и цели разработки программы, трудоемкость этапов разработки программ.**

**Ответ:**

**Этапы и цели разработки программы:**

1. Постановка задачи.
2. Выбор метода решения.
3. Разработка алгоритма решения задачи.
4. Написание программы на языке программирования (кодирование)
5. Ввод программы в компьютер
6. Трансляция
7. Компоновка
8. Выполнение
9. Отладка
10. Тестирование
11. Документирование
12. Эксплуатация
13. Модификация (Реинжиниринг)
14. Снятие с эксплуатации



**14. Алгоритм программы. Назначение и свойства алгоритмов. Способы описания алгоритмов.**

**Ответ:**

**Алгоритм** – точное предписание, определяющее вычислительный процесс, ведущий от исходных данных к нужному результату.

**Свойства алгоритма:**

Дискректность

Конечность

Понятность

Определенность

Массовость

Эффективность

**Способы описания алгоритма:**

Графический (Блок-схема)

Программный

Псевдокод

Словесно-формульное описание

**15. Системы программирования Microsoft, Linux, Unix, IBM. Стандарты языков программирования. Парадигмы (стили) программирования.**

**Ответ:**

Стили программирования:

Императивное и декларативное.

Императивное делится на структурное и объектно-ориентированное.

Декларативное: функциональное и логическое

**Windows –** семейство закрытых (или проприетарных) операционных систем, разрабатываемых компанией Microsoft. Доступ к системным ресурсам осуществляется с помощью системных функций программного интерфейса или (API - Application Programming Interfase). Функции API содержатся в библиотеках динамической загрузки (Dynamic Link Libraries, или DLL).

**Многозадачность в Windows:**

Процесс – это программа, которая выполняется. При многозадачности такого типа две или более программы могут выполняться параллельно.

Поток – это отдельная часть исполняемого кода (название произошло от понятия «направление протекания процесса»). В многозадачности данного типа отдельные потоки внутри одного процесса также могут выполняться одновременно.

**Система программирования UNIX:**

Unix («UNIX» – зарегистрированной торговой маркой The Open Group) – семейство переносимых, многозадачных и многопользовательских операционных систем, которые основаны на идеях оригинального проекта AT&T Unix, разработанного в 1970-х годах в исследовательском центре Bell Labs Кеном Томпсоном, Деннисом Ритчи и другими.

Особенности UNIX-подобных систем:

1. Unix-подобные системы в отличие от других операционных систем изначально разрабатывались как многопользовательские многозадачные системы. В Unix может одновременно работать сразу много людей, каждый за своим терминалом, при этом каждый из них может выполнять множество различных вычислительных процессов, которые будут использовать ресурсы именно этого компьютера.

2. Unix является мультиплатформенной системой. Ядро системы разработано таким образом, что его легко можно приспособить практически под любой микропроцессор.

**Linux пренадлежит к POSIX-системам**

Стандарт POSIX – переносимый интерфейс операционных систем) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой (системный API), библиотеку языка C и набор приложений и их интерфейсов.

POSIX – это стандарт, описывающий интерфейс между операционной системой и прикладной программой.

Общие принципы POSIX систем:

1) Это многопользовательские системы с разграничением прав по имени пользователя и группы; наличие пользователя root с неограниченными правами.

2) Древовидная файловая система, с единым корнем от /; большинство сущностей системы отображается как имя в дереве файловой системы; функциональное назначение каталогов файловой системы сохраняется примерно постоянным от одной системы к другой.

3) Применение символьных форматов: конфигурация системы и всех программных пакетов представляются в текстовых файлах (например, в файлах XML), это позволяет изменять все конфигурации простым текстовым редактированием.

4) Единообразный набор консольных утилит-команд (стандарт POSIX 2).

5) Единый API программирования языка C.

Преимущества, которые даёт совместимость операционной системы со стандартами POSIX:

− простота переноса (портирование) программных проектов их одной операционной системы в другую: часто это достигается путём выполнения ряда чисто формальных действий;

− простота для программиста-разработчика перехода от одной операционной системы к другой.

**IBM:**

**Стандарты языков программирования:**

Стандарт языка программирования: Visual C++ 2017 версия 15.3 – это реализация стандарта С++17 или ISO/IEC 14882:2017.

ECMAScript — это встраиваемый расширяемый язык программирования, основа для построения других скриптовых языков. Стандартизирован международной организацией Ecma в спецификации ECMA-262. Расширениями языка являются JavaScript (Netscape), JScript (Microsoft) и ActionScript.

ECMAScript — это официальный стандарт языка JavaScript (Слово JavaScript не могло быть использовано, потому что слово Java является торговой маркой компании Sun)

Это восьмое издание спецификации языка ECMAScript.

**16. Интегрированная среда разработки: определение. Примеры IDE. Назначение, основные возможности. Понятие отладки кода на языке программирования.**

**Ответ:**

Интегрированная среда разработки - набор инструментов для разработки и отладки программ, имеющий общую интерактивную графическую оболочку, поддерживающую выполнение всех основных функций жизненного цикла разработки программы.

**Примеры:** JDeveloper, NetBeans, VS, Eclipse.

**Отладчик** – инструментальное средство разработки программ, которое присоединяется к работающему приложению и позволяет проверять код, наблюдать за выполнением исследуемой программы, останавливать и перезапускать её, изменять значения в памяти, просматривать стек вызовов и т.д.

**17. Интегрированная среда разработки MS Visual Studio 20ХХ. Назначение, возможности. Страница свойств проекта.**

**Ответ:**

Это среда для разработки программное обеспечения, имеющая в себе нужный функционал для создания, отладки, сборки и запуска проекта.

На странице свойств проекта в «Свойства конфигурации» есть вкладки «Общие», «Дополнительно», «Отладка», «Каталоги VC++». В первой вкладке можно выбрать **Тип конфигурации (екзешник и т. д.), Стандарт языка С/С++, Имя целевого объекта, Промежуточный каталог, Версия SDK для Windows.**

Во второй вкладке есть **Расширенные свойства, Свойства С++/CLI.** В первой части находятся пункты **Расширение целевого файла, Набор символов(в какой табл. кодировок представ. символы),**  **Файл журнала сборки.**

В третьей вкладке есть **Тип отладчика, Отладка SQL, Рабочий каталог и т.д.**

В четвертой вкладке можно посмотреть и изменить **Каталоги исполняемых файлов, ссылок, библиотек, исходного кода; И посмотреть настройки содержимого общедоступного проекта.**

**18. Среда разработки: назначение и основные возможности отладчика. Точки остановки. Отображение и модификация локальных данных. Пошаговая отладка. Понятие и назначение дизассемблера.**

**Ответ:**

**Отладчик** – инструментальное средство разработки программ, которое присоединяется к работающему приложению и позволяет проверять код, наблюдать за выполнением исследуемой программы, останавливать и перезапускать её, изменять значения в памяти, просматривать стек вызовов и т.д.

**Назначение отладчика** – устранение ошибок в исходном коде программы.

**Отладка** – процесс запуска и выполнения программы в режиме отладки.

Можно управлять отладкой через некоторый шаги с помощью их горячих клавиш (с заходом, с обходом, с выходом). Запуск отладки через горячую клавишу F5 и через панель отладчика.

Точки останова нужны для остановки программы в определенном месте. Чаще всего именно после этого места идет проблемная часть, которую нужно отладить. Они ставятся через панель отладки, щелком левой мыши по левой части поля от кода и нумерации строк, щелчок правой мыши по тому полю и нажать на «Вставить точку останова», горячая клавиша F9.

Локальные данные можно посмотреть в окне «Локальные», где и можно изменить их значение. Конечно значение этих переменных можно поменять и в коде, наведясь на переменную. Эти же переменные есть в окне «Видимые» и, при добавлении, в окне «Контрольные значения».

**Понятие и назначение дизассемблера:**

В окне «Дизассемблированный код» отображается код сборки, соответствующий инструкциям, созданным компилятором.

**Дизассемблер -** это компьютерная программа, которая переводит машинный язык на язык ассемблера - операция, обратная операции ассемблера. Состоит из метки, мнемоники, операнда и комментария.

**19. Методология разработки программного обеспечения. Структурный подход к проектированию ПО. Сущность структурного подхода. Методы структурного программирования**

**Ответ:**

**Методология разработки ПО** представляет собой набор принципов, подходов и практик, которые используются для организации и управления процессом разработки ПО. Методология разработки ПО включет в себя специфичные подходы, такие как Agile, Scrum, Waterfall и другие. Она определяет, как команда разработки организует работу, взаимодействие и управление проектом, а также какие методы и инструменты используются для достижения конечной цели. Методологии разработки ПО могут быть гибкими и адаптивными, позволяя командам быстро реагировать на изменения требований и обратную связь.

**Структурный подход** к программированию представляет собой совокупность рекомендуемых технологических приёмов, охватываю

щих выполнение всех этапов разработки программного обеспечения.

В основе структурного подхода лежит декомпозиция (разбиение на части) сложных систем с целью последующей реализации в виде от-дельных небольших (до 40 – 50 операторов) подпрограмм.

**Структурное программирование** – методология и технология разработки программных средств, основанная на трёх базовых конструкциях:

− следование;

− ветвление;

− цикл.

**20. Методология разработки программного обеспечения. Модульное программирование. Сущность модульного подхода. Методы модульного программирования**

**Ответ:**

**Методология разработки ПО** представляет собой набор принципов, подходов и практик, которые используются для организации и управления процессом разработки ПО. Методология разработки ПО включет в себя специфичные подходы, такие как Agile, Scrum, Waterfall и другие. Она определяет, как команда разработки организует работу, взаимодействие и управление проектом, а также какие методы и инструменты используются для достижения конечной цели. Методологии разработки ПО могут быть гибкими и адаптивными, позволяя командам быстро реагировать на изменения требований и обратную связь.

**Модульное программирование** – это организация программы как совокупности небольших независимых блоков, называемых модулями.

**Модуль** – функционально законченный фрагмент программы, оформленный в виде отдельного файла с исходным кодом.

**Функциональная декомпозиция задачи** – разбиение большой задачи на ряд более мелких, функционально самостоятельных подзадач – модулей.

Технология модульного программирования базируется на следующих методах:

− методы нисходящего проектирования (назначение – декомпозиция большой задачи на меньшие так, чтобы каждую подзадачу можно было рассматривать независимо.);

− методы восходящего проектирования.

**21. Системы контроля версий. Классификация. Назначение, разновидности систем контроля версий.**

**Ответ:**

**Система управления версиями (от англ. Version Control System, VCS или Revision Control System, RCS)** – программное обеспечение для облегчения работы с изменяющейся информацией и разработки проекта совместно с коллегами.

**Разновидности:**

- локальные системы контроля версий (Version Control System, VCS, Revision Control System, RCS);

- централизованные системы контроля версий (Centralized Version Control System, CVCS);

- распределенные системы контроля версий (Distributed Version Control System, DVCS).

**Назначение:**

­- автоматическое создание архива (бэкап) для синхронизации кодовой базы;

- отслеживание изменений (кто, когда и зачем сделал изменения);

- совместная работа над одним и тем же проектом;

- отслеживание ошибок (Bug трекинговая система).

**22. Три состояния файлов в Git. Сохранение изменений в репозитории. Основные определения и основные команды Git.**

**Ответ:**

**Три состояния файлов:**

- модифицированное (изменения уже внесены в файл, но пока не зафиксированы в базе данных)

- индексированное (текущая версия модифицированного файла помечена как требующая последующей фиксации)

- зафиксированное (данные надежно сохранены в локальной базе)

**Сохранить изменения в репозиторий** можно с помощью команд git add, got commit -m.

Локальный репозиторий – место для хранения данных на вашем компьютере.

Коммит – изменение каких-либо данных, сохраненных в репозитории.

Ветвление (branching) означает отклонение от основной линии разработки, после которого работа перестает затрагивать основную линию и переходит в ветвь.

HEAD указывает на ветку.

Merge – слияние веток.

Конфликт – ошибка при слиянии веток.

Git - распределённая система управления версиями.

**23. Ветвления в Git. Слияния веток. Конфликты при слиянии веток. Ветвление проектов. Распределенная разработка.**

**Ответ:**

Ветвление (branching) означает отклонение от основной линии разработки, после которого работа перестает затрагивать основную линию и переходит в ветвь.

Merge – слияние веток.

Конфликт – ошибка при слиянии веток. Если откроем, например, текстовый файл, из-за которого конфликт, то там буду помечены различия между фалом из одной ветки и из другой. Конфликт этот решается вручную разработчиком.

**Распределенная разработка -** это одна из высших форм совместной работы в любой инженерной или научной среде. Требует сильных лидерских качеств и хорошей работы в команде.

**24. Понятие веб-сервиса Github. Отличие Git и GitHub. Назначение и основные возможности GitHub. Совместная работа над проектом.**

**Ответ:**

**Git** - распределённая система управления версиями.

**GitHub** – сервис онлайн-хостинга репозиториев, обладающий всеми функциями распределённого контроля версий и функциональностью управления исходным кодом – всё, что поддерживает **Git**.

**Чем отличается Git и GitHub**

**Git:** инструмент, позволяющий реализовать распределённую систему контроля версий.

**GitHub:** сервис для проектов, использующих Git.

**Назначение и основные возможности GitHub:**

Место для хранение удаленных репозиториев, а также хостинг для совместной работы над проектами на удаленке. Он использует программное обеспечение Git, обеспечивающее распределенный контроль версий Git плюс контроль доступа, отслеживание ошибок, запросы программных функций, управление задачами, непрерывную интеграцию и wiki для каждого проекта.

**25. Этапы создания программного продукта. Понятие жизненного цикла разработки программного обеспечения. Назначение модели жизненного цикла ПО. Структура процессов жизненного цикла программного обеспечения.**

**Ответ:**

**Этапы создания программного продукта:**

постановку задачи

разработку интерфейса пользователя

разработку алгоритма работы программного средства

написание программы на специальном языке

подготовку наборов данных, независимых от пользователя

преобразование текста программы и наборов данных в машинные коды

отладку программы

сопровождение программного средства

**Назначение модели жизненного цикла ПО:**

- дает рекомендации по организации процесса разработки ПО в целом, конкретизируя его до видов деятельности, артефактов, ролей и их взаимосвязей

- служит основой для планирования программного проекта

- способствует правильному распределению обязанностей сотрудников

**Структура процессов жизненного цикла программного обеспечения:**

Основные процессы:

Приобретение;

Поставка;

Разработка;

Эксплуатация;

сопровождение.

Организационные процессы:

Управление;

Усовершенствование;

создание инфраструктуры;

обучение.

Вспомогательные процессы:

документирование

управление конфигурацией

обеспечение качества

верификация

аттестация

совместная оценка

аудит

разрешение проблем

**Жизненный цикл разработки программного обеспечения** – это период времени, который начинается с момента принятия решения о необходимости создания ПО и заканчивается в момент полного его изъятия из эксплуатации.

**26. Каскадная модель жизненного цикла ПС: содержание этапов, область применения, достоинства и недостатки.**

**Ответ:**

**Содержание этапов:**

1 **Сбор и анализ требований:**

* Определение функциональных и нефункциональных требований.
* Подготовка технического задания.

2 **Проектирование:**

* Разработка архитектуры системы.
* Подготовка детального технического проекта.

3 **Реализация (кодирование):**

* Написание исходного кода программы на основе проектной документации.
* Трансляция и сборка кода в исполняемый файл.

4 **Тестирование:**

* Проверка системы на соответствие требованиям.
* Выявление и исправление ошибок.

5 **Ввод в эксплуатацию:**

* Установка ПО на рабочих местах пользователей.
* Обучение персонала.

6 **Сопровождение:**

* Исправление обнаруженных ошибок.
* Добавление новых функций и адаптация к изменяющимся условиям.

**Достоинства:**

1. **Простота и логичность:**
   * Лёгкость управления проектом благодаря строгой последовательности этапов.
2. **Подробная документация:**
   * Каждая фаза сопровождается подробной документацией, что облегчает поддержку.
3. **Контроль:**
   * Простота контроля выполнения проекта через завершение этапов.
4. **Подходит для небольших проектов:**
   * Эффективна для проектов с минимальной изменчивостью требований.

**Недостатки:**

1. **Низкая гибкость:**
   * Изменение требований на поздних стадиях приводит к большим затратам.
2. **Длительное время ожидания:**
   * Пользователь получает готовый продукт только в конце разработки.
3. **Ограниченная обратная связь:**
   * Тестирование проводится после реализации, что увеличивает риск выявления критических ошибок на поздних стадиях.
4. **Неэффективность для крупных и сложных проектов:**
   * При высокой изменчивости требований возникают дополнительные затраты и задержки.

**Область применения:**

* Проекты с чётко определёнными требованиями и стабильной архитектурой.
* Разработка ПО с минимальным риском изменения требований.
* Государственные или регулируемые проекты, требующие строгой документации.

**Каскадная модель** (*waterfall model*, иногда переводят как **модель «Водопад»**) — модель процесса [разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), в которой процесс разработки выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, интеграции и поддержки.

**27. Эволюционная модель жизненного цикла ПС: последовательность действий, область применения, достоинства и недостатки.**

**Ответ:**

Эта модель основана на следующей идее: разрабатывается первоначальная версия программного продукта, которая передается на испытание пользователям, затем она дорабатывается с учетом мнения пользователей, получается промежуточная версия продукта, которая также проходит " испытание пользователем", снова дорабатывается и так несколько раз, пока не будет получен необходимый программный продукт.

**28. Спиральная модель разработки ПО: содержание этапов создания ПС, область применения, достоинства и недостатки.**

**Ответ:**

**Спиральная модель:**

− заказчик и команда разработчиков серьёзно анализируют риски проекта и выполняют его итерациями. Последующая стадия основывается на предыдущей;

− в конце каждого витка (цикла разработки) итераций принимается решение, продолжать ли проект.

**преимущества:**

− большое внимание уделяется проработке рисков.

**недостатки:**

− есть риск застрять на начальном этапе: бесконечно совершенствовать первую версию продукта и не продвинуться к следующим;

− разработка длится долго и стоит дорого.

**область применения:**

− для решения критически важных бизнес-задач.

**29. Инкрементальная модель разработки ПО. Развитие инкрементального подхода. XPпроцессы.**

**Ответ:**

**Инкрементная модель:**

Метод, в котором ПО проектируется, реализуется и тестируется инкрементно (каждый раз с небольшими добавлениями) до самого окончания разработки;

**преимущества:**

− быстрый выпуск минимального продукта;

− ошибка обходится дешевле;

− постоянное тестирование пользователями.

**недостатки:**

− требуется переработка проекта;

− отсутствие фиксированного бюджета и сроков.

**область применения:**

− когда основные требования к системе четко определены и понятны, некоторые детали могут дорабатываться с течением времени (поэтапно);

− требуется ранний вывод продукта на рынок;

− при разработке веб-приложений и продуктов компаний-брендов.

**30. Методологии разработки программного обеспечения. Инструментарий технологии программирования. Управление требованиями. Техническое задание на разработку программного продукта.**

**Ответ:**

**Методологии разработки программного обеспечения**

Методология разработки ПО — это структурированный подход к организации процесса создания программного обеспечения. Она определяет порядок выполнения задач, взаимодействие участников проекта, а также применяемые инструменты.

**Основные методологии:**

1. **Каскадная модель (Waterfall):**
   * Линейная последовательность этапов (анализ, проектирование, реализация, тестирование, внедрение, сопровождение).
   * Подходит для проектов с чётко фиксированными требованиями.
2. **Итеративная модель:**
   * Разработка осуществляется в циклах, где каждая итерация добавляет новую функциональность.
   * Удобна для проектов с неопределёнными требованиями.
3. **Инкрементальная модель:**
   * ПО создаётся поэтапно, причём каждая новая версия добавляет функционал.
   * Снижает риски, позволяет пользователям оценивать промежуточные версии.
4. **Гибкие методологии (Agile):**
   * Включают Scrum, Kanban, Extreme Programming (XP).
   * Основываются на итеративном подходе, обратной связи и гибкости в изменении требований.
   * Scrum: акцент на спринты, бэклог и ежедневные встречи.
   * Kanban: визуализация процесса на доске и управление потоками задач.
5. **Модели RAD (Rapid Application Development):**
   * Быстрая разработка через прототипы и итерации.
   * Ускоряет процесс, но требует активного участия заказчика.
6. **Спиральная модель:**
   * Комбинирует итеративный подход с управлением рисками.
   * Используется для сложных и рискованных проектов.

**Инструментарий технологии программирования**

Совокупность инструментов и средств, используемых на разных этапах разработки ПО.

**Категории инструментов:**

1. **Средства управления проектами:**
   * Jira, Trello, Asana — для планирования задач, управления временем и ресурсами.
2. **Интегрированные среды разработки (IDE):**
   * Visual Studio, IntelliJ IDEA, Eclipse — для написания и отладки кода.
3. **Системы контроля версий:**
   * Git, GitHub, GitLab — для хранения и управления изменениями исходного кода.
4. **Инструменты тестирования:**
   * Selenium, JUnit, Postman — для автоматизированного и ручного тестирования.
5. **Средства проектирования:**
   * UML-диаграммы, ERD (Entity-Relationship Diagram).
6. **Средства документирования:**
   * Confluence, Doxygen — для создания и хранения проектной документации.

**Управление требованиями**

**Определение:**

Управление требованиями — это процесс сбора, анализа, документирования и контроля за выполнением требований, предъявляемых к программному продукту.

**Этапы управления требованиями:**

1. **Сбор требований:**
   * Интервью с заказчиком, анкетирование, анализ рынка.
2. **Анализ и уточнение:**
   * Выявление противоречий, приоритизация требований.
3. **Документирование:**
   * Создание спецификаций требований (SRS — Software Requirements Specification).
4. **Управление изменениями:**
   * Отслеживание изменений в требованиях и их влияние на проект.

**Классификация требований:**

* **Функциональные:** описывают, что должно делать ПО.
* **Нефункциональные:** описывают качество (производительность, безопасность, масштабируемость).
* **Бизнес-требования:** связаны с целями заказчика.

**Техническое задание (ТЗ) на разработку программного продукта**

**Определение:**

ТЗ — это документ, который описывает требования к функциональности, характеристикам и этапам разработки программного продукта.

**Основные разделы ТЗ:**

1. **Общие сведения:**
   * Название проекта, цели и задачи.
2. **Требования к функциональности:**
   * Перечень функций и их описание.
3. **Технические требования:**
   * Операционная система, оборудование, производительность.
4. **Требования к интерфейсу:**
   * Дизайн, удобство использования.
5. **Тестирование:**
   * Методы и критерии оценки качества.
6. **Этапы разработки:**
   * График выполнения работ.
7. **Приложения:**
   * Диаграммы, спецификации, макеты.

**Роль ТЗ:**

* Обеспечивает единое понимание между разработчиками и заказчиками.
* Является основой для управления проектом.
* Используется для контроля и тестирования продукта.

**31. Методологии быстрой разработки ПО. Жизненный цикл ПО по методологии RAD. Преимущества, недостатки, область применения.**

**Ответ:**

**Жизненный цикл ПО по методологии RAD состоит из фаз:**

Анализ

Проектирование

Реализация

Внедрение

Сопровождение

**RAD:**

− использование фокус-групп для сбора требований;

− прототипирование и пользовательское тестирование

− повторное использование программных компонентов;

− использование плана, не включающего переработку, или дизайн следующей версии продукта;

− проведение неформальных совещаний по запросу одной из сторон.

**преимущества:**

− разработка ввыполняется быстро и дешево;

− обеспечивается приемлемый для пользователя уровень качества;

− пользователь может оперативно внести изменения в проект;

− функциональность, которая нужна заказчику «еще вчера», можно разработать в первую очередь, и использовать, даже если остальные части программы еще не готовы.

**недостатки:**

− RAD применима для небольших команд разработчиков;

− RAD зависит от степени участия заказчика в работе проекта.

**область применения:**

− для проектов, которые легко разделить на независимые или слабосвязанные модули;

− если требования к программному обеспечению быстро меняются;

− в условиях ограниченного бюджета;

− нет ясного представления, как должен выглядеть и работать продукт;

− разработка ведется командой профессионалов;

− если пользователь готов активно участвовать в проекте на протяжении всей работы.

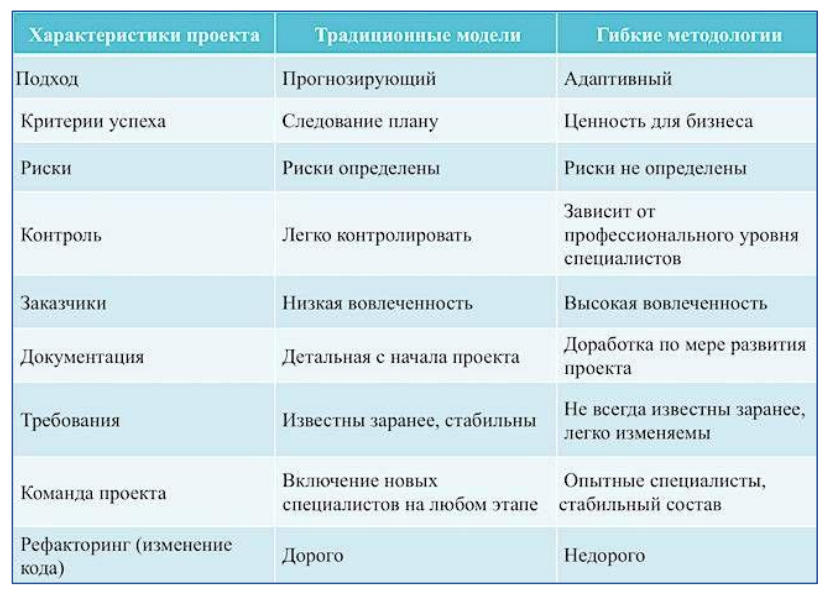
**32. Инкрементальная модель разработки ПО. Итерационная модель разработки ПО. Отличие итерационной модели от инкрементной модели.**

**Ответ:**

Отличие этих моделей заключается в том, что в инкрементной модели возможности программы добавляются и тестируются постепенно, а в итерационной модели к тестированию мы имеем уже рабочий, но не законченный продукт, который уже постепенно доделывается.

**33. Методологии гибкой разработки ПО. Различия между Agile и традиционным подходом к разработке ПО. Преимущества и недостатки технологий быстрой разработки программного обеспечения.**

**Ответ:**



**Преимущества быстрой разработки:**

− разработка выполняется быстро и дешево;

− обеспечивается приемлемый для пользователя уровень качества;

− пользователь может оперативно внести изменения в проект;

− функциональность, которая нужна заказчику «еще вчера», можно разработать в первую очередь, и использовать, даже если остальные части программы еще не готовы.

**Недостатки быстрой разработки:**

− RAD применима для небольших команд разработчиков;

− RAD зависит от степени участия заказчика в работе проекта.

**34. Модель компетентного разработчика (Personal Software Process).**

**Ответ:**

P**ersonal Software Process** определяет требования к компетенциям разработчика

**Цели PSP**

PSP помогает разработчикам:

− улучшить оценку и планирование навыков;

− управлять качеством проектов;

− снизить количество ошибок в своих разработках.

Один из основных аспектов PSP — использование накопленной статистики для анализа и улучшения показателей процесса разработки. Сбор статистики включает 4 элемента:

− скрипты.

− оценки.

Включают 4 основных элемента:

• размер — оценка размера для части продукта. Например, количество строк кода (LOC — Lines Of Code).

• качество — количество ошибок в продукте.

• усилия — оценка времени, требующегося для завершения задачи, обычно записываемое в минутах.

• планирование — оценка хода проекта, перемещаемая между планируемыми и завершенными пунктами.

− стандарты кодирования. Применение стандартов к процессу может обеспечить точные и постоянные данные.

− формы.

**35. Этапы конструирования. Подходы к конструированию программных средств.**

**Ответ:**

**Конструирование программного обеспечения —** дисциплина [программной инженерии](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%B0%D1%8F_%D0%B8%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D0%B8%D1%8F). Это детальное создание работающего [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) посредством комбинации [кодирования](https://en.wikipedia.org/wiki/Software_construction#Coding), [верификации](https://en.wikipedia.org/wiki/Software_verification), [модульного тестирования](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [интеграционного тестирования](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [отладки](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B).

**Минимизация сложности**

Уменьшение [сложности](https://en.wikipedia.org/wiki/Programming_complexity) достигается через акцентирование создания [кода](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), который был бы простым и читаемым, а не искусным. Минимизация [сложности](https://en.wikipedia.org/wiki/Programming_complexity) получается путём использования [стандартов](https://en.wikipedia.org/wiki/Software_construction#Standards_in_construction) и посредством многочисленных специфических методов [написания кода](https://en.wikipedia.org/wiki/Software_construction#Coding). Также она поддерживается методами [конструктивно-ориентированного качества](https://en.wikipedia.org/wiki/Software_construction#Construction_quality).[[1]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F#cite_note-SWEBOK-1)

**Предусматривание изменений**

Предвосхищение изменений поддерживается многими специфическими методами [набора кода](https://en.wikipedia.org/wiki/Software_construction#Coding):[[1]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F#cite_note-SWEBOK-1)

* Способы общения: такие как стандарты форматов документов и содержимого.
* [Языки программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)
* Платформы
* Инструменты: такие как схематические стандарты для нотаций, например, [язык UML](https://ru.wikipedia.org/wiki/UML).

**Конструирование для верификации**

Конструирование для [верификации](https://en.wikipedia.org/wiki/Software_verification) означает построение [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) таким образом, что ошибки могут быть легко обнаружены [инженером](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%B0%D1%8F_%D0%B8%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D0%B8%D1%8F) пишущим это [ПО](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), а также в ходе независимого [тестирования](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) и эксплуатационной деятельности. Специфические методы, которые поддерживают конструирование для верификации включают в себя следующие: стандарты написания кода для поддержки [проверки кода](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D0%BC%D0%BE%D1%82%D1%80_%D0%BA%D0%BE%D0%B4%D0%B0), [модульное тестирование](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), организация [код](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) для поддержки [автоматизированного тестирования](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%B7%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), ограниченное использование сложных или труднодоступных для понимания [языковых](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) структур и некоторые другие.[[1]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F#cite_note-SWEBOK-1)

**Стандарты в конструировании**

Стандарты, которые непосредственно затрагивают вопросы конструирования, включают:[[1]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F#cite_note-SWEBOK-1)

* Использование внешних стандартов: стандарты для [языков конструирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), инструменты конструирования, технические интерфейсы и взаимодействие между конструированием ПО и другими дисциплинами.
* Использование внутренних стандартов: [минимизация сложности](https://en.wikipedia.org/wiki/Software_construction#Minimizing_complexity), [предвидение изменений](https://en.wikipedia.org/wiki/Software_construction#Anticipating_change) и [конструирование для верификации](https://en.wikipedia.org/wiki/Software_construction#Constructing_for_verification).

**???? 36. Технологии разработки ПО. Управление требованиями. Понятие требования к ПО. Виды и уровни требований, классификация требований.**

**Ответ:**

**Требование** – это утверждение, которое идентифицирует эксплуатационные, функциональные параметры, характеристики или ограничения проектирования продукта или процесса, которое однозначно, проверяемо и измеримо.

Свойства требований:

− корректность (correct);

− однозначность (unambiguous);

− полнота (complete);

− непротиворечивость (consistent);

− приоритезация (prioritized);

− проверяемость (verifiable);

− модифицируемость (modifiable);

− отслеживаемость (traceable).

**Классификация требований:**

1. Функциональные:

- Бизнес требования

- Требования пользователей

- Системные требования

2. Нефункциональные

- Бизнес требования

- Требования пользователей

- Системные требования

https://ru.wikipedia.org/wiki/Требования\_к\_программному\_обеспечению#Виды\_требований\_по\_уровням

**37. Формализация функциональных требований: диаграммы вариантов использования. Назначение и компоненты диаграмм вариантов использования. Примеры. (14 лекция прочитать)**

**Ответ:**

****

**Диаграмма вариантов** использования позволяет наглядно представить ожидаемое поведение системы. Основными понятиями диаграмм вариантов использования являются: действующее лицо, вариант использования и связь.

38.**Тестирование ПО: основные понятия и определения. Классификация видов тестирования. Цели, задачи и принципы тестирования.** (15 лекция)

**Ответ:**

**Тестирование программного обеспечения (Software Testing)** – проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. **[IEEE Guide to Software Engineering Body of Knowledge, SWEBOK, 2004].**

Цели тестирования

• убедиться, что ПО отвечает заявленным требованиям.

• выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим требованиям.

Задачи тестирования

• убедиться, что ПО отвечает заявленным требованиям.

• выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим требованиям.

• предотвратить как можно больше дефектов

• проверить, что известные дефекты устранены

• проверить, что при устранении известных дефектов, не было внесены новые дефекты

• информировать всех заинтересованных лиц о качестве системы.

Принципы тестирования

1. Тестирование демонстрирует наличие дефектов.

2. Исчерпывающее тестирование недостижимо.

3. Раннее тестирование.

4. Скопление дефектов.

5. Парадокс пестицида.

6. Тестирование зависит от контекста.

7. Заблуждение, что ошибки отсутствуют.

**39. Цикл тестирования ПО. Тестирование по позитивному и негативному признаку.**

**Ответ:**

Тестирование по признаку позитивности:

**Позитивное тестирование** – это тестирование на тестовых данных или сценариях, которые соответствуют ожидаемому (штатному, нормальному) поведению системы согласно техническим требованиям и документации.

**Негативное тестирование** – это тестирование на тестовых данных или сценариях, которые соответствуют внештатному поведению тестируемой системы. Это могут быть исключительные ситуации (ошибки) или неверные данные.

**Цикл тестирования:**

- Планирование тестирования

- Разработка тестов

- Выполнение тестов

- Анализ и отчет о результатах тестирования

**40. Тестирование ПО: методы тестирования. Ручное тестирование.**

**Отчет:**

**Методы тестирование:**

Функциональное тестирование рассматривает заранее указанное поведение и основывается на анализе функциональных требований.

Тестирование безопасности проверяет безопасность системы, а также выполняется для анализа рисков, связанных с защитой приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.

Тестирование взаимодействия оценивает способность приложения взаимодействовать с одним и более компонентами или системами.

Нефункциональные виды тестирования:

Тестирование производительности – определение масштабируемости приложения под нагрузкой. Виды тестирования производительности:

• нагрузочное тестирование (Performance and Load Testing);

• стрессовое тестирование (Stress Testing);

• тестирование стабильности или надежности (Stability / Reliability Testing);

• объемное тестирование (Volume Testing).

Тестирование установки направленно на проверку успешной инсталляции и настройки, а также обновления или удаления программного обеспечения.

Тестирование удобства пользования – это метод тестирования, направленный на установление степени удобства использования, обучаемости, понятности и привлекательности для пользователей разрабатываемого продукта в контексте заданных условий. [ISO 9126]

Тестирование на отказ и восстановление (Failover and Recovery Testing) проверяет тестируемое ПО на способность противостоять и успешно восстанавливаться после возможных сбоев (ошибки программного обеспечения, отказы оборудования или проблемы связи).

Конфигурационное тестирование (Configuration Testing) направлено на проверку работы ПО при различных конфигурациях системы (заявленных платформах, поддерживаемых драйверах, при различных конфигурациях компьютеров и т.д.).

Тестирование, связанное с изменениями:

Дымовое тестирование – минимальный набор тестов, проверяющих базовую функциональность (нового или исправленного ПО).

Регрессионное тестирование – это вид тестирования направленный на проверку изменений, сделанных в приложении или окружающей среде (починка дефекта, слияние кода, миграция на другую операционную систему, базу данных, веб сервер или сервер приложения), для подтверждения того факта, что существующая ранее функциональность работает как и прежде (используются тест кейсы, написанные на ранних стадиях разработки и тестирования).

Тестирование сборки (Build Verification Test) – это тестирование, направленное на определение соответствия, выпущенной версии, критериям качества для начала тестирования.

Санитарное тестирование или проверка согласованности/исправности (Sanity Testing) – это узконаправленное тестирование для доказательства того, что конкретная функция работает согласно заявленным в спецификации требованиям (подмножество регрессионного тестирования). Используется для определения работоспособности определенной части приложения после изменений произведенных в ней или окружающей среде. Обычно выполняется вручную.

**Ручное тестирование** – все тестирование проводится тестировщиком вручную без помощи скриптов.

**41. Разработка программной документации. Назначение документирования программного обеспечения. Стандарты документирования.**

**Отчет:**

**Документация** – это рабочее пространство проекта.

- документирование позволяет четко разграничить зоны ответственности между участниками проекта;

- документы избавляют от ненужных конфликтов;

- позволяет в любой момент времени быстро найти нужную информацию и понять, как решать конкретные задачи;

- в документах четко прописано, кто что делает, кто за что отвечает, как работает система и что делать, если что-то пошло не так;

- команда говорит «на одном языке»;

- только тщательно описанные требования могут быть проверены на полноту и непротиворечивость.

**Стандарт документирования:**

Аудитория (audience): категория пользователей, предъявляющих к документации одинаковые или аналогичные требования, определяющие содержание, структуру и назначение данной документации.

Документатор (documenter): сторона, создающая документацию.

Справочный текст (help text): текст, автоматически выбираемый в зависимости от контекста, в котором он вызывается, облегчающий и убыстряющий при эксплуатации ПС поиск содержащихся в издании объектов. Справочный текст контекстно зависим.

Диалоговая документация (on-line documentation): информация, доступная пользователю при эксплуатации ПС, которая необязательно привязана к конкретному контексту.

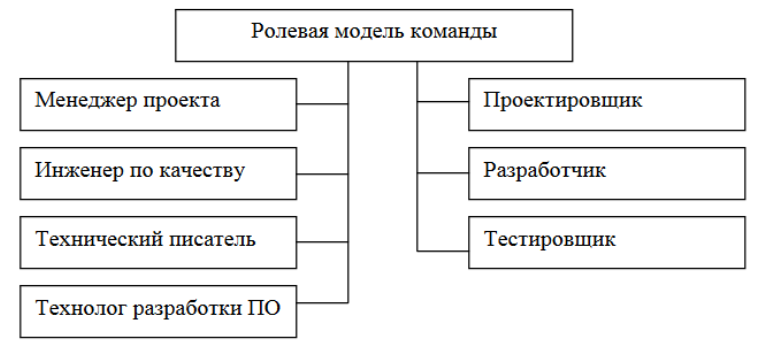
Система диалоговой информации или справочная система (on-line documentation system or help вsystem): часть программы или отдельная программа, запрашиваемая пользователем и позволяющая ему просматривать части диалоговой документации или справочного текста.

Продукт (программный продукт, product): полный набор компьютерных программ, процедур и соответствующих им документации и данных, предназначенный для поставки пользователю.

Тестирование на практичность (usability testing): формальный процесс оценки соответствия документации установленным требованиям.

**42. Управление командой проекта. Ролевая модель команды. Функции каждого из участников.**

**Отчет:**

****

**Функции каждого из участников:**

− менеджер проекта – главное действующее лицо:

o подбор и управление кадрами; o подготовка и исполнение плана проекта;

o руководство командой;

o обеспечение связи между подразделениями;

o обеспечение готовности продукта.

− проектировщик – проектирование архитектуры высокого уровня и контроль ее выполнения. В небольших командах функция распределяется между менеджером и разработчиками. В больших проектах это может быть целый отдел.

− разработчик – роль, ответственная за непосредственное создание конечного продукта.

− тестировщик – роль, ответственная за удовлетворение требований к продукту (функциональных и нефункциональных).

− инженер по качеству. Обеспечивает три уровня качества:

o качество конечного продукта – обеспечивается тестированием;

o качество процесса разработки (тезис: для повышения качества продукта надо повысит качество процесса разработки);

o качество (уровень) организации работ (тезис: для повышения качества процесса надо повысить качество организации работ).

− технический писатель или разработчик пользовательской (и иной) документации как части программного продукта (включая организацию тестирования документации).

− технолог разработки ПО обеспечивает поддержку модели ЖЦ, создание и сопровождение среды сборки продукта, управление исходными текстами.

**43. Международные организации стандартизации. Объекты стандартизации.**

**Отчет:**

**Стандартизация:** - принятие соглашения по спецификации, производству и использованию аппаратных и программных средств вычислительной техники; - установление и применение стандартов, норм, правил и т.п.

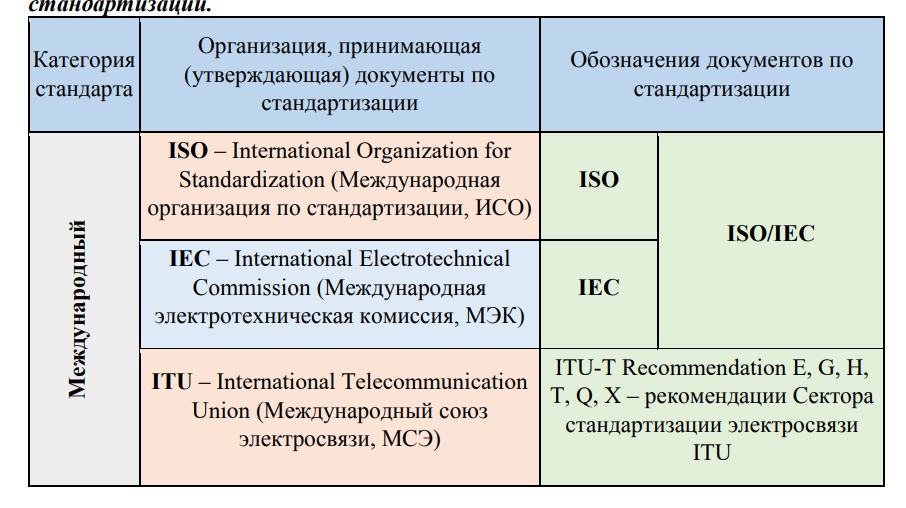
**Кодекс этики программной инженерии (краткая версия)**

Разработчики:

• ACM (Association for Computing Machinery – Ассоциация по вычислительной технике);

• IEEE (Institute of Electrical and Electronic Engineers – Институт инженеров по электротехнике и электронике);

• BCS (British Computer Society – Британское компьютерное общество).



Объекты стандартизации в программной инженерии

- процессы разработки ПО

- продукты разработки

- ресурсы, которые используют процессы для создания программного продукта.

**44. Принципы Agile. Манифест Agile. Управление проектами.**

**Ответ:**

**Включает в себя:**

1. экстремальное программирование (Extreme Programming, XP);

2. бережливую разработку программного обеспечения (Lean);

3. фреймворк для управления проектами Scrum;

4. разработку, управляемую функциональностью (Feature-driven development, FDD);

5. разработку через тестирование (Testdriven development, TDD);

6. методологию «чистой комнаты» (Cleanroom Software Engineering);

7. итеративно-инкрементальный метод разработки (OpenUP);

8. методологию разработки Microsoft Solutions Framework (MSF);

9. метод разработки динамических систем (Dynamic Systems Development Method, DSDM);

10. метод управления разработкой Kanban.

**Манифест Agile** - Манифест гибкой [**разработки программного обеспечения**](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Agile Manifesto*) — основной документ, содержащий описание ценностей и принципов [гибкой разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%B1%D0%BA%D0%B0%D1%8F_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8), разработанный в [феврале 2001 года](https://ru.wikipedia.org/wiki/2001_%D0%B3%D0%BE%D0%B4#%D0%A4%D0%B5%D0%B2%D1%80%D0%B0%D0%BB%D1%8C)[[1]](https://ru.wikipedia.org/wiki/Agile_Manifesto#cite_note-History-1) на встрече 17 независимых практиков нескольких методик программирования, именующих себя «[Agile Alliance](http://www.agilealliance.org/)».

**45. Схема процесса работы по методологии Scrum. Планирование и длительность спринта в Scrum. Бэклог продукта и пользовательские истории. Основные Scrum встречи.**

**Ответ:**

**Scrum кратко**: agile-подход к разработке и управлению проектами:

− деление работы на части, которые называются спринтами (две недели);

− спринты планируются исходя из требований для данного момента;

− относительная оценка времени выполнения работ;

− ревью каждого спринта, чтобы понять, как он прошёл и что можно было бы улучшить;

− фидбек (обратная связь) по поставляемому продукту;

− ежедневные собрания (15 мин.).

**Бэклог продукта (Product backlog)** представляет собой упорядоченный по степени важности список требований, предъявляемых к разрабатываемому продукту.

Элементы этого списка называются **Пользовательскими историями** **(User story).**

Шаблон пользовательской истории: Как , Я хочу , Для того, чтобы.

**Владелец продукта (Produсt Owner)** – преподаватель.

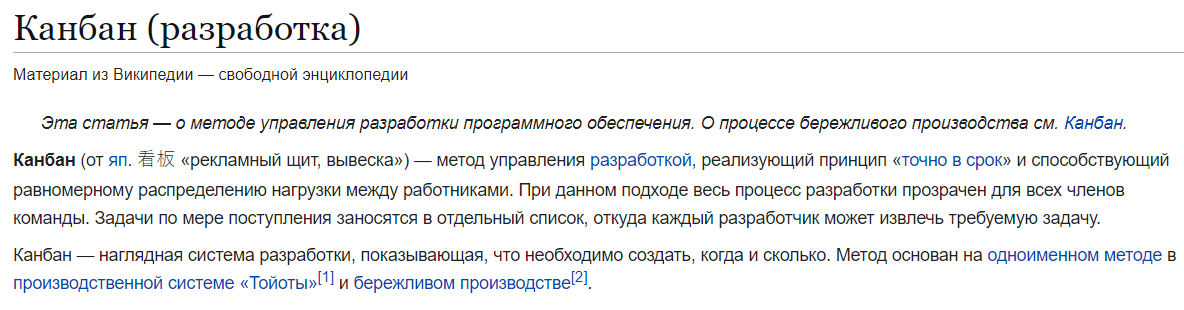
**Скрам-мастер (Scrum master)** – активный, заинтересованный студент, который координирует процесс, проводит ежедневные собрания (Scrum Meetings).

**Скрам-команда (Scrum team)** – команда из 5 человек, работающая над реализацией проекта.

**Начальные требования к проекту/модулю** предоставляются в виде списка задач - Бэклог (backlog), из которого формируется Бэклог спринта (sprint backlog).

**46. Разработка проекта по методологии Kanban. Назначение доски Kanban. Отличие методологии Scrum от Kanban.**

**Ответ:**

****

**Канбан-доска:** карточки перемещаются по колонкам-этапам. Легко контролировать то, что сделано, что нет, а что в процессе.

**Основная разница между Kanban и Scrum в том, что** при использовании второго метода, спринты идут по 1-2 недели, а при использовании первого метода задачи можно давать хоть каждый день. Как раз из этого и выходит наличие канбан-доски.

**47. Язык программирования: определение, назначение, примеры. Схема создания исполняемого файла для компилируемых языков программирования. Стиль оформления кода.**

**Ответ:**

**Язы́к программи́рования** — [формальный язык](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA), предназначенный для записи [компьютерных программ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0).

В языке используются команды, которые понятны компьютеру и которые в конечно итоге в ходе компиляции переходят в машинный код для исполнения.

Стиль оформления кода **читать в лекции 7.**

**Схема создания исполняемого файла для компилируемых языков программирования:**

- препроцессинг

- компиляция

- компоновка объектных модулей

**48. Кодирование информации. Кодировка UNICODE: назначение, структура. Маркер BOM: определение, назначение, примеры.**

**Ответ:**

Стандарт предложен в 1991 году некоммерческой организацией Unicode Consortium, стандарт ISO/IEC 10646:2020.

**Юникод** – стандарт кодирования символов, позволяющий представить знаки почти всех письменных языков, состоит из 2х разделов:

− **UCS** – universal character set (универсальный набор символов);

− **UTF** – Unicode transformation format (семейство кодировок).

Принято обозначение символа U+xxx, где xxx- число в шестнадцатеричном формате.

• **UNICODE**:

− UCS расположены в 17 плоскостях (0-16);

− в каждой плоскости 2^16d

(65 536) символов;

− плоскость 0 – основная (основные символы);

− 1-14 – дополнительные;

− 15-16 – для частного использования.

**Маркер последовательности байтов UNICODE:** BOM (Byte Order Mark) Для определения формата представления Юникода в начало текстового файла записывается сигнатура (обозначение) — символ U+FEFF — маркер последовательности байтов.

**49. Система контроля версий Git: основные возможности. Отличия Git от других систем контроля версий.**

**Ответ:**

**Отличия Git от других систем контроля версий:**

1) Хранит снимки состояний, а не изменений

2) Локальность операций

3) Целостность Git (вычисление контрольных сумм – хеш SHA-1)

**Возможности:**

− 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.

**50. Технологии разработки ПО. Управление требованиями. Функциональные требования и нефункциональные требования. Разработка требований.**

**Ответы:**

**Управление требованиями** процесс, включающий:

- идентификацию, выявление, документацию, анализ, отслеживание, приоритизацию требований, достижение соглашений по требованиям и затем управление изменениями и уведомление заинтересованных лиц.

**Функциональные требования** определяют функции, которые выполняет система, и зависят от потребностей пользователей и типа решаемой задачи.

**Нефункциональные требования** определяют характеристики и ограничения системы и не связаны непосредственно с функциональными требованиями. Они формируются на основе имеющихся атрибутов качества, требований к внешнему интерфейсу и ограничений.