# **Вопрос 1 (Цели и задачи технологий разработки ПО. Особенности современных проектов разработки ПО)**

В конце 60-х - начале 70-х годов появились первые признаки кризиса в области программирования - колоссальные успехи в области развития средств вычислительной техники пришли в противоречие с низкой производительностью труда программистов и низкими темпами ее роста. В связи с усложнением бизнеса, усложнением программных систем стало очевидным, что их трудно проектировать, кодировать, тестировать и особенно трудно понимать, когда возникает необходимость их модификации в процессе сопровождения. Появилась жизненная потребность в создании технологии разработки программных средств и инженерных методов их проектирования для существенного улучшения производительности труда разработчиков.

Современные крупные проекты ИС характеризуются, как правило, следующими особенностями:

- сложность описания (достаточно большое количество функций, процессов, элементов данных и сложные взаимосвязи между ними), требующая тщательного моделирования и анализа данных и процессов;

- наличие совокупности тесно взаимодействующих компонентов (подсистем), имеющих свои локальные задачи и цели функционирования;

- отсутствие прямых аналогов, ограничивающее возможность использования каких-либо типовых проектных решений и прикладных систем;

- необходимость интеграции существующих и вновь разрабатываемых приложений;

- функционирование в неоднородной среде на нескольких аппаратных платформах;

- разобщенность и разнородность отдельных групп разработчиков по уровню квалификации и сложившимся традициям использования тех или иных инструментальных средств;

- существенная временная протяженность проекта, обусловленная, с одной стороны, ограниченными возможностями коллектива разработчиков, и, с другой стороны, масштабами организации-заказчика и различной степенью готовности отдельных ее подразделений к внедрению ИС.

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

**Программа** – это объект разработки, который не является осязаемым (нельзя пощупать, взвесить и т. п.), доступен пониманию ЭВМ, для которой написан.

**Программный продукт** (ПП)**:** программа, работающая без авторского присутствия. Программный продукт исполняется, тестируется, конфигурируется без присутствия автора и сопровождается документацией.

**Программное обеспечение** (ПО) – совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ (ГОСТ 19781-90)

**Программная инженерия** (Software Engineering) ориентирована на разработку программного обеспечения прикладных и информационных систем разного назначения.

**Жизненный цикл ПО** – непрерывный процесс с момента принятия решения о создании ПО до снятия его с эксплуатации.

**Стандарт** (standard) – нормативно-технический документ, устанавливающий нормы и правила по отношению к объекту стандартизации, утверждается компетентным органом;

**Типы стандартов.**

***Корпоративные стандарты*** разрабатываются крупными фирмами с целью повышения качества своей продукции. Создаются на основе собственного опыта компании, но с учетом требований мировых стандартов. Не сертифицируются, но являются обязательными для применения внутри корпорации.

***Отраслевые стандарты*** действуют в пределах организаций некоторой отрасли (министерства). Разрабатываются с учетом требований мирового опыта и специфики отрасли. Являются обязательными для отрасли. Подлежат сертификации.

***Государственные стандарты*** (ГОСТы) принимаются государственными органами и имеют силу закона. Разрабатываются с учетом мирового опыта или на основе отраслевых стандартов. Могут иметь как рекомендательный, так и обязательный характер. Для сертификации создаются государственные или лицензированные органы сертификации.

***Международные стандарты*** разрабатываются специальными международными организациями на основе мирового опыта и лучших корпоративных стандартов. Имеют сугубо рекомендательный характер.

***Основные стандарты***

***ISO/IEC 12207 – Information Technology – Software Life Cycle Processes –***

процессы жизненного цикла программных средств***.***

***SEI CMM – Capability Maturity Model (for Software)*** – модель зрелости процессов разработки программного обеспечения.

***ISO/IEC 15504 – Software Process Assessment –*** оценка и аттестация зрелости процессов создания и сопровождения ПО. Является развитием и уточнением ISO 12207 и SEI CMM.

***PMBOK – Project Management Body of Knowledge –*** свод знаний по управлению проектами.

***SWEBOK – Software Engineering Body of Knowledge –*** свод знаний по программной инженерии.

***ACM/IEEE CC2001 – Computing Curricula 2001*** академический образовательный стандарт в области компьютерных наук.

# **Вопрос 3 (Классификация программного обеспечения. Определение и состав системы программирования.)**

***Системное ПО –*** комплекс программ, которые обеспечивают управление компонентами компьютерной системы:

-управление ресурсами компьютера;

-создание копий используемой информации;

-проверка работоспособности устройств компьютера;

-и др.

***Прикладное ПО –*** предназначено для выполнения определённых пользовательских задач и рассчитано на непосредственное взаимодействие с пользователем.

***Инструментальное ПО –*** для автоматизации процесса разработки.

***Операционная система —*** комплекс системных программ, расширяющий возможности вычислительной системы, обеспечивающий управление её ресурсами, загрузку и выполнение прикладных программ, взаимодействие с пользователями.

***Системы программирования –*** системные программы, предназначенные для разработки программного обеспечения.

**Состав системы программирования**:

* Трансляторы,
* компоновщики,
* отладчики,
* профилировщики,
* программные библиотеки,
* редакторы кода,
* системы поддержки версий и пр.

# **Вопрос 4 (Интегрированная среда разработки Visual Studio. Представление символьной информации в кодировке Windows-**1251)

**Интегрированная среда разработки (IDE)** — это многофункциональная программа, которая поддерживает многие аспекты разработки программного обеспечения. Интегрированная среда разработки Visual Studio — это стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений.

**ASCII (American Standard Code for Information Interchange)** — американский стандартный код для обмена информацией.

**ASCII** — 8-битная кодировка для представления десятичных цифр, латинского и национального алфавитов, знаков препинания и управляющих символов

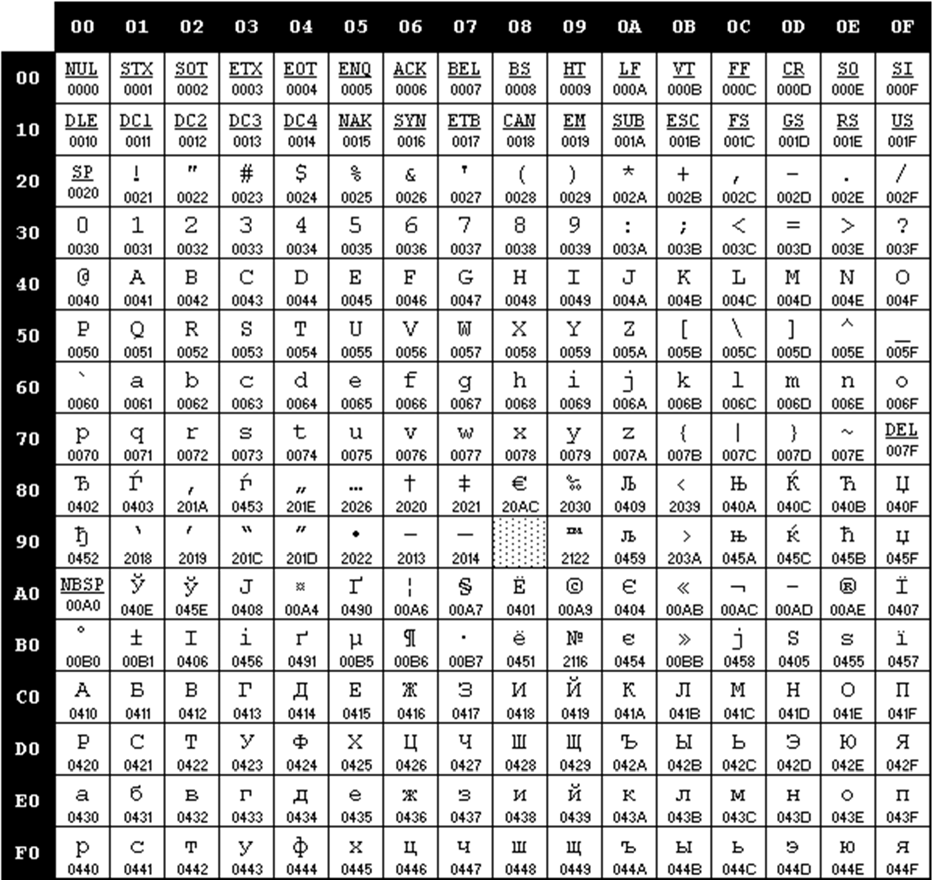
**Таблица кодов ASCII делится на две части:**

**Международным** стандартом является первая половина таблицы, т.е. символы с номерами от 0 (00000000), до 127 (01111111).

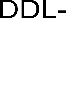
**Переносимый набор** является базовым алфавитом для практически всех современных языков программирования.

Русская Windows-кодировка (**Windows-1251**, синоним **CP1251**)

**Windows-1251** — набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий Microsoft Windows до 10-й версии.

Нижняя часть таблицы кодировки (латиница) полностью соответствует кодировке **[ASCII](https://ru.wikipedia.org/wiki/ASCII" \o "ASCII)**. Числа под буквами обозначают шестнадцатеричный код подходящего символа в [Юникоде](https://ru.wikipedia.org/wiki/%D0%AE%D0%BD%D0%B8%D0%BA%D0%BE%D0%B4" \o "Юникод).

# **Вопрос 5 (Компоненты классической системы программирования. Трансляторы, ассемблеры, интерпретаторы. Схема работы транслятора.)**



**Система программирования** – инструментальное ПО, предназначенное для разработки программного продукта на этапах программирования и отладки. Каждая система программирования должна иметь некоторый встроенный в нее язык программирования, предназначенный для общения разработчика с используемыми инструментами.

**Компилятор (транслятор)** – программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке; результат – объектный модуль

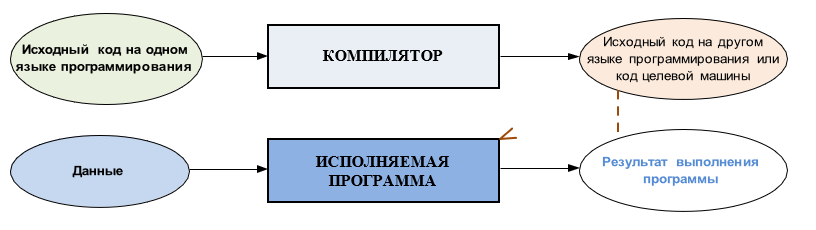
**Компоновщик (linker, редактор связей)** – программа, принимающая один или несколько объектных модулей и формирующая на их основе загрузочный модуль

**Загрузчик (loader)** – программа, предназначенная для запуска процесса операционной системы на основе загрузочного модуля

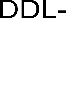
**Интерпретатор** – разновидность транслятора. Переводит и выполняет программу с языка высокого уровня в машинный код строка за строкой

**Язы́к ассе́мблера** ([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" \o "Английский язык) assembly language) — машинно-ориентированный [язык 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" \o "Язык программирования) [низкого уровня](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D0%B7%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%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" \o "Низкоуровневый язык программирования). Представляет собой систему обозначений, используемую для представления в удобно читаемой форме программ, записанных в [машинном коде](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4" \o "Машинный код). Его команды прямо соответствуют отдельным [командам машины](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8" \o "Код операции) или их последовательностям. Является существенно платформо-зависимым: языки ассемблера для различных [аппаратных платформ](https://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%B0" \o "Аппаратная платформа компьютера) несовместимы, хотя могут быть в целом подобны.

Схема работы транслятора: Исходный код на одном языке программирования 🡪 Компилятор (транслятор) 🡪 Исходный код на другом языке программирования или код целевой машины 🡪 Результат выполнения программы 🡨 ИСПОЛНЯЕМАЯ ПРОГРАММА 🡨 Данные

ё

# **Вопрос 6 (Структура классической системы программирования. Этапы обработки исходного кода программы. Язык программирования. Основные элементы языка программирования. Алфавит языка программирования, символы времени трансляции, символы времени выполнения.)**



**От исходного кода к исполняемому модулю, основные этапы преобразования:**

Классическая схема создания исполняемого файла выполняется для компилируемых языков:

1. обработка исходного кода препроцессором,
2. компиляция в объектный код
3. компоновка объектных модулей, включая модули из объектных библиотек, в исполняемый файл.

**Система программирования** – инструментальное ПО, предназначенное для разработки программного продукта на этапах программирования и отладки. Каждая система программирования должна иметь некоторый встроенный в нее язык программирования, предназначенный для общения разработчика с используемыми инструментами.

**Язык программирования -** формальная знаковая система, предназначенная для записи компьютерных программ. Знаковая система определяет набор лексических, синтаксических и семантических правил написания программы (программного кода). Язык программирования представляется в виде набора спецификаций, определяющих его синтаксис и семантику.

**Алфавит языка программирования** – набор символов, разрешенных к использованию языком программирования. Основывается на одной из кодировок.

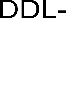
**Базовый набор символов исходного кода:**

1. строчные и прописные буквы латинского и национального алфавитов
2. цифры
3. знаки операций
4. символы подчеркивания \_ и пробельные символы
5. ограничители и разделители
6. специальные символы

**Набор символов времени трансляции:** текст программы на языке программирования хранится в исходных файлах и основан на определенной кодировке символов

**Набор символов времени выполнения:** символы, отображаемыми в среде выполнения. Любые дополнительные символы зависят от локализации

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



**Системой программирования** будем называть весь комплекс программных средств, предназначенных для кодирования, тестирования и отладки программного обес­печения. Нередко системы программирования взаимосвязаны и с другими тех­ническими средствами, служащими целям создания программного обеспечения на более ранних этапах жизненного цикла (от формулировки требований и ана­лиза до проектирования).

***Область кода*** – память, в которой размещается код программы.

***Статическая память***

**Static** или ***статическая память*** выделяется до начала работы программы, на стадии компиляции и служит для хранения статических переменных.

Типы статических переменных: **глобальные переменные** и **статические переменные**

**Глобальные** переменные – это переменные, определенные вне функций. Память для глобальных переменных выделяется на этапе компиляции.

Глобальные переменные доступны в любой точке программы во всех ее файлах.

**Статические** переменные – это переменные, в описании которых присутствует ключевое слово static. Компилятор выделяет для таких переменных постоянное место хранения в статической области памяти.

При объявлении переменной в функции ключевое слово static указывает, что переменная ***удерживает свое состояние между вызовами этой функции***.

***Стековая память***

**Stack** или ***стековая (автоматическая) память*** предназначена для хранения локальных переменных.

**Локальные** переменные хранятся в стеке.

**Стек** – это непрерывная область оперативной памяти, организованная по принципу LIFO (последний вошел, первый вышел).

***Динамическая память***

**Heap** или динамическая память, или куча – это область памяти, выделение которой в языке программирования С++ производится с помощью оператора new, освобождение — оператором delete.

|  |  |
| --- | --- |
| старшие адреса  stack  heap static  code  младшие адреса | Стековая память (автоматическая)  Динамическая память (куча)  Статическая память  Код программы |

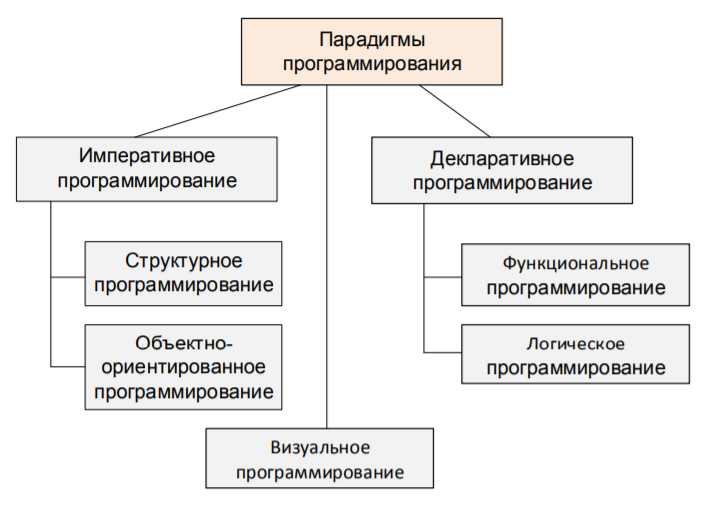
# **Вопрос 8 (Парадигмы программирования)**

**Парадигма программирования** – это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию)

**Стиль (парадигма) программирования** определяет базовые концепции языков программирования и их сочетания

Стили программирования классифицируются по трем признакам:

1. низкоуровневые языки;
2. высокоуровневые языки;
3. глобальность либо локальность действий и условий.



**Императивное программирование:**

Программа = последовательность действий, дающих указания компьютеру о том, **что** надо выполнить.

**Декларативное программирование:**

Программа = описание действий, которые необходимо выполнить компилятору для получения результата.

|  |  |
| --- | --- |
| **Декларативное программирование:**  Программа = описание действий, которые необходимо выполнить компилятору для получения результата.  Отвечает на вопрос **как** получить решение. | |
| ***Пример:***  Функция с именем add, принимает на вход массив и возвращает сумму всех его элементов. | ***Псевдокод:***  function add(array)  {  return array.reduce((prev, current) => prev + current, 0)  } |
| **Функциональное программирование:**  Программа = система определений и функций, описывающих что нужно вычислить, а как это сделать – решает транслятор; последовательность действий не прослеживается.  Раздел дискретной математики. Основой функционального программирования является лямбда-исчисление | |
| **Объектно-ориентированное программирование:**  Программа = несколько взаимодействующих объектов + функциональность (действия и данные распределяются между этими  объектами). | |
| **Распределённое (параллельное) программирование:**  Программа = совокупность описаний процессов, которые могут выполняться как параллельно (при наличии нескольких процессоров), так и в псевдопараллельном режиме (при наличии одного процессора). | |
| **Логическое программирование:**  Программа = система определений вида «условие => новый факт». Программа представляет собой описание фактов и правил вывода в некотором логическом исчислении. Результат, (который часто  записывается как вопрос), получается системой путем логического вывода. Раздел математической логики. | |
| **Распределённое (параллельное) программирование:**  Программа = совокупность описаний процессов, которые могут  выполняться как параллельно (при наличии нескольких процессоров), так и в псевдопараллельном режиме (при наличии одного процессора). | |
| **Визуальное программирование:**  Программа = способ создания программы для ЭВМ путём манипулирования графическими объектами вместо написания её текста.  Визуальное программирование позволяет программировать на уровне алгоритмов, а не программного кода.  Пакет визуального программирования генерирует, написанный на языках  программирования (1GL, 2Gl, 3GL), на основании составленной программистом «блок-схемы» в автоматическом режиме. | |
| **Аспектно-ориентированное программирование:**  Программа = к уже существующему коду добавляется дополнительного поведение, так называемой сквозной функциональности. | |

**Декларативное программирование** (лат. declaratio – объявление, подход возник в 60-х годах) – это предварительная реализация «решателя» для целого класса задач.

Тогда для решения конкретной задачи этого класса достаточно декларировать в терминах данного языка только её условие:

(исходные данные + необходимый вид результата)

«Решатель» сам выполняет процесс получения результата, реализуя известный ему алгоритм решения.

**Императивное программирование** (от греч. imper — действие) предполагает, что программа явно описывает алгоритм решения конкретной задачи (действия исполнителя), т.е. описывает как решать поставленную задачу.

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

* следование;
* ветвление;
* цикл.

# **Вопрос 9 (Язык программирования: определение, назначение, примеры. Исходный код. Объектный код. Объектный модуль. Загрузочный модуль. Оформления кода.)**

Язык программи́рования — формальный язык, предназначенный для записи компьютерных программ. Язык программирования определяет набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит ЭВМ под её управлением.

**Программа** – алгоритм, записанный на языке программирования.

**Текст программы (исходный код)** – полное законченное и детальное описание алгоритма на языке программирования.

**Объектный код:** – результат работы транслятора. Один файл объектного кода

– объектный модуль.

**Объектный модуль** – двоичный файл, который может быть объединён с другими объектными файлами при помощи редактора связей (компоновщика) для получения готового исполняемого модуля, либо библиотеки.

**Загрузочный код** – результат работы компоновщика.

Один файл загрузочного кода – загрузочный модуль.

|  |  |
| --- | --- |
| **Язык программирования:** | **формальная знаковая система**, предназначенная для записи компьютерных программ.  Знаковая система определяет набор **лексических, синтаксических и семантических** правил написания программы (программного кода).  Язык программирования представляется в виде набора спецификаций, определяющих его синтаксис и семантику. |

**Стандарт оформления кода** – набор правил и соглашений, используемых при написании исходного кода на некотором языке программирования.

Стандарт оформления кода описывает:

* способы выбора названий и используемый регистр символов для имен переменных и других идентификаторов:
  + запись типа переменной в ее идентификаторе;
  + регистр символов (нижний, верхний, «верблюжий», «верблюжий» с малой буквы), использование знаков подчёркивания для разделения слов;
* стиль отступов при оформлении логических блоков – используются ли символы табуляции, ширина отступа;
* способ расстановки скобок, ограничивающих логические блоки;
* использование пробелов при оформлении логических и арифметических выражений;
* стиль комментариев и использование документирующих комментариев.

# **Вопрос 10-12 Кодирование информации: определение, назначение, данные, представление данных, кодировки. Примеры.**

# **11.** **Спецификация системы программирования. Кодировка ASCII, кодировка Windows-1251, стандарт кодирования символов Unicode. Кодирование информации: определение, назначение, примеры. Кодировка UNICODE: назначение, структура, UCS, UTF. Прямой (LE) и обратный (BE) порядок байт. BOM: определение, назначение, примеры.**

**Кодирование** – это перевод информации с одного языка на другой (запись в другой системе символов, в другом алфавите). При этом обычно кодированием называют перевод информации с «человеческого» языка на формальный, например, в двоичный код, а **декодированием** – обратный переход.

**ASCII** (American Standard Code for Information Interchange) — американский стандартный код для обмена информацией.

ASCII — 8-битная кодировка для представления десятичных цифр, латинского и национального алфавитов, знаков препинания и управляющих символов.

***Расширенные таблицы*:** в расширенных таблицах символы с порядковыми номерами 128-255 представляют символы национальных языков.

**Переносимый набор символов**

является базовым алфавитом для практически всех современных языков программирования.

**Переносимый набор символов** (portable character set) – набор из 103 символов, которые (стандарт POSIX) должны присутствовать в любой используемой кодировке.

**Переносимый набор символов** включает в себя все печатные символы US-ASCII и часть управляющих и является базовым алфавитом для практически всех современных языков программирования

Русская Windows-кодировка (**Windows-1251**, синоним **CP1251**)

**Windows-1251** — набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий Microsoft Windows до 10-й версии.



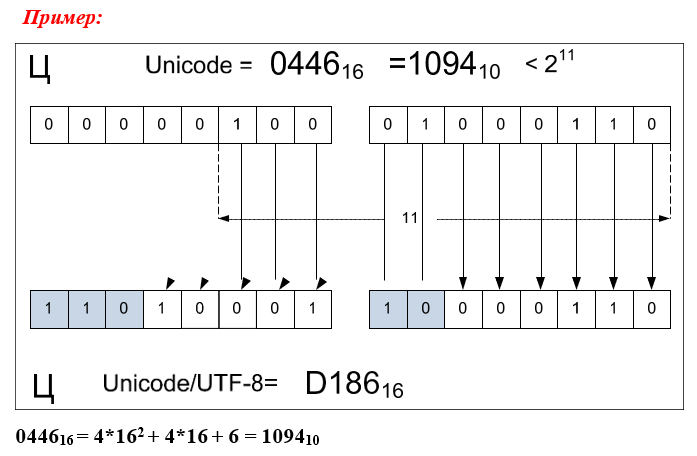
**Юникод** – стандарт кодирования символов, позволяющий представить знаки почти всех письменных языков, состоит из 2х разделов:

* **UCS** – universal character set (универсальный набор символов);
* **UTF** – Unicode transformation format (семейство кодировок).

UTF-8 — представление Юникода, обеспечивающее совместимость со старыми системами, использовавшими 8-битные символы.

**UNICODE: *кодировка UTF-16***

В UTF-16 символы кодируются двухбайтовыми словами (16 битов) с использованием всех возможных диапазонов значений (от 0 до FFFF16).



Спецификация системы программирования: набор требований к системе программирования, достаточный для ее разработки.

* **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.

Шестнадцатеричное представление маркера последовательности байтов для кодировок:

UTF-16 (BE) UTF-16 (LE) UTF-32 (BE)

UTF-32 (LE)

FF FE 00 00

00 00 FE FF

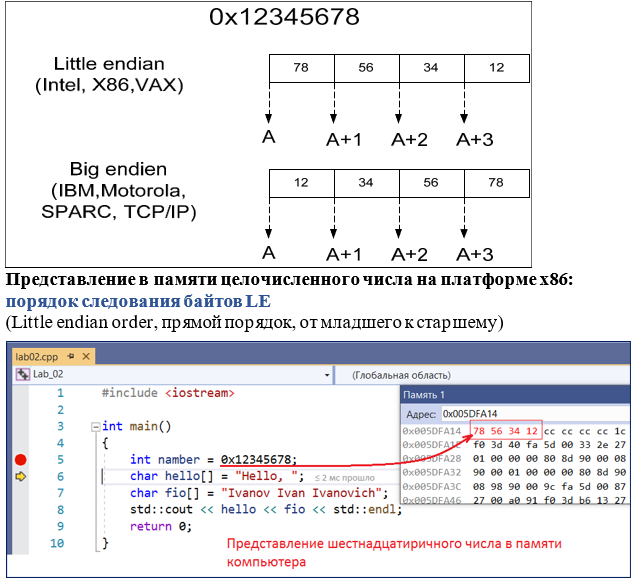
FF FE

FE FF

|  |  |
| --- | --- |
| **Кодировка** | **Представление (hex)** |
| UTF-8 | EF BB BF |

Порядок следования байтов:

* + **LE** (Little endian order, прямой порядок, от младшего к старшему);
  + **BE** (Big endian order, обратный порядок, от старшего к младшему).



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

Этапы и цели разработки программы:

|  |
| --- |
| 1. ***Постановка задачи.***    * определение функциональных возможностей программы;    * подготовка технического задания |
| 1. ***Выбор метода решения.***    * определение исходных и выходных данных, ограничений на них;    * выполнение формализованного описания задачи;    * построение математической модели, для решения на компьютере. |
| 1. ***Разработка алгоритма решения задачи.***    * выполняется на основе ее математического описания;    * полное и точное описание, определяющее вычислительный процесс, ведущий от начальных данных к искомому результату. |
| 1. ***Написание программы на языке программирования (кодирование)***    * запись алгоритма на языке программирования. |
| 1. ***Ввод программы в компьютер***    * подготовка исходного кода программы в виде текстового, который поступает на вход транслятора. |
| 1. ***Трансляция***    * преобразование исходного кода с одного языка программирования в семантически эквивалентный код на другом языке;    * получение объектного модуля. |
| 1. ***Компоновка***    * объединение одного или нескольких объектных модулей программы и объектных модулей статических библиотек в исполняемую программу;    * связывание вызовов функций и их внутреннего представления (кодов), расположенных в различных модулях;    * получение исполняемого (загрузочного) файла. |
| 1. ***Выполнение***    * выполнение исполняемого файла программы на целевой машине. |
| 1. ***Тестирование***    * устранение ошибок в программе. |
| ***10. Отладка***   * обнаружение, локализация и устранение ошибок. |
| ***11. Документирование***   * создание пользовательской документации. |
| ***12. Эксплуатация***   * выполнение в предназначенной для этого среде в соответствии с пользовательской документацией |
| ***13. Модификация (Реинжиниринг)***   * внесение изменений в ПО в целях исправления ошибок, повышения   производительности или адаптации к изменившимся условиям работы или требованиям. |
| ***14. Снятие с эксплуатации***   * завершение жизненного цикла ПП и изъятие его из эксплуатации. |

1. **Трудоемкость этапов**

|  |  |  |  |
| --- | --- | --- | --- |
| Этапы | Трудозатраты | Ошибки |  |
|  |  | Появление | Выявление |
| Постановка задачи | 10% | 40-46% | 50% |
| Математическая формулировка |  |  |  |
| Выбор метода решения |  |  |  |
| Составление алгоритма | 20% | 35-38% |  |
| Написание программы на языке программирования | 15% |  |  |
| Ввод программы в компьютер | 5% | 5-10% |  |
| Выполнение программы |  |  |  |
| Тестирование | 40% |  | 45% |
| Отладка |  |  |  |
| Документирование | 10% |  | 3% |
| Эксплуатация |  |  |  |
| Реинжиниринг |  |  |  |

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

14.1

***последовательность операций для решения конкретного множества задач и обладает пятью важными чертами: конечность, определённость, ввод, вывод, эффективность.***

***определяет***

***который***

***правил,***

***набор***

***конечный***

**Алгоритм**

14.2

***конечность*** *(возможность получения результата за конечное число шагов);*

***массовость*** *(применимость к некоторому классу объектов);*

***эффективность*** *(оптимальность времени и ресурсов, необходимых для реализации алгоритма).*

*толкования*

*(однозначность*

***определенность***

*инструкций);*

***дискретность*** *(возможность разбиения на шаги);*

***понятность*** *(ориентирован на исполнителя);*

**Свойства алгоритмов**

1

|  |  |
| --- | --- |
|  | ***словесно-формульный*** *(на естественном языке);* |
|  | ***графический*** *(структурный или блок-схемой);* |
| **Способы описания алгоритмов** | ***использование псевдокода*** *(специальных алгоритмических языков);* |
|  | ***с помощью сетей Петри;*** |
|  | ***программный.*** |

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

15.1 **Система программирования** – инструментальное ПО, предназначенное для разработки программного продукта на этапах программирования и отладки. Каждая система программирования должна иметь некоторый встроенный в нее язык программирования, предназначенный для общения разработчика с используемыми инструментами

15.2

|  |  |  |
| --- | --- | --- |
| Windows | ARM, IA-64, Itanium, MIPS, DEC  Alpha, PowerPC и x86 | Microsoft |
| Linux | DEC Alpha, x86, x86\_64, ARM,  PowerPC, RISC-V и MIPS | Свободное программное  обеспечение |
| macOS | Motorola 68k, PowerPC, IA-32,  x86-64, ARM | Apple Inc. |
| OS/2 | x86 | IBM, Microsoft |
| Unix | Intel x86 |  |
| Solaris | SPARC, x86, x86-64, PowerPC | Sun Microsystems |

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

Международная организация по стандартизации (ИСО) одобрила C++ 20, последнюю версию объектно-ориентированного языка программирования. Официальный стандарт опубликуют к концу 2020 года.

Новая версия C++ выходит примерно каждые три года, ей присваивают номер года. Стандарт языка C++ 20 является преемником C++ 17.

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.

**Стандарт POSIX** – POSIX - это стандарт, описывающий интерфейс между операционной системой и прикладной программой. Фактически, этот стандарт описывает то, как программа должна вызывать системные функции для того, чтобы свободно запускаться под всеми операционными системами, отвечающими данному стандарту.

15.4 **Парадигма программирования** – это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию).

|  |  |
| --- | --- |
| **Императивное программирование:**  Программа = последовательность действий, дающих указания компьютеру о том, **что** надо выполнить.  Конструкции языка: последовательность действий; условная конструкция;  цикл. | |
| ***Пример:***  Функция с именем add, принимает на вход массив и возвращает сумму всех его элементов. | ***Псевдокод:***  function add(array)  {  let result = 0  for (let i = 0; i < array.length; i++) result += array[i];  return result;  } |

|  |  |
| --- | --- |
| **Декларативное программирование:**  Программа = описание действий, которые необходимо выполнить компилятору для получения результата.  Отвечает на вопрос **как** получить решение. | |
| ***Пример:***  Функция с именем add, принимает на вход массив и возвращает сумму всех его элементов. | ***Псевдокод:***  function add(array)  {  return array.reduce((prev, current) => prev + current, 0)  } |
| **Функциональное программирование:**  Программа = система определений и функций, описывающих что нужно вычислить, а как это сделать – решает транслятор; последовательность действий не прослеживается.  Раздел дискретной математики. Основой функционального программирования является лямбда-исчисление | |
| **Объектно-ориентированное программирование:**  Программа = несколько взаимодействующих объектов + функциональность (действия и данные распределяются между этими  объектами). | |
| **Распределённое (параллельное) программирование:**  Программа = совокупность описаний процессов, которые могут выполняться как параллельно (при наличии нескольких процессоров), так и в псевдопараллельном режиме (при наличии одного процессора). | |
| **Логическое программирование:**  Программа = система определений вида «условие => новый факт». Программа представляет собой описание фактов и правил вывода в некотором логическом исчислении. Результат, (который часто  записывается как вопрос), получается системой путем логического вывода. Раздел математической логики. | |
| **Распределённое (параллельное) программирование:**  Программа = совокупность описаний процессов, которые могут  выполняться как параллельно (при наличии нескольких процессоров), так и в псевдопараллельном режиме (при наличии одного процессора). | |
| **Визуальное программирование:**  Программа = способ создания программы для ЭВМ путём манипулирования графическими объектами вместо написания её текста.  Визуальное программирование позволяет программировать на уровне алгоритмов, а не программного кода.  Пакет визуального программирования генерирует, написанный на языках  программирования (1GL, 2Gl, 3GL), на основании составленной программистом «блок-схемы» в автоматическом режиме. | |
| **Аспектно-ориентированное программирование:**  Программа = к уже существующему коду добавляется дополнительного поведение, так называемой сквозной функциональности. | |

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

16.1 **Интегрированная среда разработки (integrated development environment – IDE):** – набор инструментов для разработки и отладки программ, имеющий общую интерактивную графическую оболочку, поддерживающую выполнение всех основных функций жизненного цикла разработки программы.

16.2 ***Примеры IDE*** (визуальные среды):

Eclipse, Microsoft Visual Studio, NetBeans, Qt Creator,

16.3  система программных средств, используемая программистами для разработки программного обеспечения, создания и тестирования программ или программного обеспечения.

16.4 Под **отладкой** программы понимается процесс испытания работы программы и исправления обнаруженных при этом ошибок. Обнаружить ошибки, связанные с нарушением правил записи программы **на** **языке** **программирования** (синтаксические и семантические ошибки), помогает используемая система **программирования**.

# 17. Интегрированная среда разработки MS Visual Studio 2019. Назначение, возможности.

|  |  |
| --- | --- |
| **Microsoft Visual Studio** — линейка продуктов компании Microsoft, включающих интегрированную среду разработки ПО и другие инструменты. Для разработки консольных приложений, игр, приложений с графическим интерфейсом, веб-сайтов, веб-приложений, веб-служб как в нативном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact  Framework и Silverlight. | |
|  | 1997 |
| разработчик | Microsoft |
| написана на | C++ и C# |
| ОС | Microsoft Windows, macOS |
| последняя версия | 16.11.3 (14 сентября 2021) |
| сайт | visualstudio.microsoft.com |

 для создания потрясающих приложений для Windows, Android и iOS, а также современных веб-приложений и облачных служб.

* **[Разработка.](https://docs.microsoft.com/ru-ru/visualstudio/ide/whats-new-visual-studio-2019?view=vs-2019" \l "develop)** Сосредоточьтесь на главном и повышайте продуктивность благодаря оптимизированной производительности, возможности мгновенной очистки кода и более точным результатам поиска.
* **[Совместная работа.](https://docs.microsoft.com/ru-ru/visualstudio/ide/whats-new-visual-studio-2019?view=vs-2019" \l "collaborate)** Воспользуйтесь возможностями совместной работы в рамках рабочего процесса Git-first, функциями редактирования и отладки, а также рецензирования кода прямо в Visual Studio.
* **[Отладка.](https://docs.microsoft.com/ru-ru/visualstudio/ide/whats-new-visual-studio-2019?view=vs-2019" \l "debug)** Выделяйте определенные значения и переходите к ним; оптимизируйте использование памяти и создавайте автоматические моментальные снимки при выполнении вашего приложения.

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

**18.1 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 коде программы.

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

**18**.**2**. **Точка останова** (breakpoint) – это точка, в которой процесс выполнения программы приостанавливается и отладчик получает управление.

Установить точку останова можно, щелкнув слева от строки по серому полю.

Если точка останова не установлена, то отладчик запускается и выполняет приложение целиком.

Иначе отладчик запускается и останавливается в первой точке останова.

***Модификация (Реинжиниринг)***

* внесение изменений в целях повышения производительности или адаптации к изменившимся условиям работы или требованиям.
  + **Пошаговая отладка**

**Некоторые возможности управления режимом отладки:**

|  |  |  |  |
| --- | --- | --- | --- |
| Иконка на  панели инструментов | Пункт меню «Отладка» | Горячие клавиши | Описание |
| Go |  |  | продолжить выполнение программы до следующей точки останова |
|  |  |  |  |
|  |  |  |  |
| Step Into |  |  | выполнить одну инструкцию с «заходом» в функцию.  Если это вызов функции, то точка выполнения перемещается на первую инструкции этой функции |
| Step Over |  |  | выполнить одну инструкцию.  Если это вызов функции, то она выполняется целиком |
| Step Out |  |  | прервать выполнение текущей функции и  вернуться в вызывающую функцию |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Дизассемблер** — Программа, преобразующая машинный или объектный код в ассемблерный исходный текст.

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

19.1 **Методология** **разработки** **программного** **обеспечения** - **это** совокупность принципов, система идей, понятий, способов, **методов** и средств, которые в конечном счете будут определять стиль **разработки** ПО. Иными словами, **методология** здесь - реализация какого-либо определенного стандарта.

Структурное программирование – стиль написания программ без ***goto***. В структурном программировании необходимо:

составить правильную логическую схему программы; реализовать ее средствами языка программирования;

Программа – множество вложенных блоков (иерархия блоков), каждый из которых имеет один вход и один выход.

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

* следование;
* ветвление;
* цикл.

**Принципы разработки:**

* + программирование «сверху-вниз» (нисходящее программирование);
  + модульное программирование с иерархическим упорядочением связей между модулями/подпрограммами «От общего к частному»

Этапы проектирования:

* формулировка целей (результатов) работы программы;
* представление процесса работы программы (модель);
* выделение из модели фрагментов: определение переменных и их назначения, стандартных программных контекстов.

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

20.1 **Методология** включает в себя модель вычислений для данного стиля.

**Методология разработки** программного обеспечения – совокупность методов, применяемых на различных стадиях жизненного цикла программного обеспечения.

**20.2 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.

***Цели модульного программирования:*** уменьшить сложность программ; предотвратить дублирование кода, упростить тестирование программы и обнаружение ошибок

Плюсы модульного программирования:

* ускорение разработки (позволяет изменять реализацию функциональности модуля, не затрагивая при этом взаимодействующие с ним модули);
* повышение надежности (локализует влияние потенциальных ошибок рамками модуля);
* упрощение тестирования и отладки;
* взаимозаменяемость.

Минусы модульного программирования:

* модульность требует дополнительной работы программиста и определенных навыков проектирования программ.

***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. Система контроля версий Git: основные возможности. Фиксирование состояния. Ветвления. Слияния веток. Конфликты при слиянии веток. Ветвление проектов. Распределенная разработка.

**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.

20.4 Что такое Git

* Git – распределенная система контроля версий;
* поддерживается автономная работа (локальные фиксации изменений могут быть отправлены в репозиторий 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;
* целостность Git: все объекты сохраняются в базу данных не по имени, а по хеш-сумме содержимого объекта;
* удобный и интуитивно понятный набор команд;
* гибкая система ветвления проектов и слияния веток между собой;
* почти все операции выполняются локально;
* высокая производительность: скорость работы в Git кажется мгновенной;
* универсальный сетевой доступ с использованием протоколов http, ftp, rsync, ssh и др.

Основные определения

|  |  |
| --- | --- |
| ***Репозиторий Git*:** | Git хранит информацию в структуре данных, называемой ***репозиторий*** (repository).  Репозиторий хранится в папке проекта – в папке .git |
| ***Репозиторий***  хранит: | * набор коммитов (***commit objects***) * набор ссылок на коммиты (***heads***) |
| ***Commit objects***  содержат: | * набор файлов, отображающий состояние проекта в текущий момент времени * ссылки на родительские ***commit objects*** * SHA1-имя – 40 символьная строка, которая   ***уникально идентифицирует commit object*** |

git init – создание репозитория

git add <имена файлов> – добавляет файлы в индекс

git commit – выполняет коммит проиндексированных файлов в репозиторий

git status – показывает какие файлы изменились между текущей стадией и HEAD. Файлы разделяются на 3 категории: ***новые*** файлы, ***измененные*** файлы, ***добавленные*** новые файлы

git checkout < SHA1 или метка> – получение указанной версии файла, позволяет переключаться между ветками

git push – отправка изменений в удаленный репозиторий

git fetch – получение изменений из удаленного репозитория

git clone <remote url> – клонирование удаленного репозитория себе

git merge – слияние веток, переходм на ветку с которой хотим произвести слияние, и пишем -git merge test , test – это название ветки, как назовёте, то и пишите. Указываем ветку которую присоединяем.

# 22. Понятие веб-сервиса Github. Назначение и основные возможности GitHub. Совместная работа над проектом.

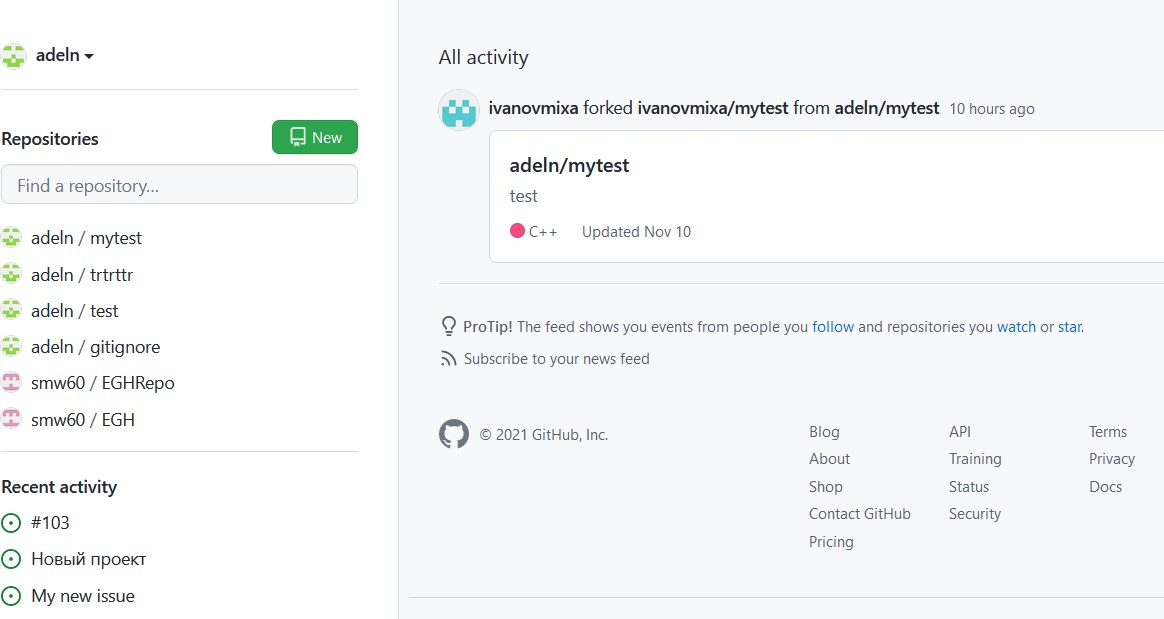
**Github** – крупнейший веб-сервис онлайн-хостинга репозиториев, используется для хостинга IT-проектов и их совместной разработки. Основан на системе контроля версий **Git**.

Github

* бесплатный сервис для проектов с открытым исходным кодом;
* место сотрудничества миллионов разработчиков и проектов (кроме размещения кода участники могут общаться, комментировать правки друг друга, а также следить за новостями знакомых); GitHub – социальная сеть для разработчиков;
* предоставляет удобный интерфейс для совместной разработки проектов и может отображать вклад каждого участника в виде дерева;
* дополнительно для проектов есть личные страницы, вики-разметка и система отслеживания ошибок;
* предоставляет возможность прямо на сайте просмотреть файлы проектов с подсветкой синтаксиса для большинства языков программирования.

21. 2 Совместная работа с репозиторием требуется, когда необходимо учитывать текущие задачи, выполнять требования к ним и исправлять баги.

На главной странице аккаунта отображается содержимое и все текущие активности:

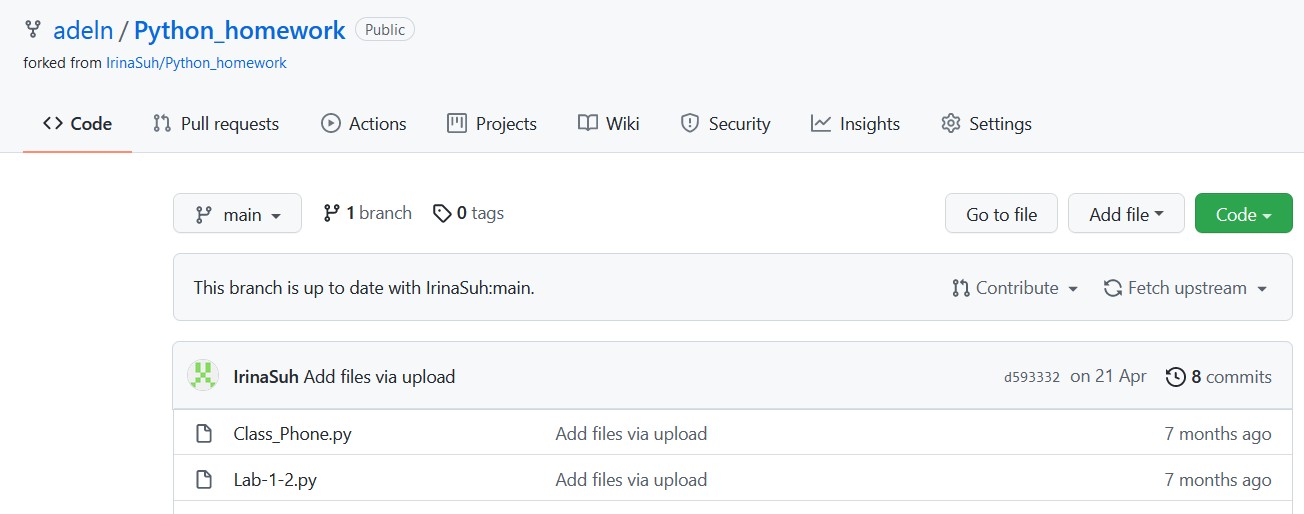


1. **Копирование репозитория в Github.**

Последовательность действий:

* + ***создайте форк репозитория коллеги:***

Нажать кнопку  в верхнем правом заголовка проекта коллеги. Репозиторий коллеги клонирован в отдельную ветку:



* + ***внесите изменения в своей ветке форка (например, добавив какой- нибудь файл).***
  + ***создайте запрос (pull-реквест) в репозиторий коллеги, предложив свои изменения:***

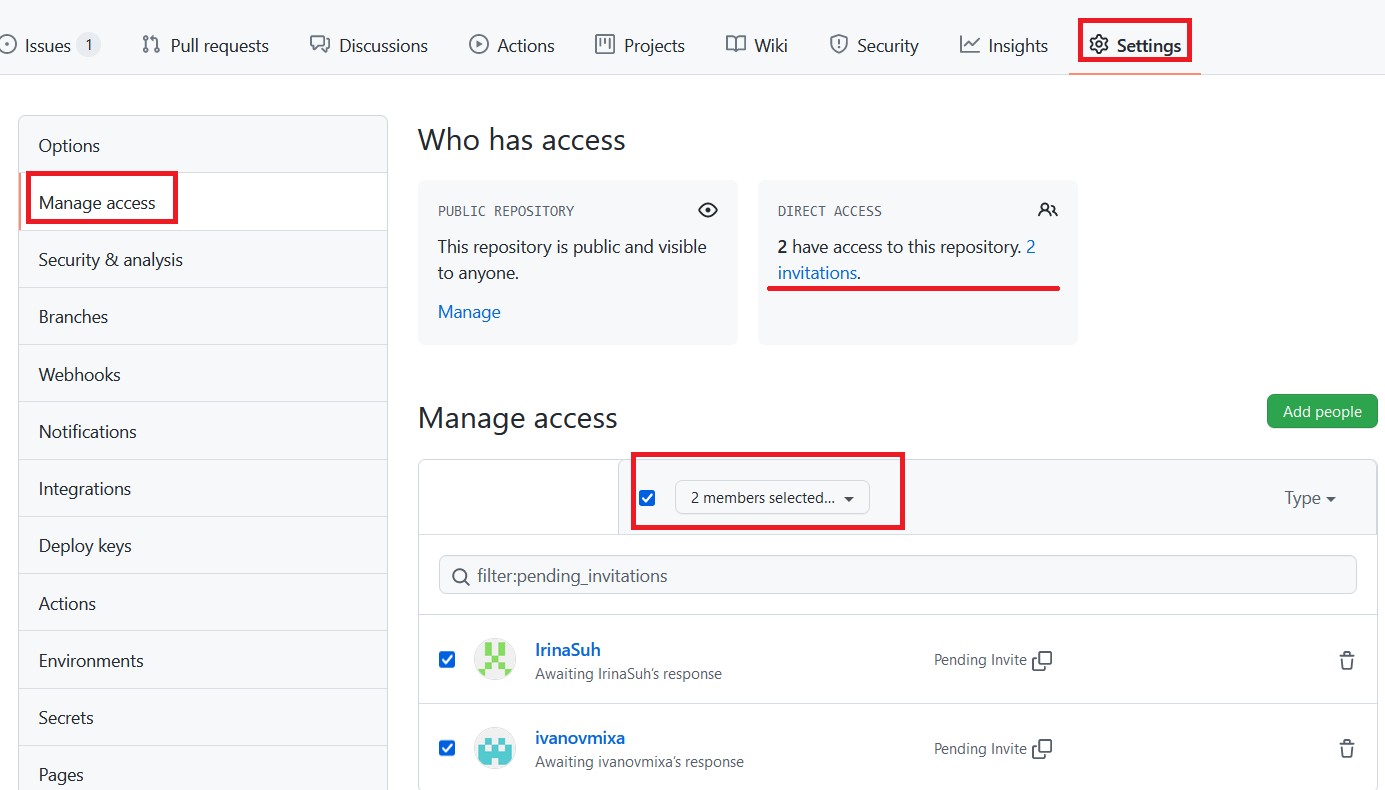


* + ***после слияния ваших изменений в исходный репозиторий его владельцем, заберите в свой форк последние изменения.***

Добавление членов команды: организация и соавторы

Существует два способа настройки Github для совместной работы:

* + ***Организации***. Владелец организации может создавать множество команд с разными уровнями доступа для различных репозиториев.
  + ***Сотрудники***. Владелец репозитория может добавлять коллабораторов с доступом Read + Write для одного репозитория.



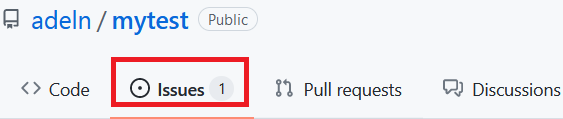
**Cоздание проблемы (issue)**

Для этого нужно включить вкладку issues.

Сделать это можно так: настройка проекта, отметить галочку issues. Появляется вкладка issues, с помощью которой можно ставить задачи и обсуждать их.

Перейти на страницу репозитория.

Под заголовком репозитория выбрать меню Issues:



Нажать кнопку



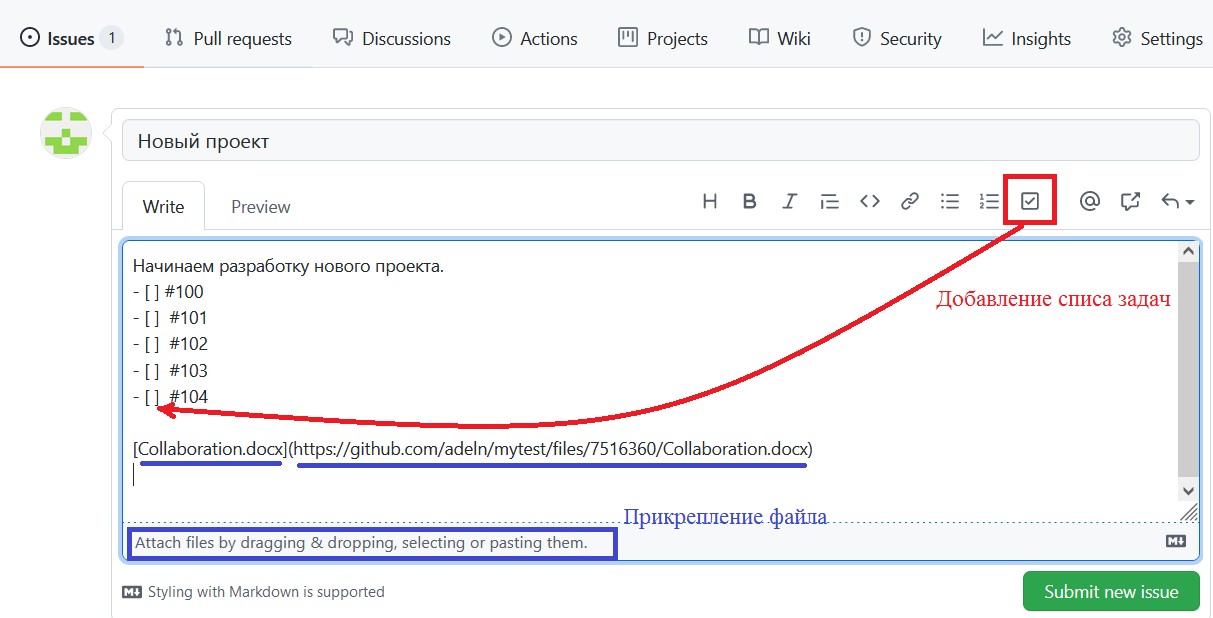
**Заполнение информации**

В поле заголовка ***даем*** проблеме ***название*** (название должно отражать суть проблемы).

В закладке Write рабочей области ввода ***добавляем*** текстовое ***описание***, объясняющее цель проблемы, включая любые подробности, которые могут помочь решить проблему. В нашем случае инициируем начало разработки нового проекта. Определяем цель и ожидаемый результат.

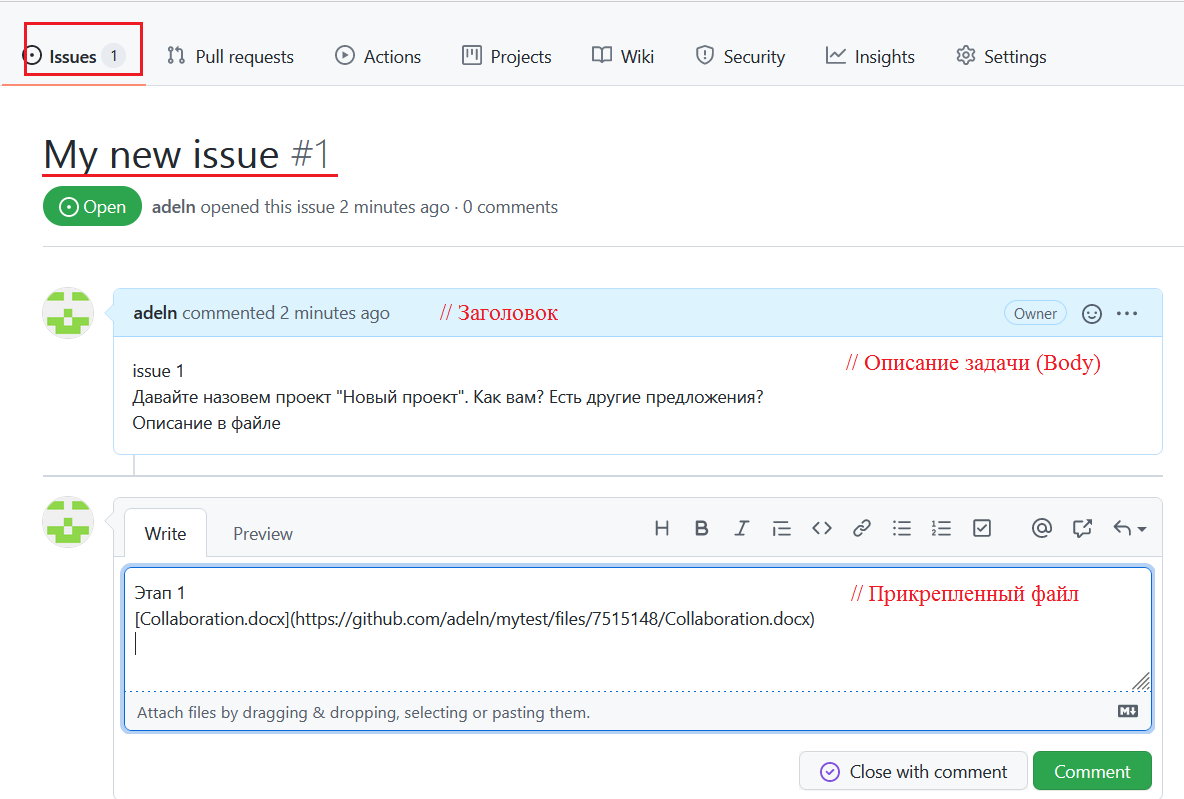
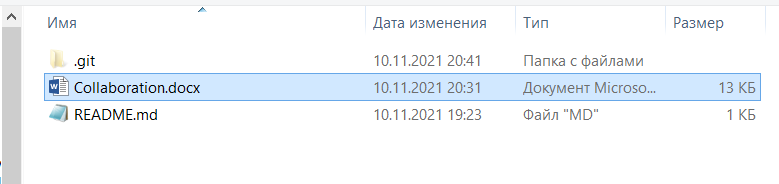
***Добавляем список задач*** – этапы разработки проекта (перед каждым элементом списка надо поставить символ [ ]). Элементы списка могут быть обычным текстом или ссылками на существующие проблемы по их номеру либо по URL. Текст можно отформатировать.

Так же можно прикрепить файл с дополнительной информацией.



Прикрепить файл можно несколькими способами:

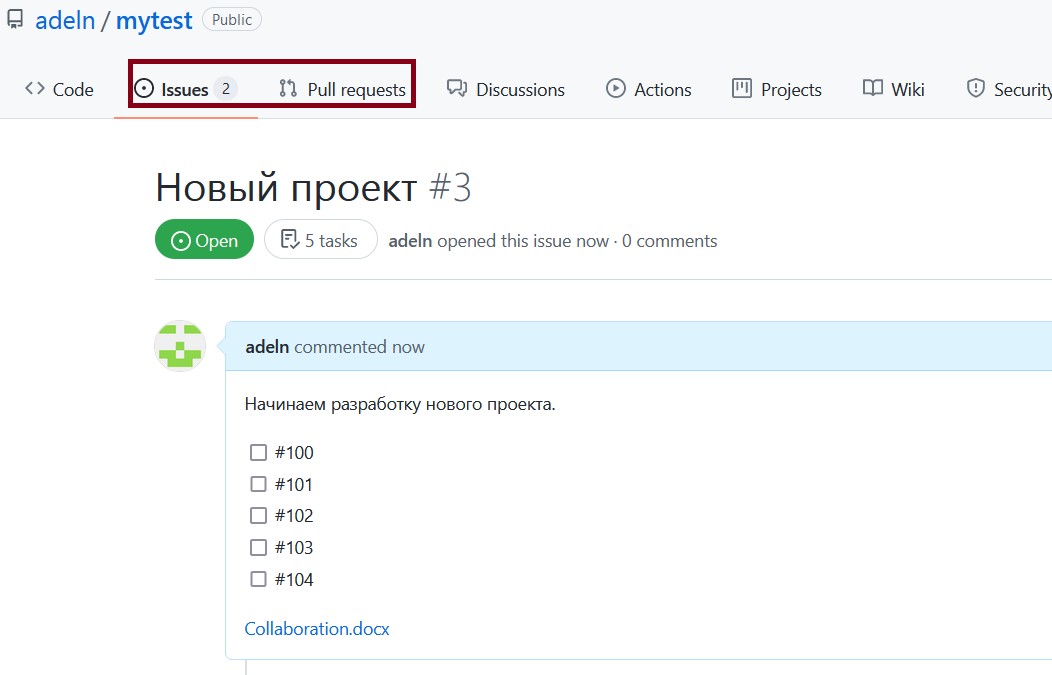
* перетащить его из папки;
* выбрать из пункта «выбрать файл»



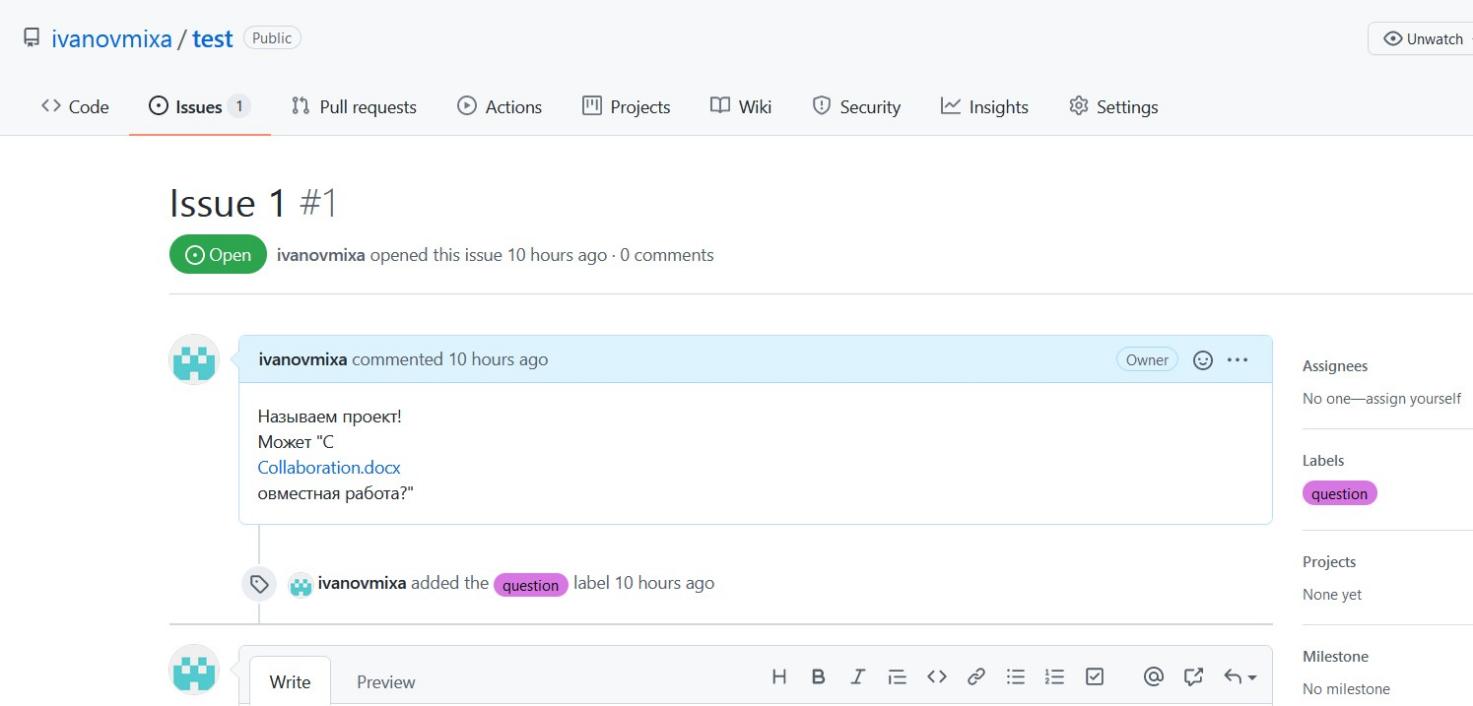
**Назначение проблем и задач другим пользователям**

Можно назначить до 10 человек для решения каждой проблемы, включая вас самих, и всех, у кого есть разрешения на запись в репозиторий.

Создаем новый проект, добавляем его описание и список задач, каждой из которых можно назначить исполнителя.



Отображение назначенной проблемы у соавтора:



22.

22.1 Создание программного обеспечения осуществляется последовательно в соответствии со следующими основными этапами:

постановка задачи (стадия «Техническое задание»)

анализ требований и разработка спецификаций (стадия «Эскизный проект»)

проектирование (стадия «Технический проект»)

реализация (стадия «Рабочий проект»)

внедрение и эксплуатация (стадия «Внедрение»)

# 23. Этапы создания программного продукта. Понятие жизненного цикла разработки программного обеспечения. Назначение модели жизненного цикла ПО. Структура процессов жизненного цикла программного обеспечения.

23. **Жизненный цикл разработки программного обеспечения**

Общепринятая модель жизненного цикла программного обеспечения, согласно которой программные системы проходят в своем развитии два этапа:

* разработка;
* сопровождение.

|  |
| --- |
| Использование |
| Разработка |
| Продолжающаяся разработка |
| (сопровождение) |

Этапы и цели разработки программы:

|  |
| --- |
| 1. ***Постановка задачи.***    * определение функциональных возможностей программы;    * подготовка технического задания |
| 1. ***Выбор метода решения.***    * определение исходных и выходных данных, ограничений на них;    * выполнение формализованного описания задачи;    * построение математической модели, для решения на компьютере. |
| 1. ***Разработка алгоритма решения задачи.***    * выполняется на основе ее математического описания;    * полное и точное описание, определяющее вычислительный процесс, ведущий от начальных данных к искомому результату. |
| 1. ***Написание программы на языке программирования (кодирование)***    * запись алгоритма на языке программирования. |
| 1. ***Ввод программы в компьютер***    * подготовка исходного кода программы в виде текстового, который поступает на вход транслятора. |
| 1. ***Трансляция***    * преобразование исходного кода с одного языка программирования в семантически эквивалентный код на другом языке;    * получение объектного модуля. |
| 1. ***Компоновка***    * объединение одного или нескольких объектных модулей программы и объектных модулей статических библиотек в исполняемую программу;    * связывание вызовов функций и их внутреннего представления (кодов), расположенных в различных модулях;    * получение исполняемого (загрузочного) файла. |
| 1. ***Выполнение***    * выполнение исполняемого файла программы на целевой машине. |
| 1. ***Тестирование***    * устранение ошибок в программе. |
| ***10. Отладка***   * обнаружение, локализация и устранение ошибок. |
| ***11. Документирование***   * создание пользовательской документации. |
| ***12. Эксплуатация***   * выполнение в предназначенной для этого среде в соответствии с пользовательской документацией |
| ***13. Модификация (Реинжиниринг)***   * внесение изменений в ПО в целях исправления ошибок, повышения   производительности или адаптации к изменившимся условиям работы или требованиям. |
| ***14. Снятие с эксплуатации***   * завершение жизненного цикла ПП и изъятие его из эксплуатации. |

**Каждая стадия разбивается на ряд этапов:**

Определение требований

Фаза разработки

Реализация

Проектирование

Спецификации

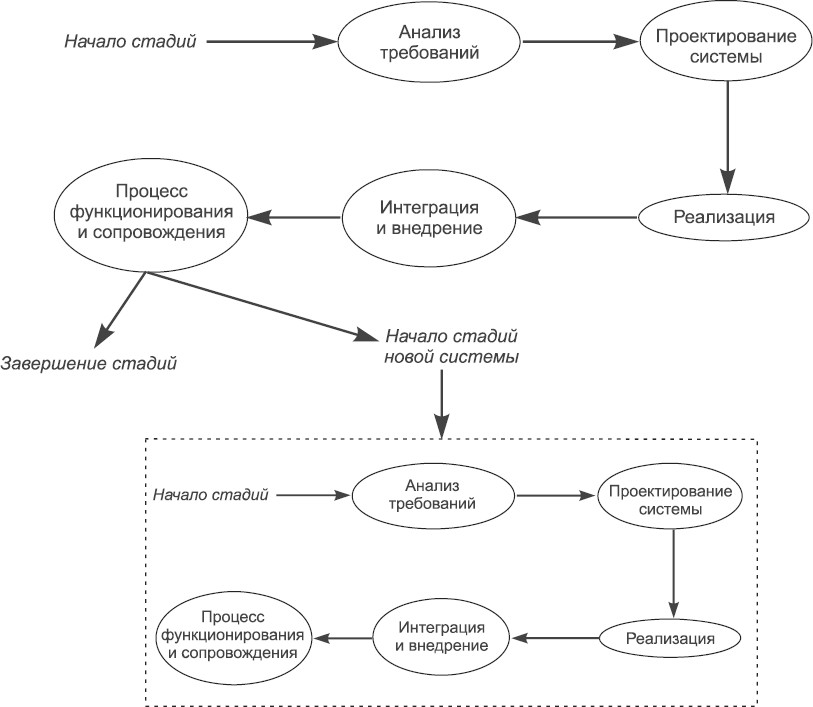
Фаза эксплуатации и сопровождения

Сопровождение

Тестирование

Развитие

**Стадии разработки ПО:**



**Определение:**

|  |  |
| --- | --- |
| **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. | |
| Миллиард лет эволюции за короткую жизнь - ОРГАНИЗМЕННЫЕ РЕГУЛЯЦИИ | это ряд событий, происходящих с ПО в процессе его создания и использования |

**Назначение**

|  |  |
| --- | --- |
| **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*** |

**МОДЕЛЬ ЖИЗНЕННОГО ЦИКЛА ПО**

|  |  |
| --- | --- |
|  | |
| **Модели разработки ПО:**   * каскадные; * итерационные; * поэтапные; * другие. | ***модели отличаются по:***   * этапности (фазы, стадии, этапы); * последовательности прохождения этапов (линейная или цикличная); * гибкости (возможность подстраивать процесс под конкретные условия); * связи с определенными методологиями разработки ПО; * использованию специализированных инструментальных средств; * другие. |

23.2 **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 |

1. **Виды моделей жизненного цикла**

В настоящее используются следующие модели жизненного цикла:

|  |  |
| --- | --- |
| **Каскадная (водопадная) модель** | * последовательное и однократное выполнение всех этапов проекта в строго фиксированном порядке; * переход на следующий этап после полного завершения работ на предыдущем этапе. |
| **Поэтапная модель**  **с промежуточным контролем**  http://toros.inevm.ru/repository/%7bBD0102D7-5B95-4CED-A288-A41E83A6260A%7d/%7b2799D3E5-9059-46E8-BF42-4E4733AEA38E%7d.gif | * разработка ведется итерациями с циклами обратной связи между этапами; * корректировки между этапами учитывают существующие взаимосвязи результатов разработки на различных этапах; * время жизни каждого из этапов растягивается на весь период разработки. |
| **Спиральная модель**  http://toros.inevm.ru/repository/%7bBD0102D7-5B95-4CED-A288-A41E83A6260A%7d/%7b12C32FB0-1C5A-4142-8D12-DA459174C4B4%7d.jpg | * на каждом витке спирали создается очередная версия продукта, уточняются требования проекта, определяется его качество и планируются работы следующего витка. |
| **каркасная модель разработки** | http://toros.inevm.ru/repository/%7bBD0102D7-5B95-4CED-A288-A41E83A6260A%7d/%7bEDDA062B-8A00-4BFE-8FC8-3E6A90CEBA21%7d.jpg |
| **сборочное программирование** | http://toros.inevm.ru/repository/%7bBD0102D7-5B95-4CED-A288-A41E83A6260A%7d/%7b02E099BA-FDB1-4C31-A2C7-879F510D3342%7d.gif |
| **исследовательское программирование** | http://toros.inevm.ru/repository/%7bBD0102D7-5B95-4CED-A288-A41E83A6260A%7d/%7bB8A69BB7-60A5-4D03-94B9-99FBD7D4F0B0%7d.gif |

# 24. Каскадная модель жизненного цикла ПС: содержание этапов, область применения, достоинства и недостатки.

24.1

|  |  |
| --- | --- |
| **Каскадная (водопадная) модель** | * последовательное и однократное выполнение всех этапов проекта в строго фиксированном порядке; * переход на следующий этап после полного завершения работ на предыдущем этапе. |
| ***преимущества:***   * на каждой стадии формируется законченный набор проектной документации; * выполняемые в логической последовательности стадии работ позволяют планировать сроки завершения всех работ и соответствующие затраты. | ***недостатки:***   * выявление и устранение ошибок производится только ***на стадии тестирования*** (как следствие, неточные спецификации приводят к переработке уже принятых решений); * реальные проекты часто требуют отклонения от стандартной последовательности шагов; * ЖЦ основан ***на точной формулировке исходных требований*** к ПО (на практике часто случается, что в начале проекта требования заказчика определены лишь частично); * результаты работ доступны заказчику только по завершении проекта. |

**Область применения каскадной модели:**

* в критически важных системах реального времени (например, управление авиационным движением или медицинским оборудованием);
* в масштабных проектах, в реализации которых задействовано несколько больших команд разработчиков;
* при разработке новой версии уже существующего продукта или переносе его на новую платформу;
* в организациях, имеющих большой практический опыт в создании программных систем определенного типа (например, бухгалтерский учет, начисление зарплаты и пр.).

# 25. Эволюционная модель жизненного цикла ПС: последовательность действий, область применения, достоинства и недостатки.

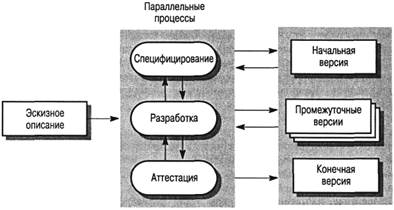
Эта модель основана на следующей идее: разрабатывается первоначальная версия программного продукта, которая передается на испытание пользователям, затем она дорабатывается с учетом мнения пользователей, получается промежуточная версия продукта, которая также проходит "испытание пользователем", снова дорабатывается и так несколько раз, пока не будет получен необходимый программный продукт (рис. 14.2.).

Отличительной чертой данной модели является то, что процессы специфицирования, разработки и аттестации ПО выполняются параллельно при постоянном обмене информацией между ними.

Различают два подхода к реализации эволюционного метода разработки.

1. *Подход пробных разработок*. Здесь большую роль играет постоянная работа с заказчиком (или пользователями) для того, чтобы определить полную систему требований к ПО, необходимую для разработки конечной версии продукта. В рамках этого подхода вначале разрабатываются те части системы, которые очевидны или хорошо специфицированы. Система эволюционирует (дорабатывается) путем добавления новых средств по мере их предложения заказчиком.

2. Прототипирование. Здесь целью процесса эволюционной разработки ПО является поэтапное уточнение требований заказчика и, следовательно, получение законченной спецификации, определяющей разрабатываемую систему. Прототип (под прототипом обычно понимается действующий программный модуль, реализующий отдельные функции создаваемого ПО) обычно строится для экспериментирования с той частью требований заказчика, которые сформированы нечетко или с внутренними противоречиями.



*Рис. 14.2. Эволюционная модель жизненного цикла*

Эволюционный подход часто более эффективен, чем подход, построенный на основе каскадной модели, особенно если требования заказчика могут меняться в процессе разработки системы. Достоинством процесса создания ПО, построенного на основе эволюционного подхода, является то, что спецификация может разрабатываться постепенно, по мере того как заказчик (или пользователи) осознает и сформулирует те задачи, которые должно решать программное обеспечение.

Вместе с тем данный подход имеет и некоторые недостатки.

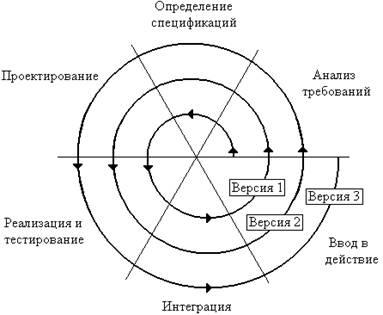
1. Многие этапы процесса создания ПО не документированы. Менеджерам проекта создания ПО необходимо регулярно документально отслеживать выполнение работ. Но если система разрабатывается быстро, то экономически не выгодно документировать каждую версию системы.

2. Система часто получается плохо структурированной. Постоянные изменения в требованиях приводят к ошибкам и упущениям в структуре ПО. Со временем внесение изменений в систему становится все более сложным и затратным.

3. Часто требуются специальные средства и технологии разработки ПО. Это вызвано необходимостью быстрой разработки версий программного продукта. Но, с другой стороны, это может привести к несовместимости некоторых применяемых средств и технологий, что, в свою очередь, требует наличия в команде разработчиков специалистов высокого уровня.

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

|  |  |
| --- | --- |
| **Спиральная модель**  http://toros.inevm.ru/repository/%7bBD0102D7-5B95-4CED-A288-A41E83A6260A%7d/%7b12C32FB0-1C5A-4142-8D12-DA459174C4B4%7d.jpg | * на каждом витке спирали создается очередная версия продукта, уточняются требования проекта, определяется его качество и планируются работы следующего витка. |
| ***преимущества:***   * прототипирование позволяет пользователям «увидеть» систему на ранних этапах разработки; * позволяет идентифицировать риски без особых дополнительных затрат; * предусматривает активное участие пользователей в работах по планированию, анализу рисков, разработке и   оценке полученных результатов. | ***недостатки:***   * модель имеет усложненную структуру; * сложность поддержания версий продукта; * сложность оценки точки перехода на следующий цикл; * спираль может продолжаться до бесконечности, поскольку каждая ответная реакция заказчика на созданную версию может порождать новый цикл, что отдаляет окончание работы над проектом. |



**Область применения спиральной модели**

* при разработке систем, требующих большого объема вычислений (например, систем принятия решений);
* при выполнении бизнес-проектов;

при выполнении проектов в области аэрокосмической промышленности, обороны и инжиниринга, где уже имеется позитивный опыт ее использования

**27. Инкрементальная модель разработки ПО. Развитие инкрементального подхода. XP-процессы.**

|  |  |
| --- | --- |
| **Incremental Model**  Описание: https://gbcdn.mrgcdn.ru/uploads/geekbrains/public/ckeditor_assets/pictures/7945/retina-ab02d17f0dcf4c0942944a17f49ac3f2.jpg | 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. | |

|  |  |
| --- | --- |
| **Extreme Programming (XP)**  Описание: 13 практик экстремального программирования | ***Экстремальное программирование*** – возможность вести разработку в условиях постоянно меняющихся требований.  ***Основные принципы:***   * итеративность: разработка ведется короткими итерациями при наличии активной взаимосвязи с заказчиком; * простота решений: принимается первое простейшее рабочее решение. Экстремальность метода связана с высокой степенью риска решения; * интенсивная разработка малыми группами (не больше 10 человек) и парное программирование; * обратная связь с заказчиком; * достаточная степень смелости и желание идти на риск. |
| ***преимущества:***   * заказчик получает именно тот продукт, который ему нужен; * команда быстро вносит изменения в код и добавляет новую функциональность; * код всегда работает за счет постоянного тестирования; * высокое качество кода; * код написан по единому стандарту и постоянно рефакторится; * быстрый темп разработки за счет парного программирования; * снижаются риски, связанные с разработкой. | ***недостатки:***   * успех проекта зависит от вовлеченности заказчика; * сложность планирования; * успех XP сильно зависит от уровня программистов; * регулярные встречи с программистами дорого обходятся заказчикам. |
| ***область применения:***   * из-за недостатка структуры и документации не подходит для крупных проектов. | |

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

**Методология разработки ПО** –

это система, определяющая порядок выполнения задач, методы оценки и контроля разработки программного обеспечения.

|  |
| --- |
| **Методологии разработки ПО:**   1. Waterfall 2. Rational Unified Process (RUP) 3. Agile – общая методология гибкой разработки 4. Spiral 5. Extreme Programming (XP) 6. Structured Analysis and Design Technique (SADT) 7. Microsoft Solutions Framework (MSF) Microsoft Operations Framework (MOF) 8. Rapid Application Development (RAD) 9. Personal Software Process 10. Scrum – концепция работы в условиях сорванных сроков и идеологического кризиса. |
|  |

**Управление требованиями** – непрерывный процесс на протяжении всего жизненного цикла продукта.

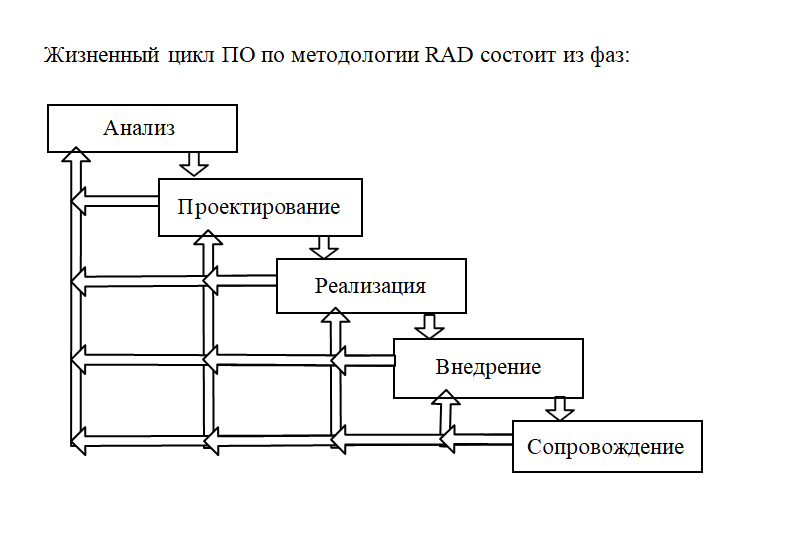
***Свойства требований:***

* + ***корректность (correct);***
  + ***однозначность (unambiguous);***
  + ***полнота (complete);***
  + ***непротиворечивость (consistent);***
  + ***приоритезация (prioritized);***
  + ***проверяемость (verifiable);***
  + ***модифицируемость (modifiable);***
  + ***отслеживаемость (traceable).***

Набор требований к продукту представляет собой техническое задание, при этом требования делятся на ***функциональные*** (то, что система позволяет сделать, желаемая функциональность) и ***нефункциональные*** (требования к оборудованию, операционной системе и т.п.).

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

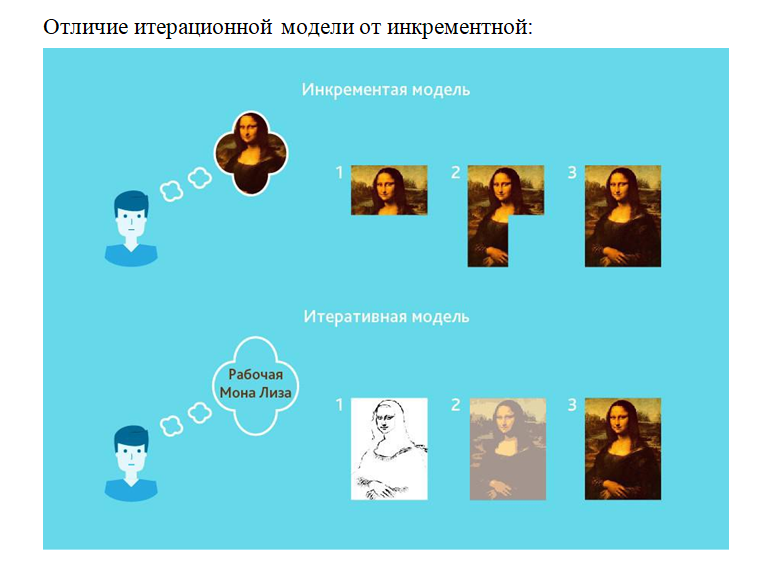
|  |  |
| --- | --- |
| **Rapid Application Development (RAD)**  Описание: https://issoft.by/wp-content/uploads/2019/06/Rapid_Application_Development-1024x384.png | * использование фокус-групп для сбора требований; * прототипирование и пользова- тельское тестирование * повторное использование программных компонентов; * использование плана, не включающего переработку, или дизайн следующей версии продукта; * проведение неформальных совещаний по запросу одной из сторон. |
| ***преимущества:***   * разработка 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:***   * 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. | |



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

|  |  |
| --- | --- |
| **Incremental Model**  Описание: https://gbcdn.mrgcdn.ru/uploads/geekbrains/public/ckeditor_assets/pictures/7945/retina-ab02d17f0dcf4c0942944a17f49ac3f2.jpg | 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. | |

|  |  |
| --- | --- |
| **Iterative Model**  Описание: Итеративная модель (Iterative model) - QALight | * определение и анализ требований; * дизайн и проектирование: согласно требованиям. Причем дизайн может как разрабатываться отдельно для данной функциональности, так и дополнять уже существующий; * разработка и тестирование: кодирование, интеграция и тестирование нового компонента; * фаза ревью: оценка, пересмотр текущих требований. |
| ***преимущества:***   * быстрый выпуск минимального продукта позволяет оперативно получать обратную связь от заказчика и пользователей; * постоянное тестирование   пользователями позволяет быстро обнаруживать и устранять ошибки. | ***недостатки:***   * переработка проекта; * заказчик не знает, как выглядит конечная цель и когда закончится разработка; * отсутствие фиксированного бюджета и сроков. |
| ***область применения:***   * требования к конечной системе заранее четко определены и понятны; * для работы над большими проектами с неопределёнными требованиями; * для задач с инновационным подходом, когда заказчик не уверен в результате. | |



**31. Методологии гибкой разработки ПО: Scrum, Kanban, Extreme Programming. Различия между Agile и традиционным подходом к разработке ПО. Преимущества и недостатки технологий быстрой разработки программного обеспечения.**

|  |  |
| --- | --- |
| **Agile/Scrum**  Описание: Фреймворк Scrum: сила простоты или сложность совершенства в Agile? | ***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.). |
| ***Agile/Scrum:***   * ***Scrum-команда*** – это команда, работающая над проектом (разработчики, тестеры, дизайнеры). * ***Scrum-мастер*** – организатор процесса, который следит, чтобы соблюдались принципы скрама. * ***Product owner*** – заказчик.   ***Митинги:***  ***Stand-up*** (короткий митинг, проводится каждый день; какие задачи должны быть выполнены каждым: Что делал? Что будет делать? Какие есть проблемы?) ** *Плэннинг*** (определяют какие задачи должны быть выполнены за следующий спринт) ****  ***Ретроспектива*** (проводится в  конце спринта) | ***недостатки:***   * успех проекта во многом зависит от скрам-мастера, квалификации команды и их приверженности своему делу; * далеко не всегда можно адаптировать метод скрам под сферу деятельности, поскольку есть проекты, требующие исключительно планового подхода в работе; * требует регулярной коммуникации с заказчиком, что порой тормозит процесс из-за невозможности получения обратной связи; * сложность внедрения в масштабных и сложных проектах, так как больше подходит для малых и средних. |
| ***область применения:***   * строительство, образование, производство товаров, Event -индустрия и другие виды деятельности. | |

|  |  |
| --- | --- |
| **Agile/Kanban** (с японского переводится, как «карточка»)  Описание: https://issoft.by/wp-content/uploads/2019/06/Simple-kanban-board--1024x610.jpg | ***Kanban кратко:*** agile-подход к разработке и управлению проектами, ориентированный на задачи:   * еженедельные собрания; * непрерывная разработка; * визуализация процесса на доске; * решение сначала самых важных задач; * поэтапные улучшения.   ***Цель***: анализ рабочего процесса и поиск точек для улучшения. |
| ***преимущества:***   * простота использования; * наглядность (помогает в определение узких мест, упрощает понимание); * высокая вовлеченность команды в сам процесс; * высокая гибкость в разработке. | ***недостатки:***   * нестабильный список задач; * сложно применять на долгосрочных проектах; * отсутствие жестких дедлайнов. |
| ***область применения:***   * когда потребности пользователей постоянно меняются в динамическом бизнесе; * отлично себя показывает в сферах неосновного производства. * хорошо работает при управлении стартапами без четкого плана, но где активно продвигается разработка. | |

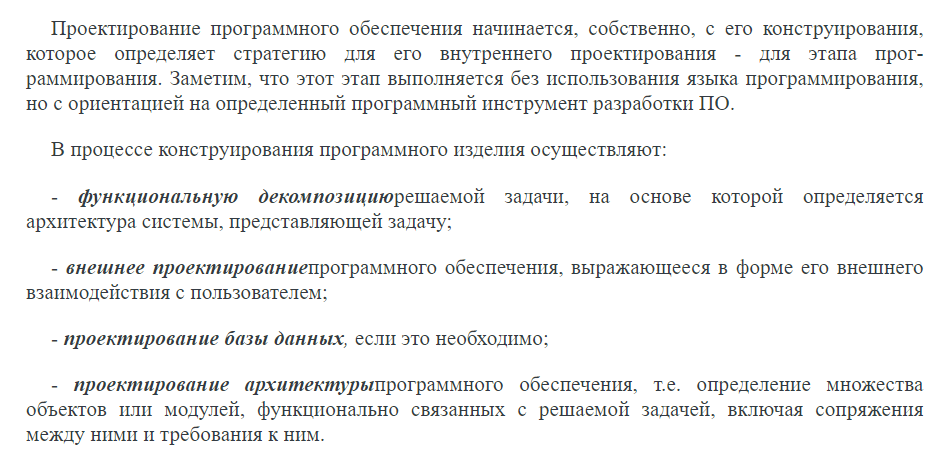
|  |  |
| --- | --- |
| **Extreme Programming (XP)**  Описание: 13 практик экстремального программирования | ***Экстремальное программирование*** – возможность вести разработку в условиях постоянно меняющихся требований.  ***Основные принципы:***   * итеративность: разработка ведется короткими итерациями при наличии активной взаимосвязи с заказчиком; * простота решений: принимается первое простейшее рабочее решение. Экстремальность метода связана с высокой степенью риска решения; * интенсивная разработка малыми группами (не больше 10 человек) и парное программирование; * обратная связь с заказчиком; * достаточная степень смелости и желание идти на риск. |
| ***преимущества:***   * заказчик получает именно тот продукт, который ему нужен; * команда быстро вносит изменения в код и добавляет новую функциональность; * код всегда работает за счет постоянного тестирования; * высокое качество кода; * код написан по единому стандарту и постоянно рефакторится; * быстрый темп разработки за счет парного программирования; * снижаются риски, связанные с разработкой. | ***недостатки:***   * успех проекта зависит от вовлеченности заказчика; * сложность планирования; * успех XP сильно зависит от уровня программистов; * регулярные встречи с программистами дорого обходятся заказчикам. |
| ***область применения:***   * из-за недостатка структуры и документации не подходит для крупных проектов. | |

****

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

|  |  |
| --- | --- |
| **Personal Software Process**  Описание: Методология PSP и TSP. Опыт внедрения, использования и адаптации процессов  PSP и TSP - PDF Скачать Бесплатно | ***Personal 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. | |

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

****

**Методологии разработки ПО:**

1. Waterfall
2. Rational Unified Process (RUP)
3. Agile – общая методология гибкой разработки
4. Spiral
5. Extreme Programming (XP)
6. Structured Analysis and Design Technique (SADT)
7. Microsoft Solutions Framework (MSF) Microsoft Operations Framework (MOF)
8. Rapid Application Development (RAD)
9. Personal Software Process

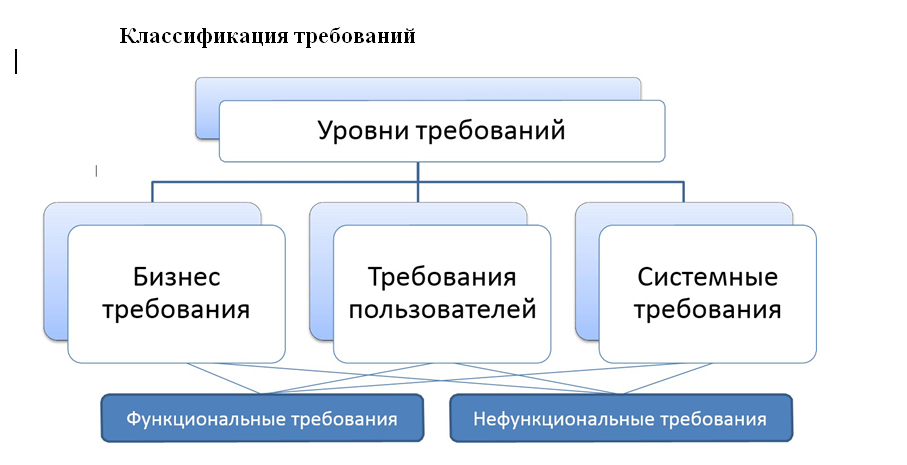
Scrum – концепция работы в условиях сорванных сроков и идеологического кризиса.

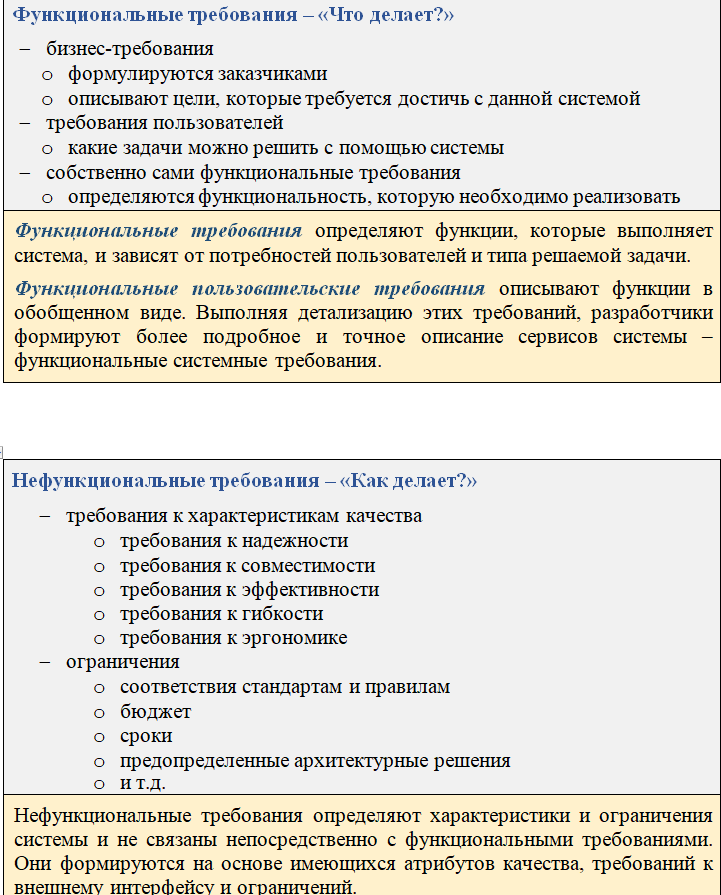
**34. Технологии разработки ПО. Управление требованиями. Понятие требования к ПО. Виды и уровни требований, классификация требований. Функциональные требования и нефункциональные требования. Разработка требований**

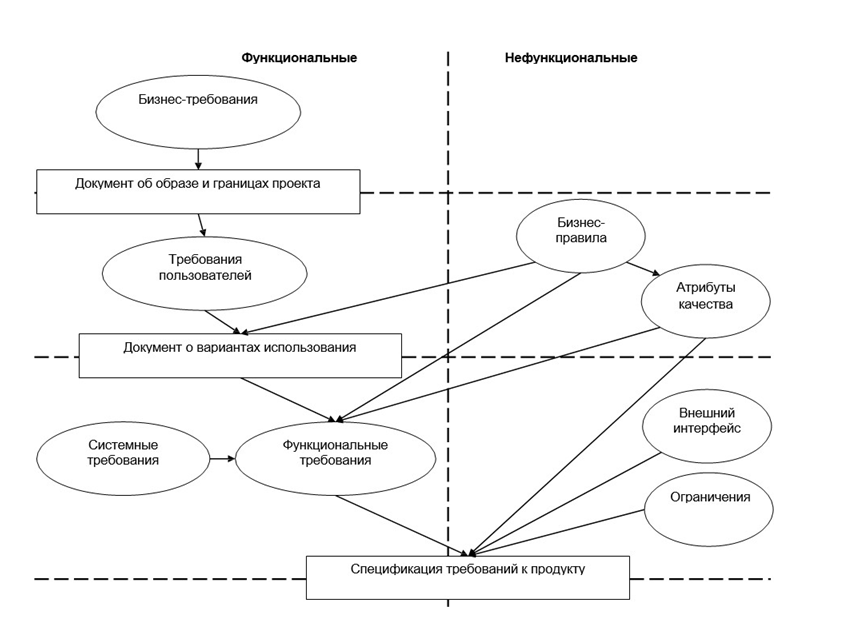
**Требование** –

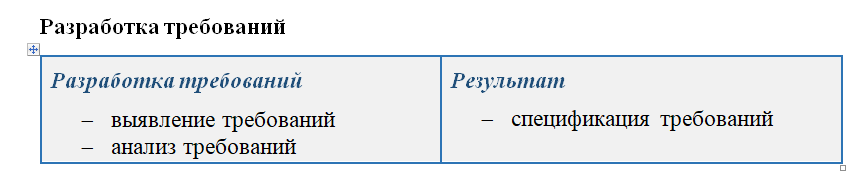
это утверждение, которое идентифицирует эксплуатационные, функциональные параметры, характеристики или ограничения проектирования продукта или процесса, которое однозначно, проверяемо и измеримо.

|  |  |
| --- | --- |
| **Управление требованиями** | ***процесс, включающий:***   * ***идентификацию, выявление, документацию, анализ, отслеживание, приоритизацию требований, достижение соглашений по требованиям и затем управление изменениями и уведомление заинтересованных лиц.*** |

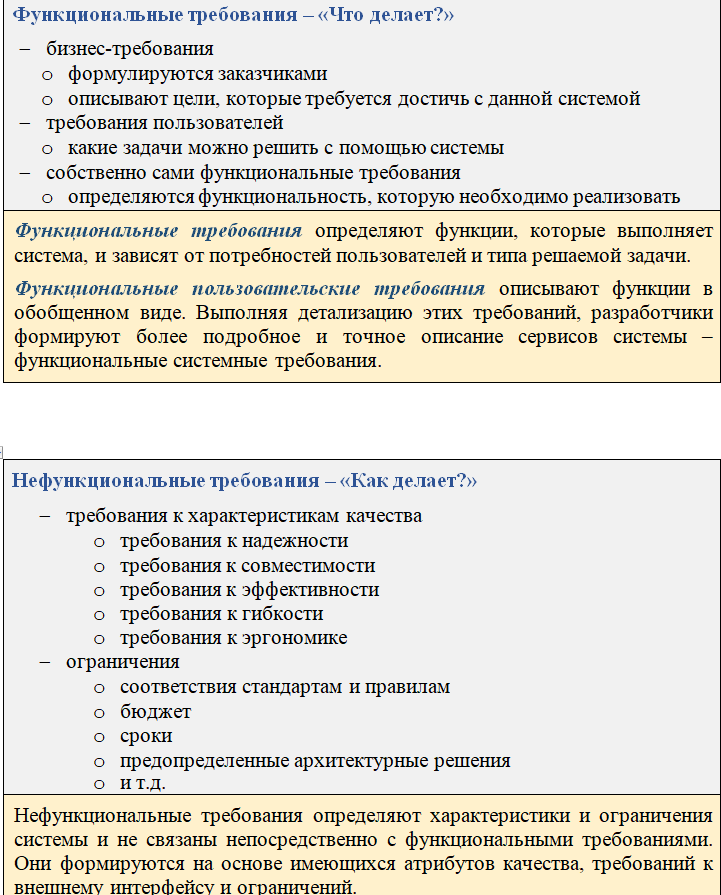
****

****

****

****

**35. Формализация функциональных требований: диаграммы вариантов использования. Назначение и компоненты диаграмм вариантов использования. Примеры.**



**Диаграмма вариантов использования (англ. use-case diagram) –**

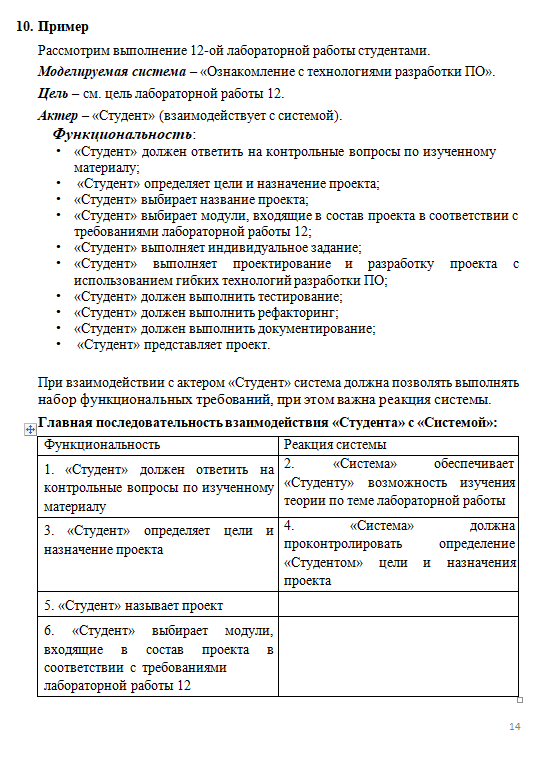
диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей.

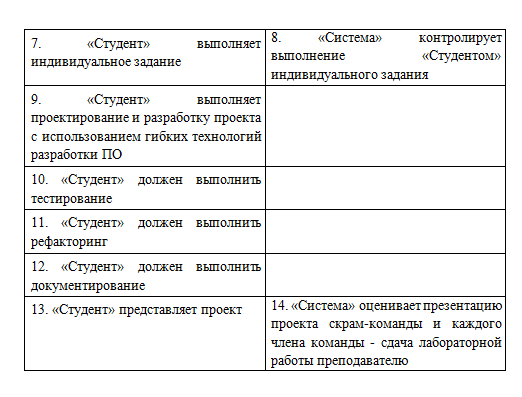
**Диаграммы вариантов использования**

* показывают взаимодействия между ***вариантами использования*** и ***действующими лицами***, отражая функциональные требования к системе с точки зрения ***пользователя***.

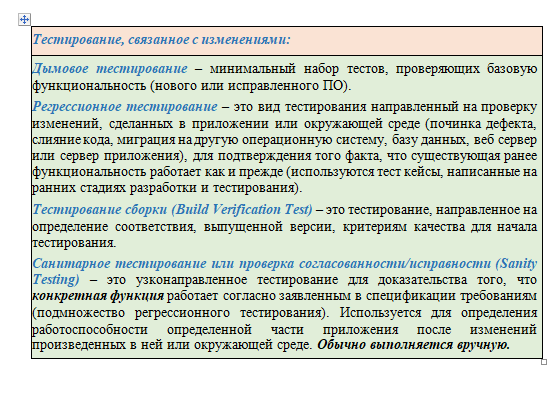
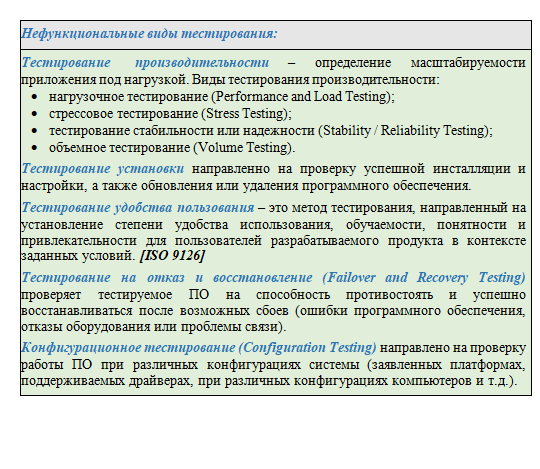
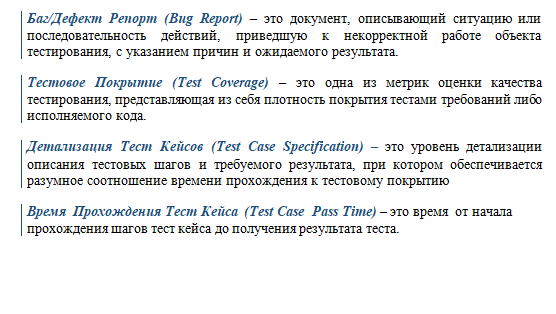
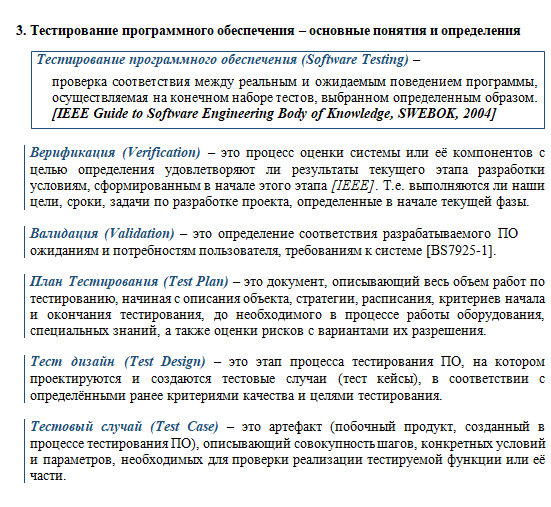
являются исходной концептуальной моделью системы в процессе ее проектирования и разработки.

|  |  |
| --- | --- |
| ***Диаграмма вариантов использования***  позволяет наглядно представить ожидаемое поведение системы.  Основными понятиями диаграмм вариантов использования являются: действующее лицо, вариант использования и связь. | |
| ***Основные понятия*** | * ***действующее лицо;*** * ***вариант использования;*** * ***связь***:   + ассоциация;   + отношение расширения;   + отношение включения;   + отношение обобщения. |

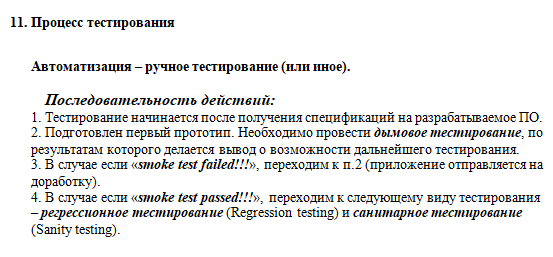
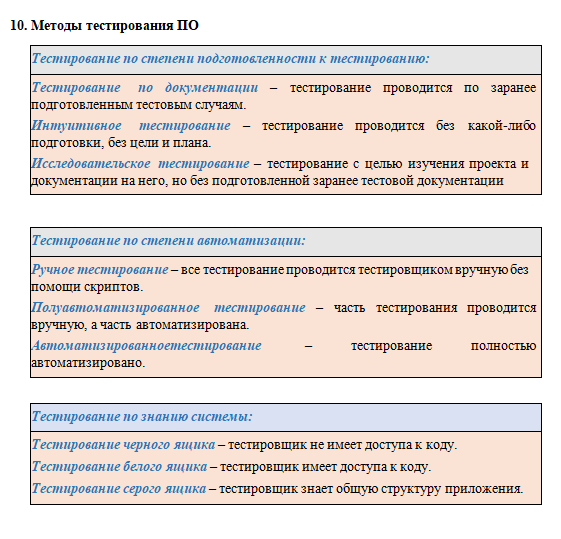




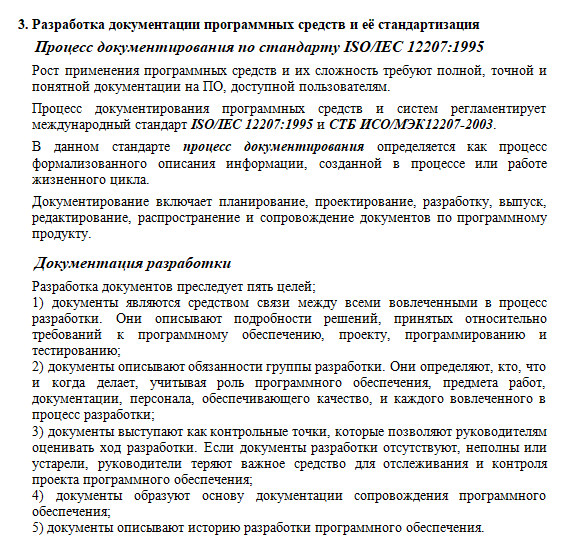
**36.Тестирование ПО: основные понятия и определения. Классификация видов тестирования. Цели, задачи и принципы тестирования.**

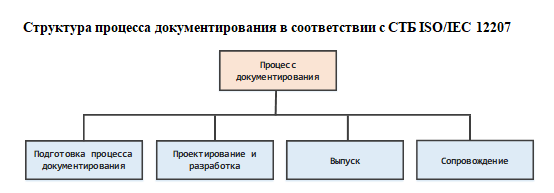
****

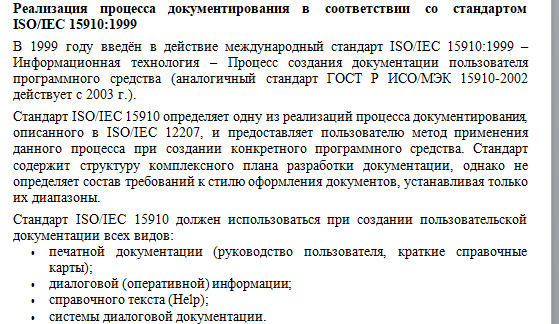
**37. Тестирование ПО: методы тестирования. Ручное тестирование.**

****

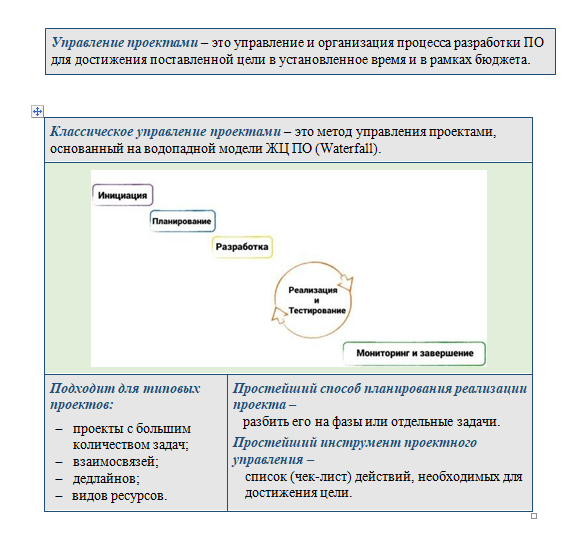
**38. Разработка программной документации. Назначение документирования программного обеспечения. Стандарты документирования.**

****

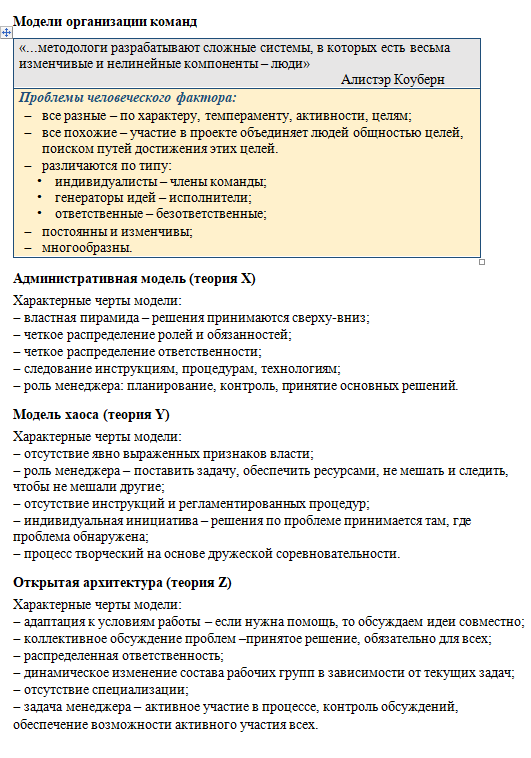
****

****

3**9.Управление командой проекта.**

****

****

****