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

Цель программной инженерии – сокращение стоимости программ.

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 (например, традиционных приложений, связанных с обработкой транзакций и решением регламентных задач, и приложений аналитической обработки (поддержки принятия решений), использующих нерегламентированные запросы к данным большого объема);

- 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.

Для успешной реализации проекта объект проектирования (ИС) должен быть прежде всего адекватно описан, должны быть построены полные и непротиворечивые функциональные и информационные модели ИС.

Накопленный к настоящему времени опыт проектирования ИС показывает, что это логически сложная, трудоемкая и длительная по времени работа, требующая высокой квалификации участвующих в ней специалистов. Однако до недавнего времени проектирование ИС выполнялось в основном на интуитивном уровне с применением неформализованных методов, основанных на искусстве, практическом опыте, экспертных оценках и дорогостоящих экспериментальных проверках качества функционирования ИС. Кроме того, в процессе создания и функционирования ИС информационные потребности пользователей могут изменяться или уточняться, что еще более усложняет разработку и сопровождение таких систем.

Перечисленные факторы способствовали развитию исследований в области методологии программирования. Программирование обрело черты системного подхода с разработкой и внедрением языков высокого уровня, методов структурного и модульного программирования, языков проектирования и средств их поддержки, формальных и неформальных языков описаний системных требований и спецификаций и т.д.

**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 – это применение систематического,

дисциплинированного и измеряемого подхода к разработке, эксплуатации и

сопровождению программного обеспечения (ПО) с применением инженерных

методов к разработке ПО

Жизненный цикл ПО – непрерывный процесс с момента принятия решения о

создании ПО до снятия его с эксплуатации.

*Стандарты программной инженерии*

Стандарт (standard) –норма, образец, мерило,

1)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.Не сертифицируются, но являются обязательными для применения внутри корпорации.

* Отраслевые стандарты действуют в пределах организаций некоторой

отрасли (министерства). Являются обязательными для отрасли. Подлежат

сертификации.

* Государственные стандарты (ГОСТы) принимаются государственными

органами и имеют силу закона. 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.

*Объекты стандартизации в программной инженерии*

* процессы разработки ПО;
* продукты разработки;
* ресурсы, которые используют процессы для создания программного продукта.

*Основные стандарты программной инженерии*

* 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.

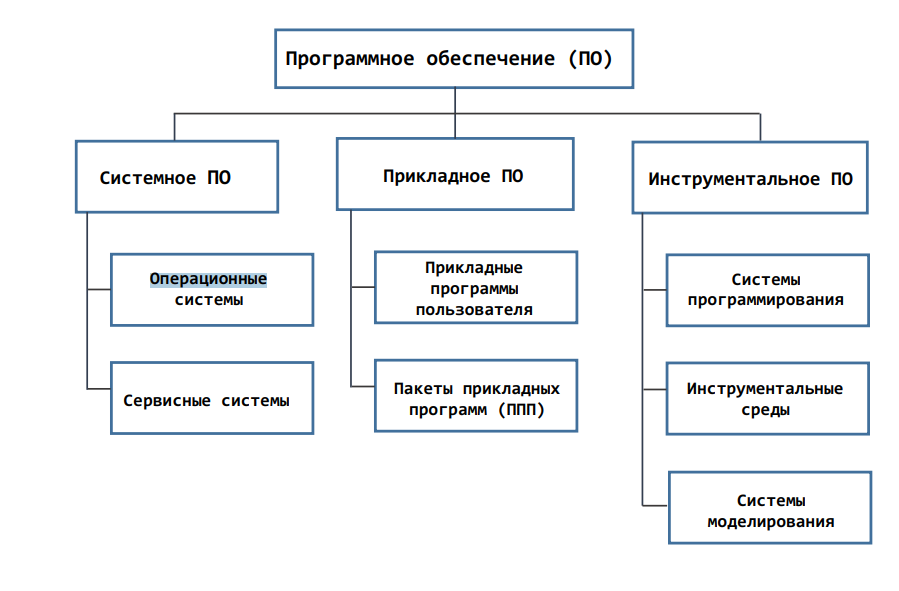
* SWEBOK – Software Engineering Body of Knowledge – свод знаний по

программной инженерии.

* ACM/IEEE CC2001 – Computing Curricula 2001 – кадемический

образовательный стандарт в области компьютерных наук

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



Классификация программного обеспечения по:

* по назначению (по области применения)
* по уровню
* по типу лицензии
* где применяется

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

* управление ресурсами компьютера;
* создание копий используемой информации;
* проверка работоспособности

НАИБОЛЕЕ ИСПОЛЬЗУЕМОЕ

Прикладное ПО – предназначено для выполнения определенных пользовательских задач и рассчитанная на непосредственное взаимодействие с пользователем.

Инструментальное ПО – используется для автоматизации процесса разработки.

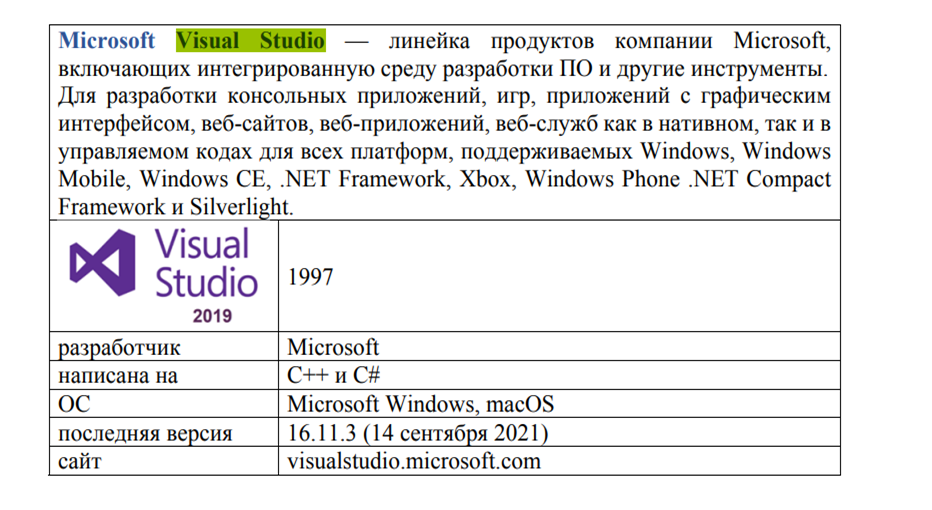
Операционная система — комплекс системных программ, расширяющий возможности вычислительной системы, обеспечивающий управление ее ресурсами, загрузку и выполнение прикладных программ, взаимодействие с пользователями.

Система программирования - комплекс программных средств, предназначенных для автоматизации процесса разработки, отладки программного обеспечения и подготовки программного кода к выполнению, является основным инструментом программиста.

*Состав системы программирования:*

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

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

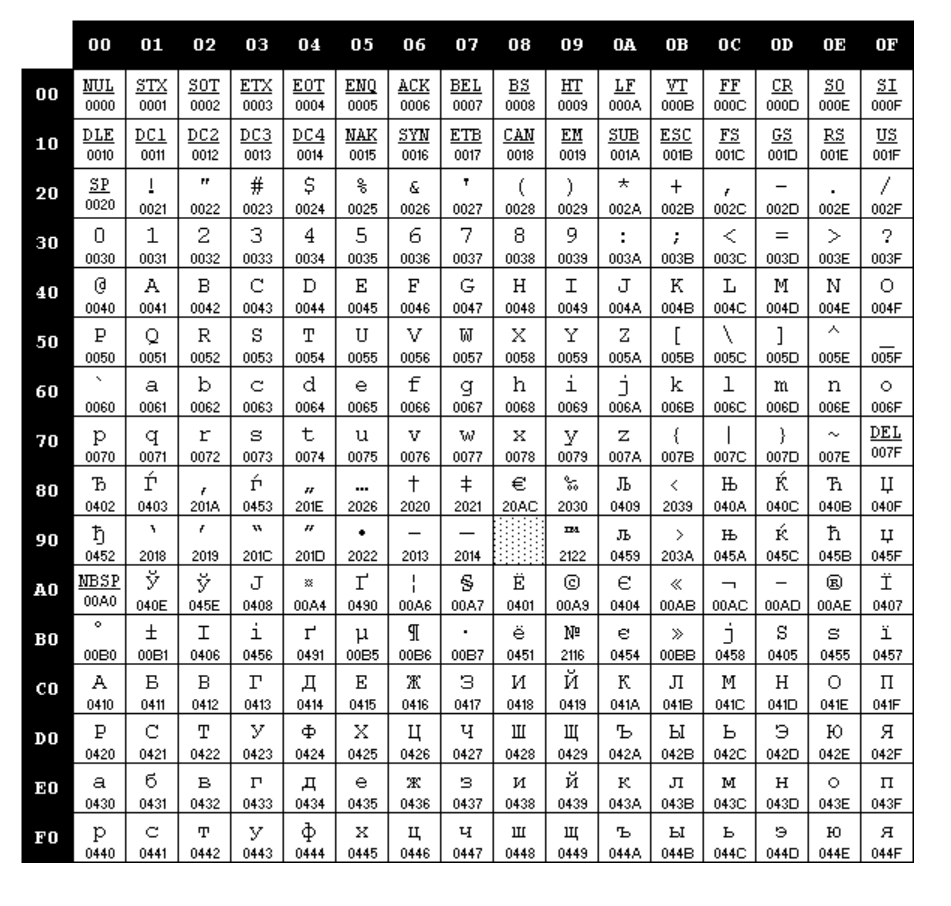


*Функции разработки:*

* Анализ
* Отладка
* Тестирование
* Управление версиями
* Совместная работа

Windows-1251 — набор символов и кодировка, являющаяся стандартной

8-битной кодировкой для русских версий Microsoft Windows .





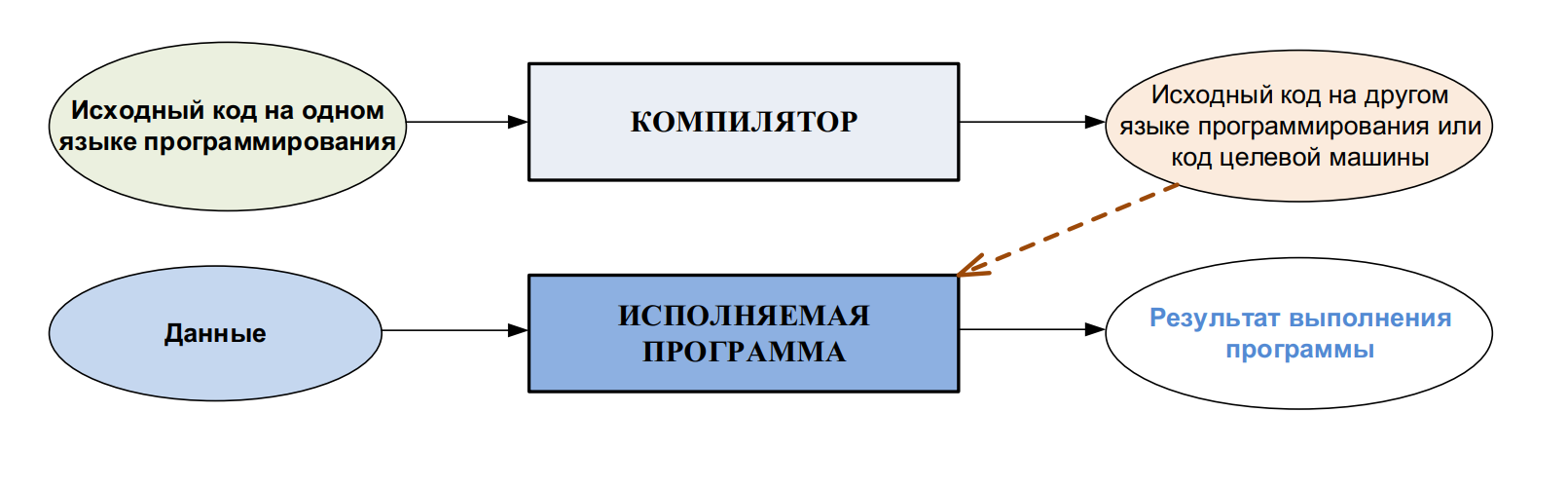
**5. Компоненты классической системы программирования. Трансляторы, ассемблеры, интерпретаторы. Схема работы транслятора.**

**Системы программирования** – системные программы, предназначенные для автоматизации процесса разработки, отладки программного обеспечения и подготовки программного кода к выполнению, является основным инструментом программиста.

**Состав системы программирования:**

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

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.

Отладчик (debugger) – программа, позволяющая контролировать ход выполнения программы и проверять её.

Профилировщик (англ, profiler) — программа, позволяющая оценить время работы каждой процедуры и функции; используется для того, чтобы выяснить, какую именно процедуру нужно оптимизировать в первую очередь.

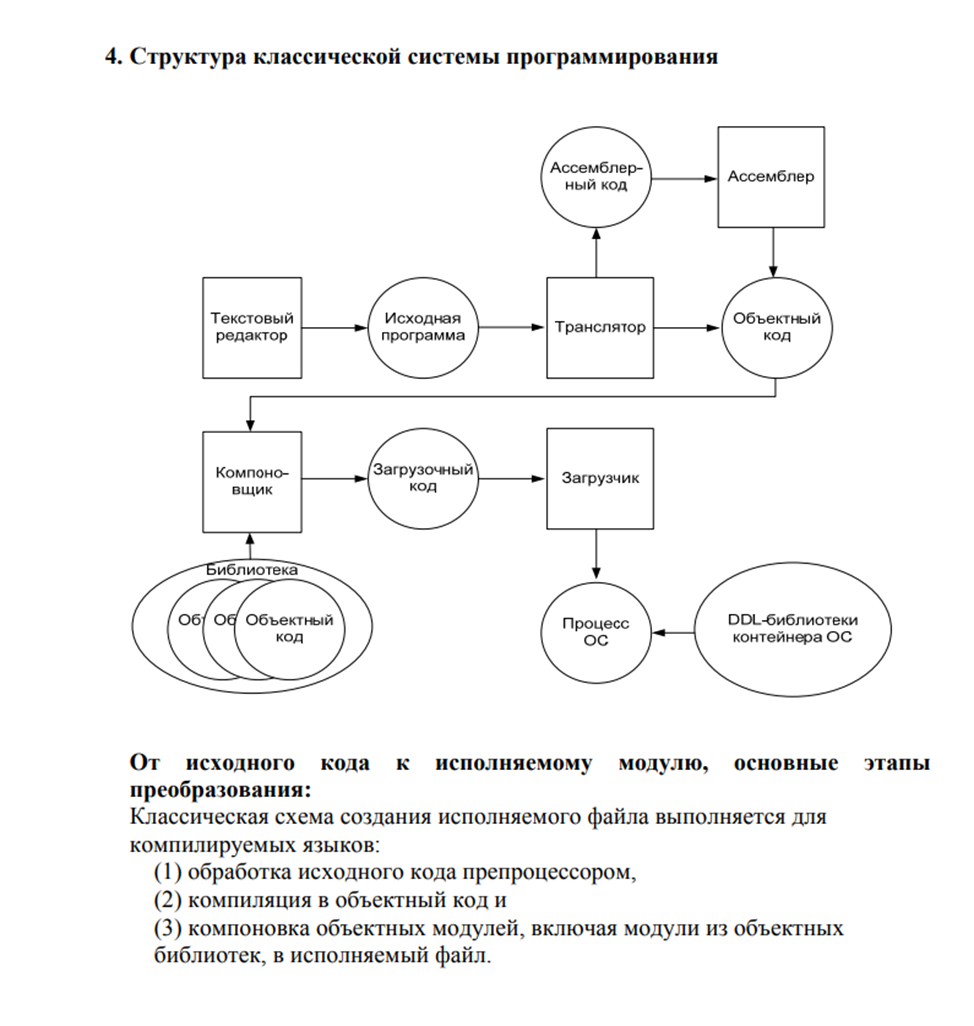
Программные библиотеки - это набор готовых процедур и функций, которые можно вызывать из своей программы.

Специализированный текстовый редактор позволяет программисту набрать и отредактировать текст программы на языке программирования высокого уровня. Для удобства ключевые слова языка программирования в текстовом редакторе могут выделяться различными цветами либо начертаниями шрифта.

Ассемблер (Assembly) — транслятор программы из текста на языке ассемблера, в программу на машинном языке.

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

**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 на языке C до исполняемой программы, он обычно проходит следующие этапы обработки:

***1) Препроцессинг***

Самая первая стадия компиляции программы.

Препроцессор — это макро процессор, который преобразовывает вашу программу для дальнейшего компилирования. На данной стадии происходит происходит работа с препроцессорными директивами. Например, препроцессор добавляет хэдеры в код (#include), убирает комментирования, заменяет макросы (#define) их значениями.

Файлы прошедшие через стадию препроцессинга C++ файлы имеют расширение .ii

***2) Компиляция***

Компилирует, то есть преобразует полученный на прошлом шаге код без директив в *ассемблерный код*. Это промежуточный шаг между высокоуровневым языком и машинным (бинарным) кодом.

Ассемблерный код — это доступное для понимания человеком представление машинного кода.Но для того, чтобы машина поняла наш код, требуется преобразовать его в машинный код, который мы и получим на следующем шаге.

***3) Ассемблирование***

Нам необходимо перевести ассемблерный код в машинный с помощью ассемблера.

Ассемблер преобразовывает ассемблерный код в машинный код, сохраняя его в объектном файле.

Объектный файл — это созданный ассемблером промежуточный файл, хранящий кусок машинного кода. Этот кусок машинного кода, который еще не был связан вместе с другими кусками машинного кода в конечную выполняемую программу, называется *объектным кодом*.

***4) Компоновка***

Компоновщик связывает все объектные файлы и статические библиотеки в единый исполняемый файл, который мы и сможем запустить в дальнейшем. Для того, чтобы понять как происходит связка, следует рассказать о *таблице символов*.

Таблица символов — это структура данных, создаваемая самим компилятором и хранящаяся в самих объектных файлах. Таблица символов хранит имена переменных, функций, классов, объектов и т.д., где каждому идентификатору (символу) соотносится его тип, область видимости. Также таблица символов хранит адреса ссылок на данные и процедуры в других объектных файлах.Именно с помощью таблицы символов и хранящихся в них ссылок компоновщик будет способен в дальнейшем построить связи между данными среди множества других объектных файлов и создать единый исполняемый файл из них.

***5) Загрузка***

Последний этап, который предстоит пройти нашей программе — вызвать загрузчик для загрузки нашей программы в память. На данной стадии также возможна подгрузка динамических библиотек - это отдельные файлы, предоставляющие прикладным программам набор наиболее часто используемых функций, и загружаемые на этапе выполнения при обращении программы к ОС с заявкой на выполнение функции из библиотеки.

Язык программирования - это формальная знаковая система, предназначенная для записи компьютерных программ.

Язык программирования определяется не только через спецификации стандарта языка, но и через реализации программных средств, обеспечивающих трансляцию или интерпретацию программ на этом языке, которые различаются по производителю, версии, времени выпуска, полноте воплощения стандарта, дополнительным возможностям; могут иметь определённые ошибки или особенности реализации.

Основные элементы языка программирования:

Алфавит - набор символов, разрешенный к использованию и воспринимаемый компилятором.С помощью символов алфавита записываются служебные слова, которые составляют словарь языка.

Лексемы(слова) - это структурная единица языка, которая состоит из элементарных символов языка и не содержит в своем составе других структурных единиц языка.

Синтаксис - набор правил, описывающий комбинации символов алфавита, считающиеся правильно структурированной программой (документом) или её фрагментом. Синтаксису языка противопоставляется его семантика. Синтаксис языка описывает «чистый» язык, в то же время семантика приписывает значения (действия) различным синтаксическим конструкциям.

Выражения – состоят из лексем и символов

Операторы – состоят из выражений, лексем и символов

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

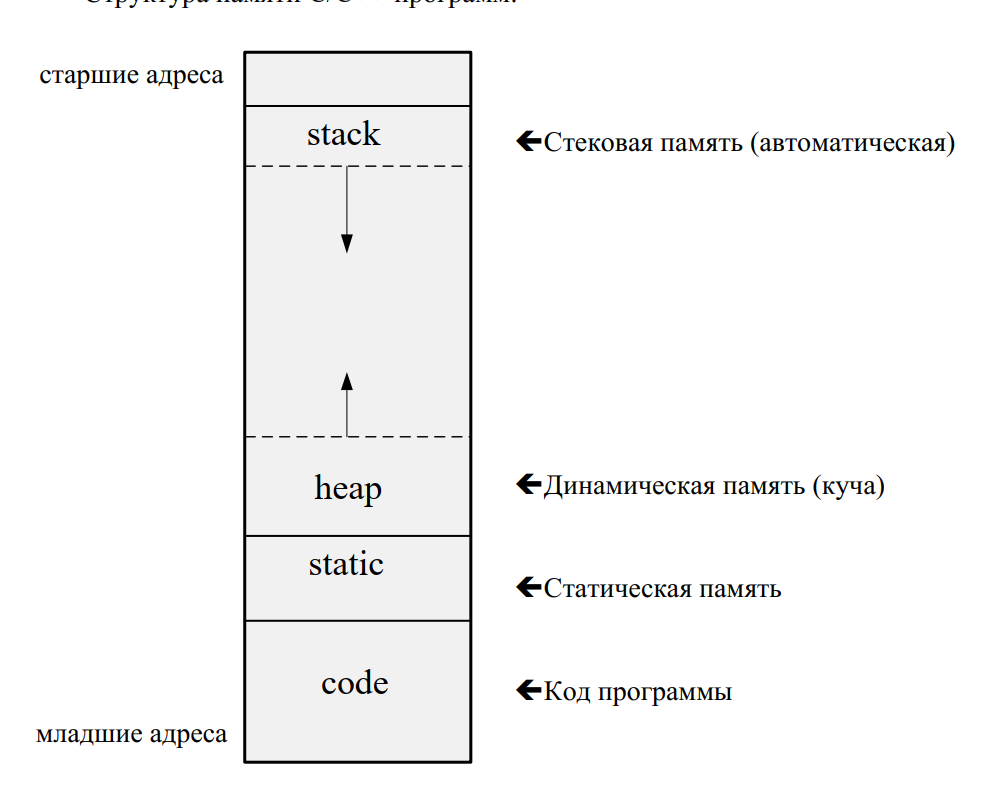
**7. Структура системы программирования. Модели памяти (классы памяти): код, статическая память, стек, динамическая память.**

*Состав системы программирования:*

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

Модель памяти языка С++ предоставляет области для хранения кода и данных.

Структура памяти C/C++-программ:



Область кода – память, в которой размещается код программы.

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.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.

Стековая память - предназначена для хранения локальных переменных.

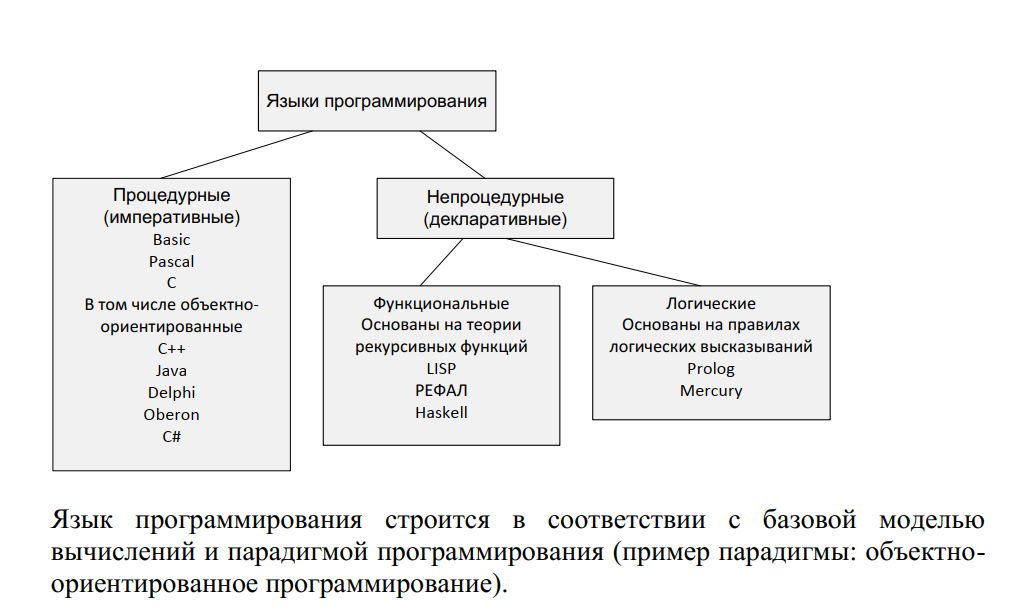
Локальные переменные хранятся в стеке.

Стек – это непрерывная область оперативной памяти, организованная по

принципу LIFO (последний вошел, первый вышел).

Динамическая память – это область памяти, выделение которой в языке программирования С++ производится с помощью оператора new, освобождение — оператором delete

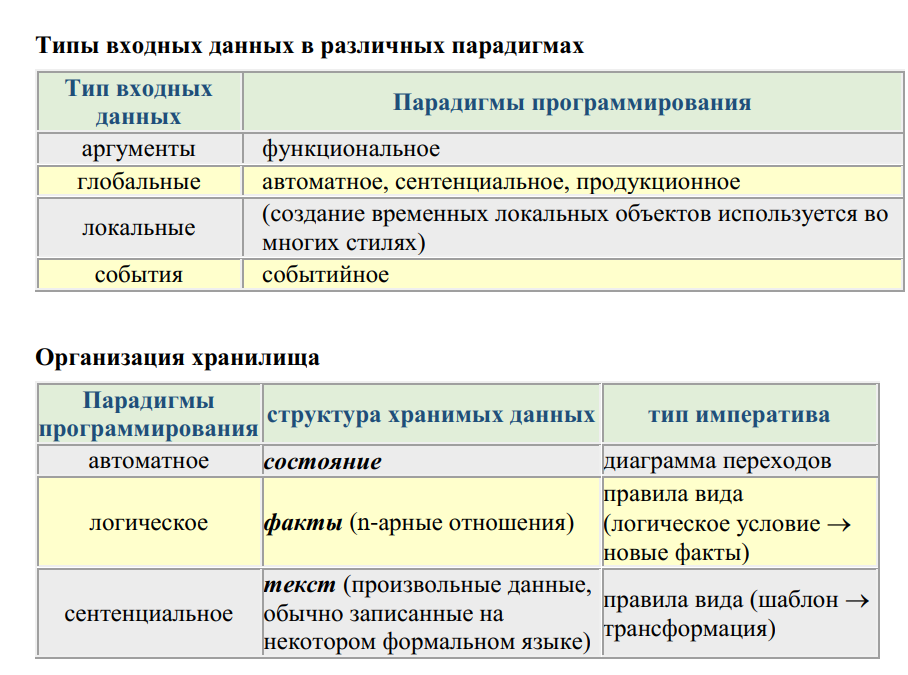
**8. Парадигмы программирования.**



Парадигма программирования – это совокупность идей и понятий,

определяющих стиль написания компьютерных программ (подход к

программированию).





| | императивное | | --- | | программа = последовательность действий, связанных  условными и безусловными переходами |
| --- | --- | --- |
| объектно-ориентированное | программа = несколько взаимодействующих объектов,  функциональность (действия) и данные  распределяются между этими объектами |
| функциональное | программа = система определений функций, описание  того, что нужно вычислить, а как это сделать – решает  транслятор; последовательность действий не  прослеживается |
| логическое | программа = система определений и правил вида  «условие => новый факт» |
| декларативное | Программа = описание действий, которые необходимо выполнить  компилятору для получения результата.  Отвечает на вопрос что надо выполнить. |
| | Визуальное | | --- | | Программа = способ создания программы для ЭВМ путём  манипулирования графическими объектами вместо написания её текста.  Пакет визуального программирования генерирует, написанный на языках  программирования (1GL, 2Gl, 3GL), на основании составленной  программистом «блок-схемы» в автоматическом режиме |

Структурное программирование – методология и технология разработки

программных средств, основанная на трёх базовых конструкциях:

* следование;
* ветвление;
* цикл.

Цели структурного программирования:

* повысить надежность программ (улучшить структуру программы);
* создание понятной, читаемой программы, которая исполняется,тестируется, конфигурируется, сопровождается и модифицируется без участия автора (создание ПП).

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.

Технология структурного программирования базируется на следующих

методах:

* нисходящее проектирование (формализация алгоритма «сверху вниз»: движение от общего к частному);
* пошаговое проектирование (нисходящая пошаговая детализация программы);
* структурное проектирование. Замена формулировки алгоритма на одну из синтаксических конструкций – последовательность, условие или цикл; программирование без goto
* одновременное проектирование алгоритма и данных ;
* модульное проектирование;
* модульное, нисходящее, пошаговое тестирование.

<цель\_результата> = <действие> + <цель\_результата вложенной конструкции>

Программирование без goto.

Использование операторов (вида continue, break, return) для более изменения

структурированной логики выполнения программы.

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

Язык программирования - это формальная знаковая система, предназначенная для записи компьютерных программ.

Язык программирования предназначен для написания компьютерных программ, которые представляют собой набор правил, позволяющих компьютеру выполнить тот или иной вычислительный процесс, организовать управление различными объектами, и т. п.( C, C++, Java, JS, Python, Pascal)

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.

Стандарт оформления кода - это набор соглашений, который принимается и используется некоторой группой разработчиков программного обеспечения для единообразного оформления совместно используемого кода.

Целью принятия и использования стандарта является упрощение восприятия программного кода человеком, минимизация нагрузки на память и зрение при чтении программы.

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 подчеркивания 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.

Вне стандарта подразумевается:

• отсутствие магических чисел;

• ограничение размера кода по горизонтали (чтобы помещался на экране) и вертикали (чтобы весь код файла держался в памяти);

• ограничение размера функции или метода – в размер одного экрана.

Рекомендации по стилю оформления кода в C++

* Пробелы и отступы

После зарезервированных ключевых слов языка C++ следует ставить пробел. Ставьте пробелы между операторами и операндами.

Отделяйте пробелами фигурные скобки, запятые и другие специальные символы.

Оставляйте пустые строки между функциями и между группами выражений.

* Именование

Основные правила стиля кодирования приходятся на именование.

Вид имени сразу же говорит нам что это: тип, переменная, функция, константа, макрос и т.д. Правила именования могут быть произвольными, однако важна их согласованность, и правилам нужно следовать.

Общие принципы именования

• используйте имена, который будут понятны даже людям из другой команды;

• имя должно говорить о цели или применении объекта;

• не экономьте на длине имени;

• не используйте аббревиатур; исключение: допускаются только известные аббревиатуры;

• не сокращайте слова.

Отметим, что типовые имена также допустимы: i для итератора или счётчика, T для параметра шаблона.

* Имена файлов

Имена файлов должны быть записаны только строчными буквами, для разделения можно использовать подчёркивание (\_) или дефис (-). Используйте тот разделитель, который используется в проекте.

* Имена типов

Имена пользовательских типов начинаются с прописной буквы, каждое новое слово также начинается с прописной буквы. Подчёркивания не используются.

* Имена переменных

Имена переменных пишутся строчными буквами, возможно с подчёркиванием между словами (в одном стиле):

line, lineAccount

! Следует инициализировать переменные в месте их объявления.

Префикс n следует использовать для представления количества объектов: nPoints, nLines // Обозначение взято из математики

Переменным-итераторам следует давать имена

i, j, k // и т.д.

* Имена констант

Объекты объявляются как const или constexpr, чтобы значение не менялось в процессе выполнения. Имена констант константы пишутся в верхнем регистре. Подчёркивание может быть использовано в качестве разделителя. OK, OUT\_OF\_MEMORY

Именованные константы должны быть записаны в верхнем регистре с нижним подчёркиванием в качестве разделителя.

* Имена функций

Названия методов и функций должны быть глаголами, быть записанными в смешанном регистре, начинаться с прописной буквы (в нижнем регистре) и каждое слово в имени пишется с прописной буквы:

getName(); computeAverage(); findNearestVertex();

* Именование пространства имён (namespace)

Пространство имен называется строчными буквами.

Пространство имён верхнего уровня – это обычно название проекта.

* Имена перечислений

Перечисления должны именоваться либо как константы, либо как макросы:

enum TableErrors {

OK = 0,

OUT\_OF\_MEMORY = 1,

SURPRISE = 2

};

* Чрезмерность

Если вы используете один и тот же код дважды или более раз, то найдите способ удалить излишний код, чтобы он не повторялся.

* Комментарии

Сложный код, написанный с использованием хитрых ходов, следует не комментировать, а переписывать!

Следует делать как можно меньше комментариев, делая код самодокументируемым путем выбора правильных имен и создания ясной логической структуры.

* Эффективность

Вызывая большую функцию и используя результат несколько раз, сохраните результат в переменной вместо того, чтобы постоянно вызывать данную функцию.

* Оформление

Основной отступ следует делать в два пробела

* Проектирование функции

Хорошо спроектированная функция имеет следующие характеристики:

• полностью выполняет четко поставленную задачу;

• не берет на себя слишком много работы;

• не связана с другими функциями бесцельно;

• хранит данные максимально сжато;

• помогает распознать и разделить структуру программы;

• помогает избавиться от лишней работы, которая иначе присутствовала бы в программе.

* Рекомендации

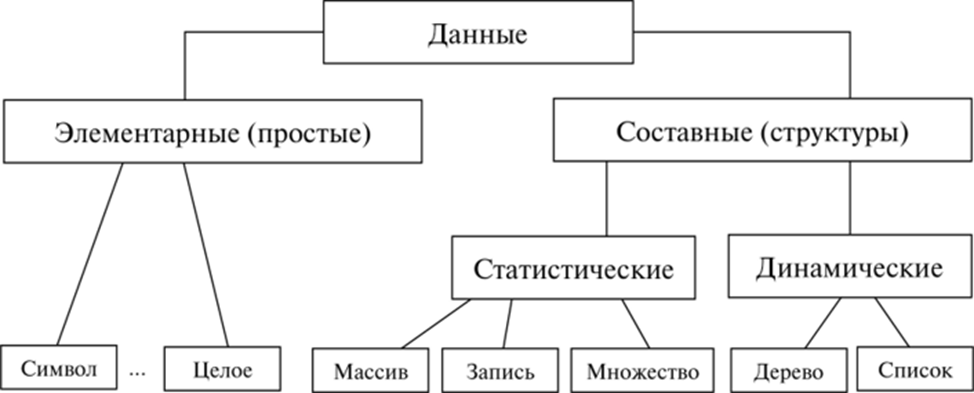
при использовании операторов управления (if / else, for, while, других) всегда используйте {} и соответствующие отступы, даже если тело оператора состоит лишь из одной инструкции

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

Кодирование информации (англ. information coding) — отображение данных на кодовые слова. Обычно в процессе кодирования информация преобразуется из формы, удобной для непосредственного использования, в форму, удобную для передачи, хранения или автоматической обработки.

Кодирование нужно для того, чтобы все, кто хочет, могли получать, передавать и работать с данными так, как им хочется. Благодаря кодированию мы можем обмениваться данными между собой — мы просто кодируем их в понятном для всех виде.

Данные — это любые значения, которые используются в работе программы: строки, числа, ссылки и символы



Данные, хранящиеся в памяти ЭВМ, представляют собой совокупность нулей и единиц (битов). Биты объединяются в последовательности: байты, слова и т.д. Каждому участку оперативной памяти, который может вместить один байт или слово, присваивается порядковый номер (адрес).

Какой смысл заключен в данных, какими символами они выражены — буквенными или цифровыми, что означает то или иное число — все это определяется программой обработки. Данные, необходимые для решения практических задач, подразделяются на несколько типов, причем понятие «тип» связывается не только с представлением данных в адресном пространстве, но и со способом их обработки.

Любые данные могут быть отнесены к одному из двух типов:

основному (простому) - форма представления которого определяется архитектурой ЭВМ

сложному - конструируемому пользователем для решения конкретных задач.

Данные простого типа — это символы, числа и т.п., элементы, дальнейшее дробление которых не имеет смысла. Из элементарных данных формируются структуры (сложные типы) данных. Некоторые структуры:

* Массив (функция с конечной областью определения) — простая совокупность элементов данных одного типа, средство оперирования группой данных одного типа. Отдельный элемент массива задается индексом.
* Запись (декартово произведение) — совокупность элементов данных разного типа. В простейшем случае запись содержит постоянное количество элементов, которые называют полями. Совокупность записей одинаковой структуры называется файлом. Для того чтобы иметь возможность извлекать из файла отдельные записи, каждой записи присваивают уникальное имя или номер, которое служит ее идентификатором и располагается в отдельном поле. Этот идентификатор называют ключом.

Такие структуры данных, как массив или запись занимают в памяти ЭВМ постоянный объем, поэтому их называют статическими структурами. К статическим структурам относится также множество.

Имеется ряд структур, которые могут изменять свою длину — так называемые динамические структуры. К ним относятся дерево, список, ссылка.

Важной структурой, для размещения элементов которой требуется нелинейное адресное пространство, является дерево. Существует большое количество структур данных, которые могут быть представлены как деревья. Это, например, классификационные, иерархические, рекурсивные и другие структуры

1. *Американский стандартный код для обмена информацией. ASCII*

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.

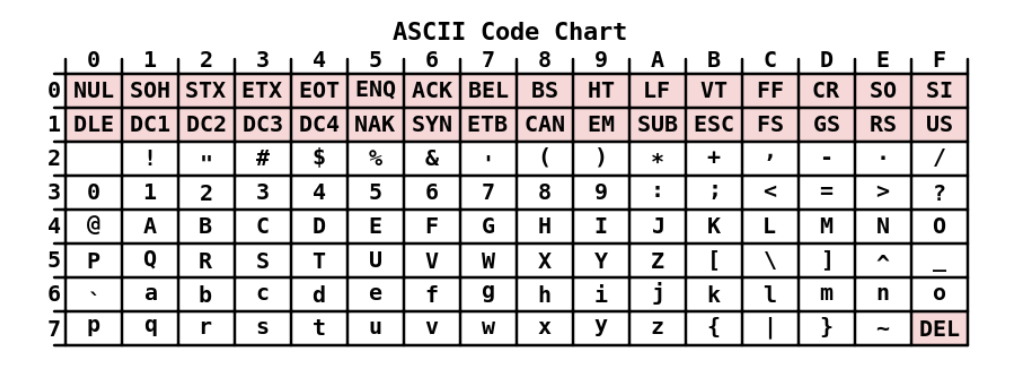
Таблица кодов ASCII делится на две части:

1. Международным стандартом является первая половина таблицы, т.е.

символы с номерами от 0 (00000000), до 127 (01111111).

1. 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.

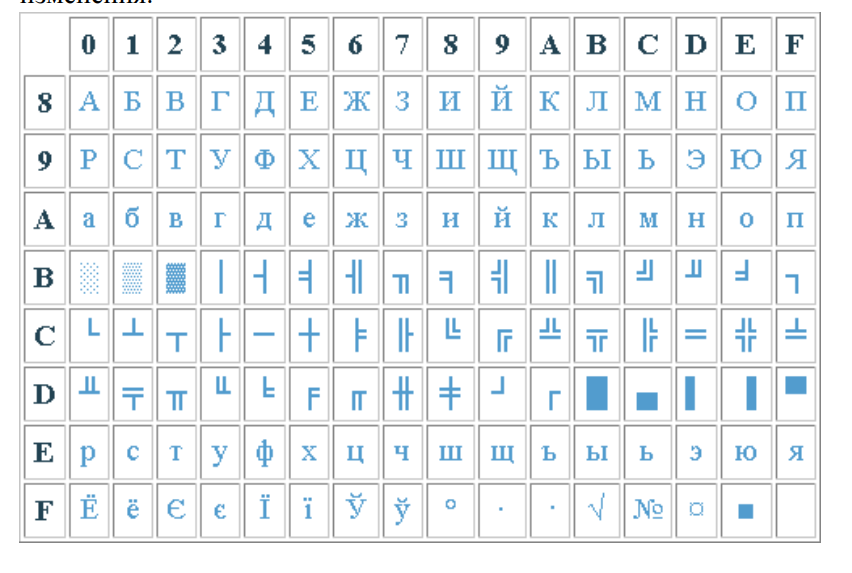
Переносимый набор символов (portable character set) – набор из 103 символов, которые (стандарт POSIX) должны присутствовать в любой используемой кодировке. Переносимый набор символов включает в себя все печатные символы US-ASCII и часть управляющих и является базовым алфавитом для практически всех современных языков программирования.

*2) кодировка CP866*

Все специфические европейские символы во второй половине таблицы CP866

заменены на кириллицу, а псевдографические символы оставлены без

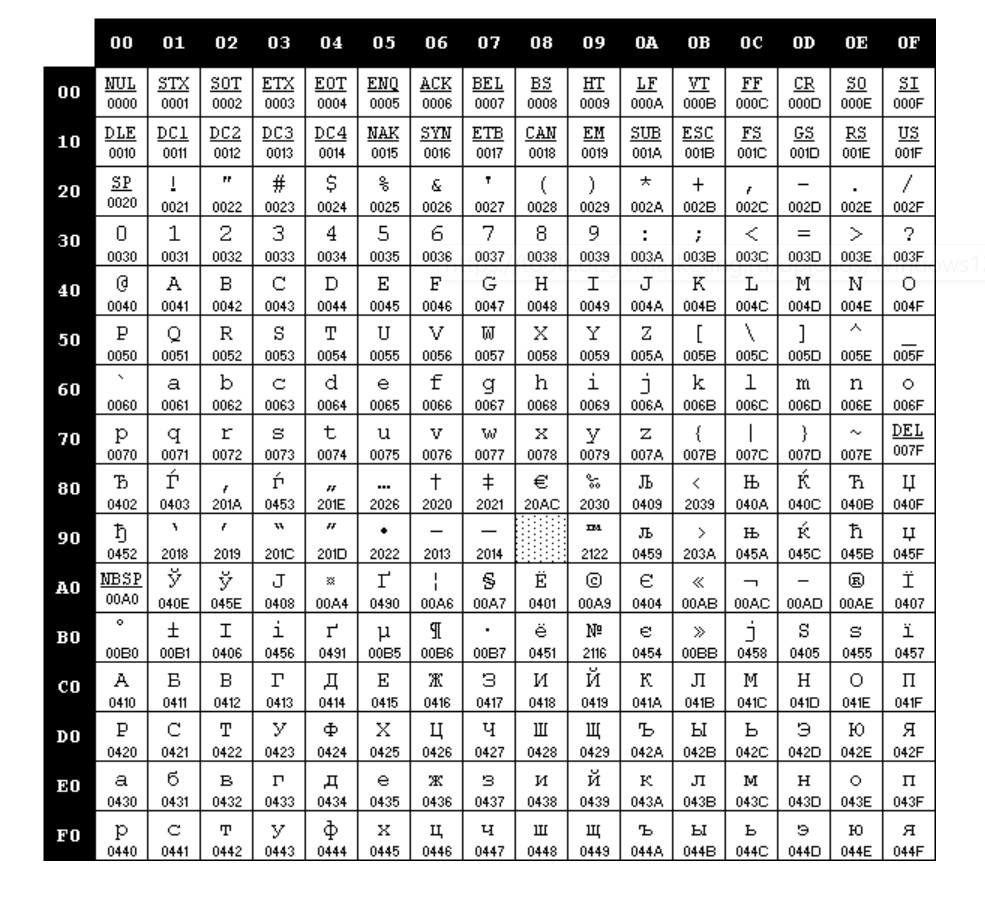
изменения



*3)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.



*4)UNICODE*

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 216 (65 536) символов;

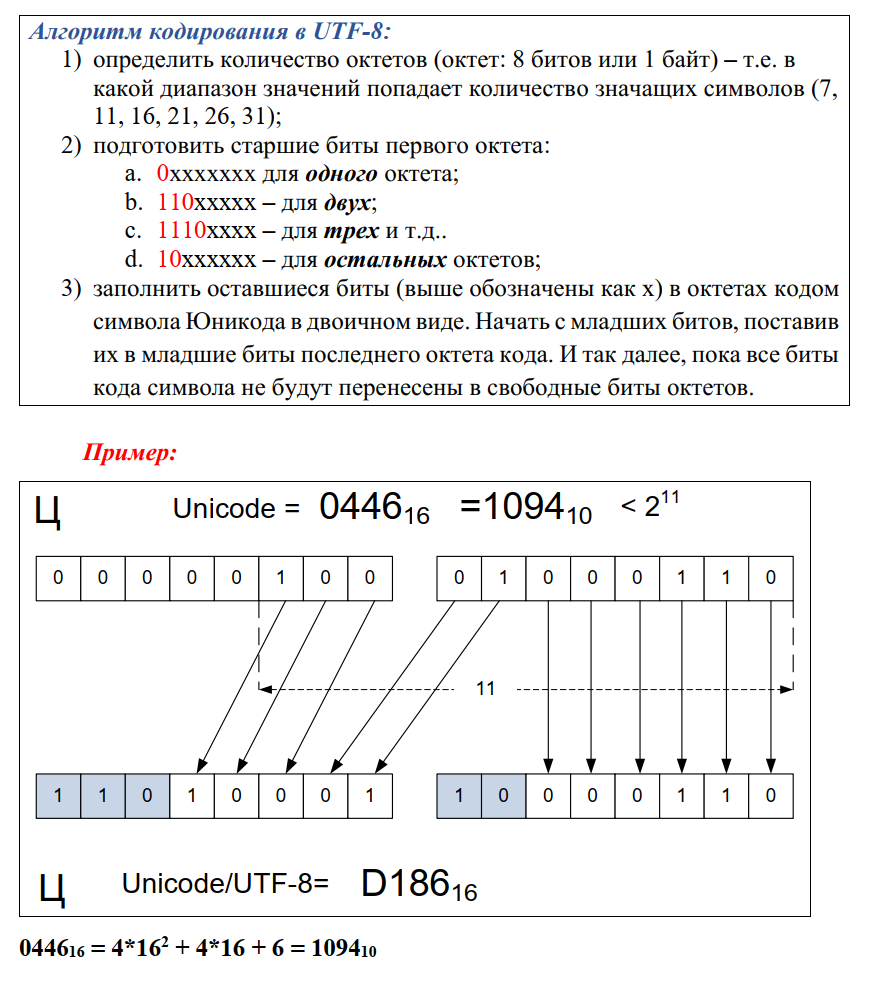
плоскость 0 – основная (основные символы);

1-14 – дополнительные;

15-16 – для частного использования

UTF-8 — представление Юникода, обеспечивающее совместимость со

старыми системами, использовавшими 8-битные символы.



В UTF-16 символы кодируются двухбайтовыми словами (16 битов) с

использованием всех возможных диапазонов значений (от 0 до FFFF16)

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.

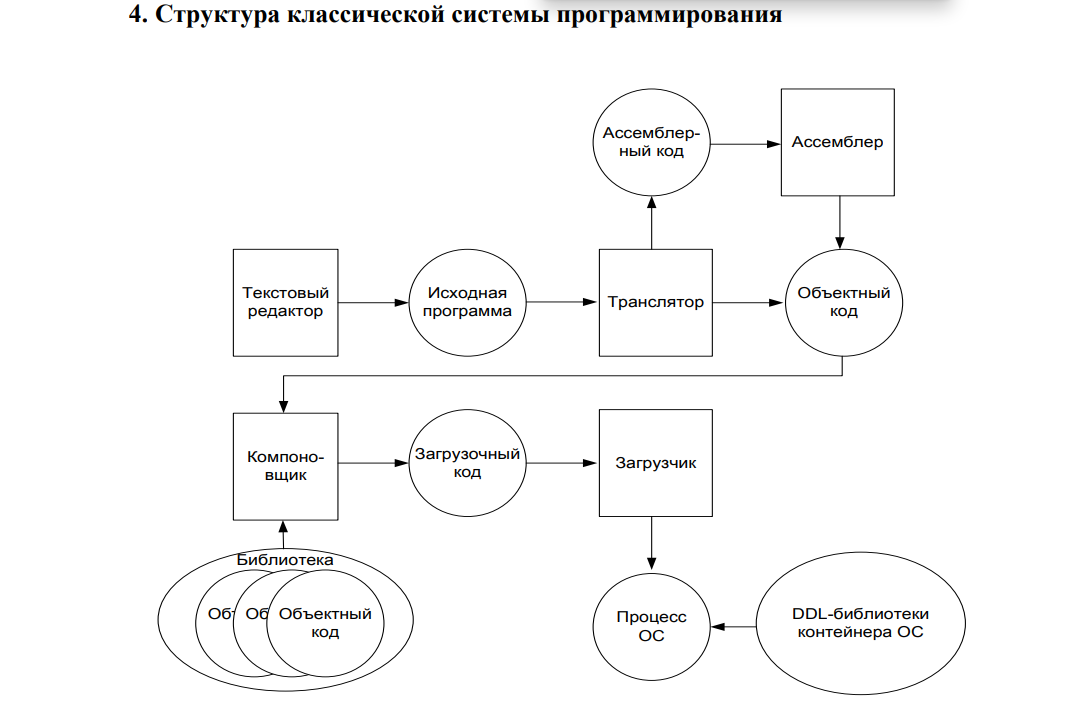
Шестнадцатеричное представление маркера последовательности байтов для

кодировок:



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

Система программирования – это комплекс инструментальных программных средств, предназначенный для автоматизации процесса разработки, отладки программного обеспечения и подготовки программного кода к выполнению.



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:

1. Международным стандартом является первая половина таблицы, т.е. символы с номерами от 0 (00000000), до 127 (01111111). К концу 1980-х годов стандартом стали 8-битные кодировки.
2. Расширенные таблицы: в расширенных таблицах символы с порядковыми номерами 128-255 представляют символы национальных языков.

Windows-1251 (русская Windows-кодировка, синоним CP1251) ) — набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий Microsoft Windows до 10-й версии.

Международный стандарт UNICODE предназначена для решения следующих проблем: -неправильного декодирования;

-ограниченность набора символов;

-преобразования из одной кодировки в другую;

-проблема дублирования шрифтов.

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).

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

Кодирование — это преобразование информации из одной ее формы представления в другую, наиболее удобную для её хранения, передачи или обработки. Кодом называют правило отображения одного набора знаков в другом. Назначение кодирования: упрощение и оптимизация работы с информацией.

Примеры: Двоичный код, используемый компьютером, азбука морзе, сигналы регулировщика.

Международный стандарт UNICODE предназначена для решения следующих проблем: неправильного декодирования;

ограниченность набора символов;

преобразования из одной кодировки в другую;

проблема дублирования шрифтов.

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

UCS – universal character set (универсальный набор символов);

Универсальный набор символов перечисляет допустимые по стандарту Юникод символы и присваивает каждому символу код в виде неотрицательного целого числа, записываемого обычно в шестнадцатеричной форме с префиксом U+, например, U+040F.

UTF – Unicode transformation format (семейство кодировок).

Семейство кодировок определяет способы преобразования кодов символов для передачи в потоке или в файле.

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

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

BOM. Маркер последовательности байтов - специальный символ из стандарта Юникод, вставляемый в начало текстового файла или потока для обозначения того, что в файле используется Юникод, а также для косвенного указания кодировки и порядок байтов, с помощью которых символы Юникода были закодированы.

Номер этого символа в стандарте Юникод — U+FEFF. Использование этого символа, согласно спецификации Юникод, не является обязательным, однако оно широко распространено, так как позволяет легко избежать неверного декодирования текстовой информации.

Пример:

Порядок следования байтов:

LE (Little endian order, прямой порядок, от младшего к старшему) - наименее значимый (самый младший) байт хранится первым, а за ним следуют байты в порядке возрастания значимости.

BE (Big endian order, обратный порядок, от старшего к младшему) - наиболее значимый (самый старший) байт хранится первым, а за ним идут байты в порядке убывания значимости.



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

Программа – это объект разработки, который не является осязаемым (нельзя

пощупать, взвесить и т. п.), доступен пониманию ЭВМ, для которой написан.

Этапы и цели разработки программы:

1. Постановка задачи.

-определение функциональных возможностей программы;

-подготовка технического задания.

2. Выбор метода решения.

-определение исходных и выходных данных, ограничений на них;

-выполнение формализованного описания задачи;

-построение математической модели, для решения на компьютере.

3. Разработка алгоритма решения задачи.

-выполняется на основе ее математического описания;

-полное и точное описание, определяющее вычислительный процесс, ведущий от начальных данных к искомому результату.

4. Написание программы на языке программирования (кодирование)

-запись алгоритма на языке программирования.

5. Ввод программы в компьютер

-подготовка исходного кода программы в виде текстового, который поступает на вход транслятора.

6. Трансляция

-преобразование исходного кода с одного языка программирования в семантически эквивалентный код на другом языке;

-получение объектного модуля.

7. 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.

8. Выполнение

-выполнение исполняемого файла программы на целевой машине.

9. Тестирование

-устранение ошибок в программе.

10. Отладка

-обнаружение, локализация и устранение ошибок.

11. Документирование

-создание пользовательской документации.

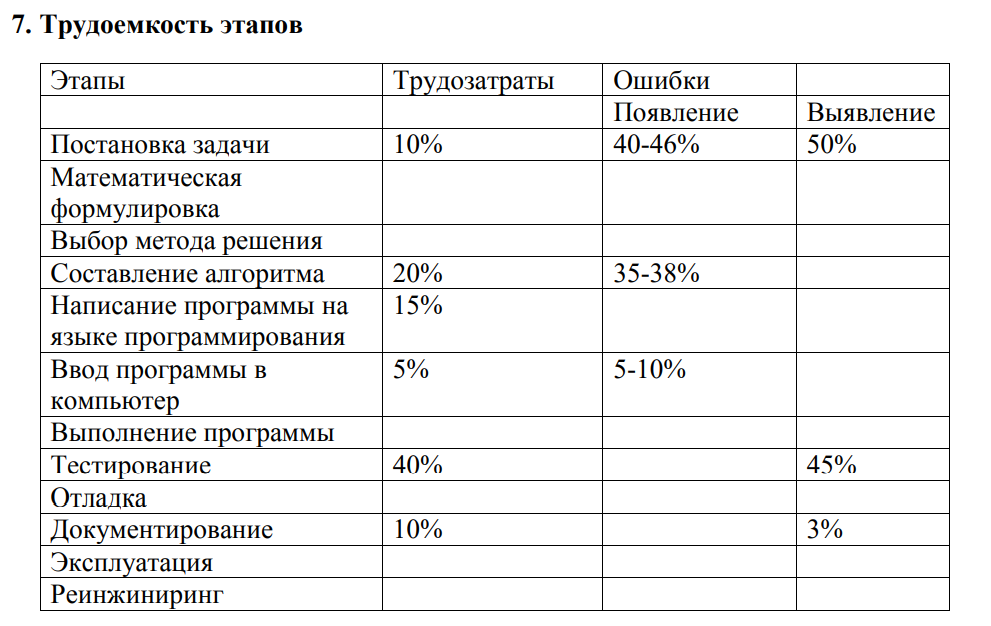
12. Эксплуатация

-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. 

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

Программа – это объект разработки, который не является осязаемым (нельзя

пощупать, взвесить и т. п.), доступен пониманию ЭВМ, для которой написан.

Алгоритм – совокупность точно заданных правил, с помощью которой можно получить решение задачи за конечное число шагов.

Назначение: описать этапы процесса/программы

Свойства:

-дискретность (возможность разбиения на шаги);

-понятность (ориентирован на исполнителя);

-определенность (однозначность толкования инструкций);

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

-массовость (применимость к некоторому классу объектов);

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

Способы описания:

-словесно-формульный (на естественном языке);

-графический (структурный или блок-схемой);

-использование псевдокода (специальных алгоритмических языков);

-с помощью сетей Петри;

-программный.

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

Системы программирования – системные программы, предназначенные для автоматизации процесса разработки, отладки программного обеспечения и подготовки программного кода к выполнению, является основным инструментом программиста.

Системы программирования Microsoft:

Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки ПО и другие инструменты. Для разработки консольных приложений, игр, приложений с графическим интерфейсом и т.д.

Unix – семейство многозадачных и многопользовательских операционных систем, которые характеризуются модульным дизайном, в котором каждая задача выполняется отдельной утилитой.

Linux – семейство Unix-подобных операционных систем на базе ядра Linux, включающих набор утилит и программ проекта GNU (проект по разработке свободного программного обеспечения). Linux-системы реализуются на модульных принципах, заложенных в Unix.

Системы программирования Linux:

1. [Bluefish](http://bluefish.openoffice.nl/) — интегрированная среда разработки, фокусируется на создании динамических и интерактивных веб-сайтов.
2. [Anjuta](http://projects.gnome.org/anjuta/) — интегрированная среда разработки для языков Си и C++.
3. [GCC](http://gcc.gnu.org/) — это набор компиляторов для Си, C++, Java и Ады, разработанный в рамках проекта GNU.

IBM. International Business Machines – американская корпорация, один из крупнейших мировых производителей всех видов компьютеров и программного обеспечения.

Системы программирования IBM:

1. IBM VisualAge — семейство [сред разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8_%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).

Стандарты языков программирования:

Язык программирования обычно представляется в виде набора спецификаций, определяющих его синтаксис и семантику. Для многих широко распространенных языков программирования созданы международные стандарты.

Стандарт POSIX — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой, библиотеку языка C и набор приложений и их интерфейсов.

Парадигмы программирования - вопрос 8

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

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

Примеры IDE:

JDeveloper — интегрированная среда разработки ПО на языках программирования Java, JavaScript, PHP и т.д.

NetBeans — интегрированная среда разработки ПО на языках программирования Java, C, C++ и др.

Eclipse — интегрированная среда разработки модульных кроссплатформенных приложений.

Oracle Sun Studio — интегрированная среда разработки для языков программирования Си, C++ и Фортран.

Microsoft Visual Studio — это линейка продуктов компании Microsoft, включающих интегрированную среду разработки ПО и другие инструменты. Используются для разработки консольных приложений, игр, приложений с графическим интерфейсом и т.д. Разработана в 1997 года, написана на C++ и C#.

Назначение. Среда разработки нужна для облегчения, упрощения процесса разработки, предоставляет программистам средства для разработки программного обеспечения. IDE облегчает визуальное представление файлов и делает его более понятным для пользователя.

Основные возможности:

-Отладка

-Модульное тестирование

-Профилирование кода

-Интеграция исходного кода

-Интеграция инструментов разработки и выпуска

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

Microsoft Visual Studio — это линейка продуктов компании Microsoft, включающих интегрированную среду разработки ПО и другие инструменты. Используются для разработки консольных приложений, игр, приложений с графическим интерфейсом и т.д. Разработана в 1997 года, написана на C++ и C#.

Назначение: разработка и отладка программ.

Visual Studio включает в себя:

* Редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода.
* Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя
* редактор форм для упрощения создания графического интерфейса приложения,
* веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe),
* добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

Возможности:

* Разработка. Быстрые навигация и исправление кода
* Сотрудничество. Использование системы управления версиями для гибкой и эффективной работы.
* Отладка. Простая отладка и диагностика кода
* Расширение. Возможность настройки интегрированной среды разработки под себя
* Тестирование. Комплексные инструменты позволяют написать высококачественный код

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

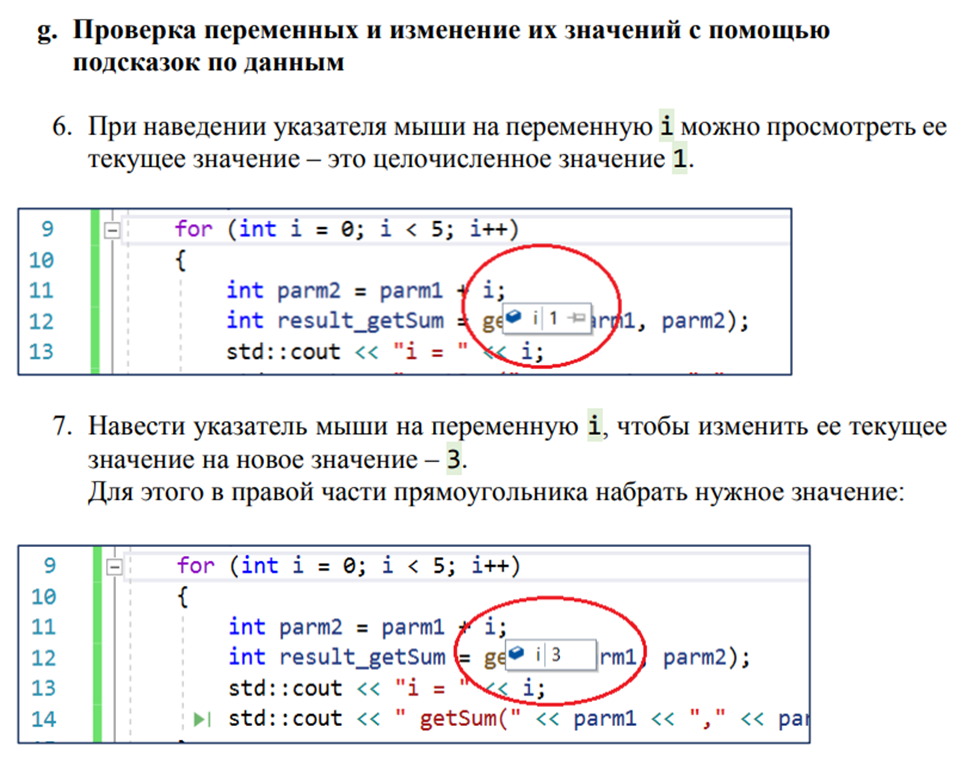
Отладчик– инструментальное средство разработки программ, которое присоединяется к работающему приложению и имеет такие возможности, как проверка кода, наблюдение за выполнением исследуемой программы, ее остановка и перезапуск, изменение значения в памяти, просмотр стека вызовов (список всех активных функций, которые вызывались, до текущей точки выполнения исходного кода) и т.д.

Назначение отладчика – устранение ошибок в коде программы

Точка останова – это точка, в которой процесс выполнения программы приостанавливается и отладчик получает управление. Если точка останова не установлена, то отладчик запускается и выполняет приложение целиком.

После того, как мы остановим программу, мы можем изучить значения локальных переменных процедуры, в которой произошла остановка выполнения приложения, а также проанализировать стек вызовов, предшествующих вызову данной процедуры. Здесь же мы можем изменить значения этих переменных

В меню - Отладка -> Окна -> Локальные - показаны значения локальных переменных





Дизассе́мблер — транслятор, который преобразует программу на машинном языке к ее ассемблерному представлению

Чаще всего дизассемблер используют для анализа программы, исходный текст которой неизвестен. Реже — для поиска ошибок в программах и компиляторах.

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

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

Структурное программирование – методология разработки программных средств без оператора безусловного перехода goto, основанная на трёх базовых конструкциях:

- следование;

- ветвление;

- цикл.

В структурном программировании необходимо составить правильную логическую схему программы и реализовать ее средствами языка программирования;

Структурное программирование характеризуется:

- ограниченным использованием условных и безусловных переходов;

- широким использованием подпрограмм и управляющих структур (циклов, ветвлений, и т.п.);

- блочной структурой.

Этапы структурного проектирования:

1.Формулировка цели алгоритма на естественном языке.

2.Создание модели процесса, используются графические способы представления.

3.Сбор информации, касающейся характеристик алгоритма.

4.Выделение наиболее существенной цели.

5.Определение переменных, необходимых для формального представления алгоритма и их назначение.

6.Выбор конструкции - простая последовательность действий, условная конструкция или цикл.

7.Для вложенных конструкций необходимо вернуться на предыдущие этапы проектирования.

Технология структурного программирования базируется на следующих методах:

- нисходящее проектирование (формализация от самой внешней синтаксической конструкции алгоритма к самой внутренней);

- пошаговое проектирование (на каждом этапе проектирования в текст программы вносится только одна конструкция языка);

- структурное проектирование (программирование без goto);

- одновременное проектирование алгоритма и данных (процессы детализации алгоритма и данных идут параллельно);

- модульное проектирование;

- модульное, нисходящее, пошаговое тестирование.

Сущность структурного подхода к разработке ПО заключается в ее декомпозиции (разбиении) на автоматизируемые функции:

Процесс разбиения системы продолжается вплоть до конкретных процедур.

Методы структурного программирования

* Метод алгоритмической декомпозиции сверху вниз — заключается в пошаговой детализации постановки задачи, начиная с наиболее общей задачи.
* Метод модульной организации частей программы — заключается в разбиении программы на специальные компоненты, называемые модулями.
* Метод структурного кодирования — заключается в использовании при кодировании трех основных управляющих конструкций (последовательное исполнение, ветвление, циклы).

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

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

Мо́дульное программи́рование — 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.

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.

Метод восходящей разработки. При восходящем проектировании разработка идет снизу-вверх. На первом этапе разрабатываются модули самого низкого уровня. На следующем этапе к ним подключаются модули более высокого уровня и проверяется их работоспособность. На завершающем этапе проектирования разрабатывается головной модуль, отвечающий за логику работы всего программного комплекса.

Нисходящее проектирование программы состоит в процессе формализации от самой внешней синтаксической конструкции алгоритма к самой внутренней; в движении от общей формулировки алгоритма к частной формулировке, составляющей его действия;

**21. Системы контроля версий. Классификация. Назначение, разновидности систем контроля версий. Система контроля версий Git: основные возможности. Отличия Git от других систем контроля версий.**

Система контроля версиями (от англ. VersionControl System, VCS) – программное обеспечение для облегчения работы с изменяющейся информацией и разработки проекта совместно с коллегами.

1.Локальные системы контроля версий

Многие люди в качестве метода контроля версий выбирают простое копирование файлов в другую папку. Этот подход крайне популярен в силу своей простоты, но при этом он потрясающе ненадежен. Легко забыть, в какой папке вы работаете, и случайно сделать запись не в тот файл или скопировать вовсе не те файлы, которые вы хотели. Для урегулирования этого вопроса программисты давно изобрели локальные системы контроля версий, снабженные простой базой данных, хранящей всю информацию об изменениях файлов.

Одной из наиболее популярных локальных систем контроля версий была система контроля редакций (Revision Control System, RCS), которая до сих пор работает на многих компьютерах. RCS сохраняет на диске в специальном формате набор всех внесенных изменений (то есть разницу между файлами). В будущем это позволяет воссоздать любой файл в любой момент времени, добавив к нему все изменения.

2.Централизованные системы контроля версий

Следующая большая проблема — необходимость сотрудничать с разработчиками других систем. Для ее решения были разработаны централизованные системы контроля версий (Centralized Version Control System, CVCS). Они обладают единым сервером, содержащим все версии файлов и набором клиентов, выгружающих файлы с сервера. На протяжении многих лет это было стандартом управления версиями.

Такая схема имеет много преимуществ, особенно перед локальными системами контроля версий. Например, каждый человек, работающий над проектом, до определенной степени знает, чем занимаются его коллеги.

Однако есть у этой схемы и серьезные недостатки. Самым очевидным является единая точка отказа, представленная центральным сервером. Отключение этого сервера на час означает, что в течение часа любые взаимодействия невозможны. При повреждении жесткого диска центральной базы данных и отсутствии нужных резервных копий теряется вся информация — вся история разработки проекта за исключением единичных снимков состояния, которые могут остаться на локальных компьютерах пользователей.

3.Распределенные системы контроля версий

Именно здесь на первый план выходят распределенные системы контроля версий (Distributed Version Control System, DVCS). В DVCS (к примеру, Git) клиенты не просто выгружают последние снимки файлов, они создают полную зеркальную копию репозитория(место, где хранятся и поддерживаются какие-либо данные). Соответственно в случае выхода работы из строя одного из серверов его работоспособность можно восстановить, скопировав один из клиентских репозиториев. Каждая такая выгрузка сопровождается полным резервным копированием всех данных.

Git - распределённая система управления версиями. Она работает необыкновенно быстро, крайне эффективна для больших проектов и обладает потрясающей ветвящейся системой нелинейной разработки.

Отличия Git от других систем контроля версий:

1.Снимки состояний, а не изменений

Главным отличием Git от любой другой системы контроля версий является восприятие данных. Большинство систем хранит информацию в виде списка изменений, связанных с файлами. Эти системы рассматривают хранимые данные как набор файлов и изменений, которые вносились в эти файлы в течение их жизни.

Система Git не воспринимает и не хранит файлы подобным образом. В ее восприятии данные представляют собой набор снимков состояния файловой системы. Каждый раз, когда вы создаете новую версию или сохраняете состояние проекта в Git, по сути, делается снимок всех файлов в конкретный момент времени и сохраняется ссылка на этот снимок. Для повышения продуктивности вместо файлов, которые не претерпели изменений, сохраняется всего лишь ссылка на их ранее сохраненные версии.

2.Локальность операций

Для осуществления практически всех операций системе Git требуются только локальные файлы и ресурсы — в общем случае информация с других компьютеров сети не нужна. Когда вся история проекта хранится на локальном диске, кажется, что большинство операций выполняется почти мгновенно.

Например, для просмотра истории проекта системе Git нет нужды обращаться к серверу, получать там историю и выводить ее на экран — система просто читает все непосредственно из локальной базы данных. Если вы хотите посмотреть, чем текущая версия файла отличается от версии месячной давности, Git ищет старый файл и вычисляет внесенные в него правки, вместо того чтобы просить об этой операции удаленный сервер или считывать с этого сервера старую версию файла для локального сравнения.

3.Целостность Git

В системе Git для всех данных перед сохранением вычисляется контрольная сумма, по которой они впоследствии ищутся. То есть сохранить содержимое файла или папки таким образом, чтобы система Git об этом не узнала, невозможно. Эта функциональность встроена в Git на самом низком уровне и является неотъемлемым принципом ее работы. Невозможно потерять информацию или повредить файл скрытно от Git.

По сути, Git сохраняет данные в базе не по именам файлов, а по хешу их содержимого.

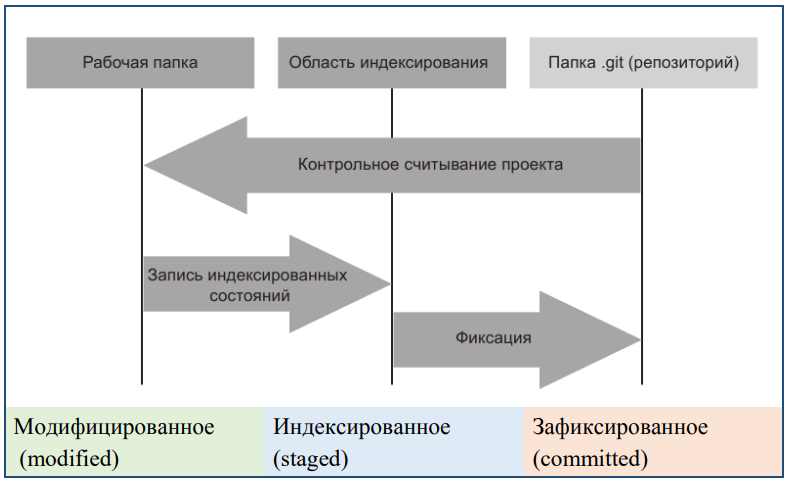
**22. Три состояния файлов в Git. Сохранение изменений в репозитории. Основные определения и основные команды Git.**

Файлы в Git могут находиться в трех основных состояниях: модифицированном, индексированном и зафиксированном.

Модифицированное (modified) состояние означает, что изменения уже внесены в файл, но пока не зафиксированы в базе данных.

Индексированное (staged) состояние означает, что вы пометили текущую версию модифицированного файла как предназначенную для следующей фиксации.

Зафиксированное (committed) состояние означает, что данные надежно сохранены в локальной базе.



В результате Git-проект разбивается на три основные области: папка Git, рабочая папка и область индексирования.

Рабочая папка — это место, куда выполняется выгрузка одной из версий проекта. Эти файлы извлекаются из сжатой базы данных в папке Git и помещаются на жесткий диск вашего компьютера, готовые к использованию или редактированию.

Область индексирования — это файл, обычно находящийся в папке Git и хранящий информацию о том, что именно войдет в следующую операцию фиксации. Иногда ее еще называют промежуточной областью.

Папка Git — это место, где Git хранит метаданные и объектную базу данных проекта. Это наиболее важная часть Git, которая копируется при дублировании репозитория (хранилища) с другого компьютера.

Базовый рабочий процесс в Git выглядит так:

1. Вы редактируете файлы в рабочей папке.

2. Вы индексируете файлы, добавляя их снимки в область индексирования.

3. Вы выполняете фиксацию, беря файлы из области индексирования и сохраняя снимки в папке Git.

Система Git поставляется с инструментом **git config**, позволяющим получать и устанавливать переменные конфигурации (совокупность настроек программы, задаваемая пользователем).

При установке Git первым делом следует указать имя пользователя и адрес электронной почты. Это важно, так как данную информацию Git будет включать в каждую фиксируемую вами версию, и она обязательно включается во все создаваемые вами коммиты (зафиксированные данные):

| $ git config --global user.name "John Doe"  $ git config --global user.email [johndoe@example.com](mailto:johndoe@example.com) |
| --- |

Передача параметра **--global** позволяет сделать эти настройки всего один раз, так как в этом случае Git будет использовать данную информацию для всех ваших действий в системе.

Проверить выбранные настройки позволяет команда **git config --list**, выводящая список всех обнаруженных в текущий момент параметров.

| $ git config --list  user.name=John Doe  user.email=johndoe@example.com  color.status=auto  color.branch=auto  color.interactive=auto |
| --- |

Существует три способа доступа к странице со справочной информацией по любой Git-команде:

| **$ git help**  $ git <команда> --help  $ man git- <команда> |
| --- |

Инициализация репозитория в существующей папке

Чтобы начать слежение за существующим проектом, перейдите в папку этого проекта и введите команду **$ git init** (создание репозитория).

В результате в существующей папке появится еще одна папка с именем .git и всеми нужными вам файлами репозитория — это будет основа вашего Git-репозитория. Чтобы начать управление версиями существующих файлов (в противовес пустому каталогу), укажите файлы, за которыми должна следить система, и выполните первую фиксацию изменений. Для этого потребуется команда **git add**, добавляющая файлы, за которыми вы хотите следить, а затем команда **git commit** (фиксация изменений):

| $ git add LICENSE  $ git commit -m 'первоначальная версия проекта' |
| --- |

Запись изменений в репозиторий

Теперь в файлы можно вносить изменения и фиксировать их, как только проект достигнет состояния, которое вы хотели бы сохранить. Помните, что каждый файл в рабочей папке может пребывать в одном из двух состояний: отслеживаемом и неотслеживаемом.

Отслеживаемое — это файлы, входящие в последний снимок системы; они могут быть неизмененными, измененными и подготовленными к фиксации.

Неотслеживаемое — это все остальные файлы рабочей папки, не вошедшие в последний снимок системы и не проиндексированные для последующей фиксации.

Основным инструментом определения состояния файлов является команда git status. Команда **git status** дает исчерпывающий, хотя и многословный результат. Но в Git существует флаг, позволяющий получить сведения в более компактной форме. Запустив команду **git status -s** или git status --short, вы получите упрощенный вариант вывода.

Бывает так, что некоторый класс файлов вы не хотите ни автоматически добавлять в репозиторий, ни видеть в списке неотслеживаемых. В эту категорию, как правило, попадают автоматически генерируемые файлы, например журналы регистрации или файлы, генерируемые системой сборки. В подобных случаях создается файл **.gitignore**

Чтобы система Git перестала работать с файлом, его нужно удалить из числа отслеживаемых (точнее, убрать из области индексирования) и зафиксировать данное изменение. Это делает команда **git rm**, которая заодно удаляет указанный файл из рабочей папки, благодаря чему он исчезает из списка неотслеживаемых.

| $ git rm file1.txt  $ git commit -m "remove file1.txt" |
| --- |

В Git есть команда mv. Для переименования файла можно написать:

| $ **git mv** file\_from file\_to |
| --- |

После сохранения нескольких версий файлов или клонирования уже имеющего содержимое репозитория вы, скорее всего, захотите взглянуть на то, что было сделано ранее. Базовым и самым мощным инструментом в данном случае является команда **git log**. По умолчанию при отсутствии параметров команда git log выводит в обратном хронологическом порядке список сохраненных в данный репозиторий версий.

| **$mkdir folder.name** | Создать папку |
| --- | --- |
| **$cd folder.name** | Перейти в папку |
| **$cd -** | Вернуться назад |
| **$touch file.name** | Создать файл |

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

Ветка в Git представляет собой подвижный указатель на один из коммитов. По умолчанию в Git ей присваивается имя master. Как только вы начинаете создавать коммиты, появляется ветка master, указывающая на последнее зафиксированное вами состояние. При каждой следующей фиксации этот указатель автоматически смещается вперед.

Что происходит при создании новой ветки? Появляется новый указатель, который можно перемещать. Предположим, вы создаете ветку testing. Эта операция выполняется командой **git branch:**

| $ git branch testing |
| --- |

Появится новый указатель на ваш текущий коммит.

Откуда Git узнает, какой именно коммит является текущим? Для этого он хранит специальный указатель HEAD. В Git он указывает на локальную ветку, в которой вы находитесь в данный момент. Команда git branch всего лишь создала новую ветку, но не перевела вас в нее. Узнать, куда именно нацелены указатели веток, позволяет команда **git log**.

Команда **git branch** позволяет не только создавать и удалять ветки. Запущенная без аргументов, она выводит на экран список имеющихся веток.

Переход на существующую ветку реализует команда **git checkou**t. Перейдем на ветку testing:

| $ git checkout testing |
| --- |

Теперь указатель HEAD нацелен на ветку testing.

История проекта разветвилась. Вы создали ветку, перешли в нее, поработали немного, вернулись в основную ветку и снова произвели некие действия. Внесенные в этих двух случаях изменения оказались принадлежащими разным веткам: вы можете переходить из одной ветки в другую, а при необходимости объединять их. И все это при помощи простых команд branch, checkout и commit. Суть происходящего демонстрирует и команда git log.

Можно объединить созданную ветку с веткой master, чтобы внедрить внесенные изменения в готовый код. Эта операция выполняется командой **git merge**(слияние):

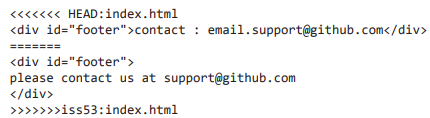
| $ git checkout master  $ git merge hotfix |
| --- |

Конфликты при слиянии

Процесс слияния далеко не всегда проходит гладко. Если в двух ветках, которые вы собираетесь слить, вы внесли разные изменения в один и тот же файл, Git не сможет просто взять и объединить их.

| CONFLICT (content): Merge conflict in index.html |
| --- |

В этом случае Git не может автоматически создать коммит слияния. Система приостанавливает процесс до момента разрешения конфликта. Посмотреть, какие файлы не прошли слияние после возникновения конфликта, позволяет команда **git status**. Все, что относится к области конфликта слияния, помечено как неслитое (unmerged). Система Git добавляет к проблемным файлам стандартные метки, позволяющие открывать эти файлы вручную и разрешать конфликты. Ваш файл содержит раздел, который выглядит примерно вот так:



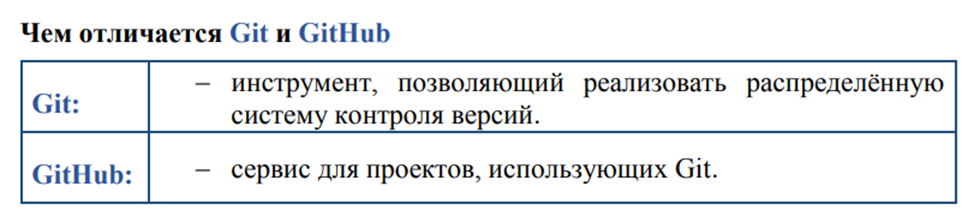
Версия с указателем HEAD (из вашей ветки master, так как именно в нее вы перешли перед выполнением команды merge) располагается в верхней части блока (то есть выше набора символов =======), а версия из ветки iss53 показана в нижней части. Для разрешения конфликта следует или выбрать одну из версий, или каким-то образом объединить их.

Разобравшись с каждым таким разделом в каждом из проблемных файлов, выполните для каждого из этих файлов команду **git add.** Индексируя файл, вы помечаете его как неконфликтующий. Остается завершить слияние командой **git commit.**

В отличие от централизованных систем контроля версий (CVCS), Git благодаря своему распределенному характеру допускает большую гибкость при совместной работе над проектами. В централизованных системах всех разработчиков можно представить в виде узлов сети, каждый из которых делает свою часть работы на центральном концентраторе. В Git же каждый разработчик может выступать как в роли узла, так и в роли концентратора. То есть он может вносить код в чужие репозитории и поддерживать открытый репозиторий. Такой подход дает рабочей группе широкие возможности.

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

Сайт GitHub является крупнейшим на сегодняшний день хостингом для Git-репозиториев и центральным местом сотрудничества миллионов разработчиков и проектов.



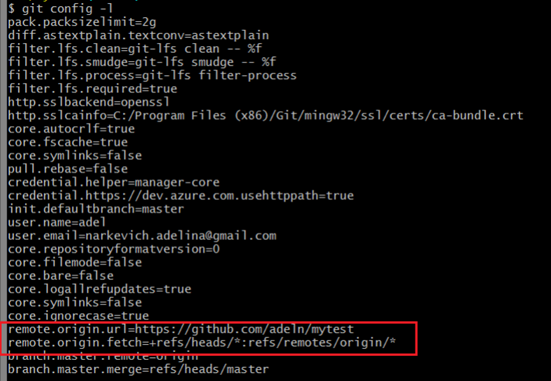
Сайт GitHub спроектирован под определенную схему работы, сконцентрированную вокруг запросов на включение. Она применяется как сплоченными командами, пользующимися общим репозиторием, так и распределенными группами и даже группами не знакомых друг с другом людей, добавляющих свою работу к проекту на десятках ветвлений.

Клонирование существующего репозитория

Для получения локальной копии существующего Git-репозитория нужно использовать команду **git clone**. Команда git clone по умолчанию забирает все версии всех файлов за всю историю проекта. Это означает, что при повреждении серверного диска практически любой клон на любом из клиентов может использоваться для возвращения сервера в состояние, в котором он пребывал до момента клонирования.

| $ git clone https://github.com/adeln/mytest |
| --- |

Просмотр конфигурационного файла: **git config -l**



Просмотр уже настроенных удаленных серверов осуществляется командой **git remote.** Она дает список коротких имен для всех указанных вами областей удаленной работы. Если репозиторий был клонирован, вы должны увидеть по крайней мере источник, то есть имя, которое Git по умолчанию присваивает клонируемому серверу. Параметр -v позволяет увидеть URL-адреса, которые Git хранит для сокращенного имени, используемого при чтении из данного удаленного репозитория и при записи в него.

| $ git remote  origin |
| --- |

Команда **git fetch** связывается с удаленным проектом и извлекает оттуда все пока отсутствующие у вас данные. После этого у вас должны появиться ссылки на все ветки удаленного проекта, которые можно подвергнуть слиянию или просмотреть.

| $ git fetch [имя удаленного репозитория] |
| --- |

Важно понимать, что команда git fetch помещает все данные в ваш локальный репозиторий, — она не выполняет автоматическое слияние с ветками, с которыми вы работаете в данный момент, и вообще никак не затрагивает эти ветки. Слияние вы выполните вручную, как только в этом возникнет необходимость.

Если же у вас есть ветка, настроенная на слежение за какой-то удаленной, команда **git pull** будет автоматически извлекать информацию из удаленной ветки и выполнять слияние с текущей веткой. В некоторых случаях такой порядок вещей оказывается проще и удобнее.

Чтобы поделиться результатами своего труда, их нужно отправить в репозиторий. Это делается простой командой **git push [имя удаленного сервера] [ветка]**. Для отправки ветки master на сервер origin (еще раз напоминаем, что в процессе клонирования эти имена присваиваются автоматически) следует написать:

| $ git push origin master |
| --- |

Команда сработает только при условии, что клонирование осуществлялось с сервера, где у вас есть доступ на запись, и за это время никто не отправлял туда свои данные. Если вы выполнили клонирование одновременно с другим пользователем и он уже отправил результаты своей работы на сервер, ваша попытка отправки данных окончится неудачей. Вам сначала нужно скачать все добавленное этим пользователем и встроить это в свои данные, и только после этого появится возможность воспользоваться командой push.

Для получения дополнительной информации о конкретном удаленном репозитории применяется команда **git remote show [имя удаленного сервера]**.

Переименование ссылок осуществляется командой **git remote rename**, меняющей сокращенные имена удаленных репозиториев. К примеру, вот как выглядит присвоение репозиторию pb имени paul:

| $ git remote rename pb paul  $ git remote  origin  paul |
| --- |

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.

Для участия в чужом проекте, где у вас нет права на запись, нужно создать ветку (fork) этого проекта. То есть GitHub генерирует отдельную копию проекта, находящуюся в вашем пространстве имен, и предоставляет вам доступ на запись туда. Такой подход избавляет владельцев проектов от необходимости добавлять пользователей, желающих принять участие в работе, и предоставлять им доступ на запись.

После щелчка на кнопке Create Pull Request владелец проекта, для которого вы создали ответвление, получит уведомление о предлагаемом изменении и ссылку на страницу с полной информацией о нем. Теперь владелец проекта может ознакомиться с предложенным изменением и выполнить его слияние, отвергнуть его или оставить к нему комментарий.

Issue в GitHub — это своего рода баг-трекер для проекта, у которого есть свой раздел в каждом репозитории. Его можно использовать для обсуждений, которые включают в себя множество вещей, таких как сообщение об ошибке в этом репозитории, задавание вопросов. Чтобы создать новую проблему в своем репозитории, перейдите к репозиторию, в котором вы хотите создать проблему, и перейдите на вкладку «Issues» и далее нажать «New issue».

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

Программный продукт (ПП): программа, работающая без авторского

присутствия

Этапы создание программного продукта:

1. Постановка задачи ( функциональные возможности программы)
2. Выбор метода решения( определение исходных и выходных данных, построение математической модели для решения на компьютере)
3. Разработка алгоритма ( полное и точное описание вычислительного процесса на основе математического описания)
4. Написание программы на языке программирования ( кодирование)
5. Ввод программы в компьютер ( текстовый код программы который поступает на вход программы)
6. Трансляция ( получение объектного модуля)
7. Компоновка ( объединение объектных модулей программы, получение исполняемого файла)
8. Выполнение исполняемого файла программы
9. Тестирование ( устранение ошибок в программе
10. Отладка ( устранение, локализация и устранение ошибок)
11. Создание пользовательской документации
12. Эксплуатация
13. Модификация ( внесение изменений в ПО в целях исправления ошибок и повышение производительности)
14. Снятие с эксплуатации

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.

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:

* разработка
* эксплуатация
* сопровождение

Организационные процессы:

* управление
* усовершенствование
* создание инфраструктуры
* обучение

Вспомогательные процессы:

* документирование
* управление конфигурацией
* обеспечение качества
* аттестация
* разрешение проблем

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

Содержание этапов:

* последовательное и однократное выполнение всех этапов проекта в строго фиксированном порядке;
* переход на следующий этап после полного завершения работ на предыдущем этапе.

Область применения:

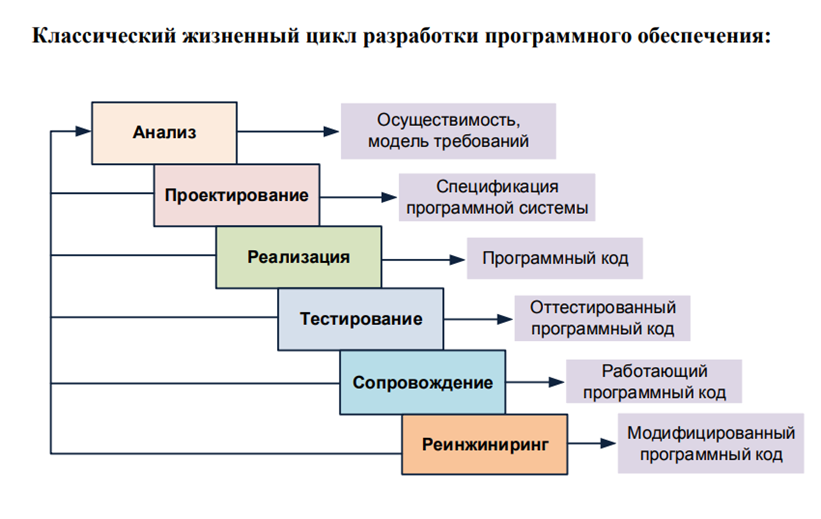
* в критически важных системах реального времени (например, управление авиационным движением или медицинским оборудованием);
* в масштабных проектах, в реализации которых задействовано несколько больших команд разработчиков;
* при разработке новой версии уже существующего продукта или переносе его на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

Достоинства:

* 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.

****

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

Последовательность действий:

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.

Область применения:

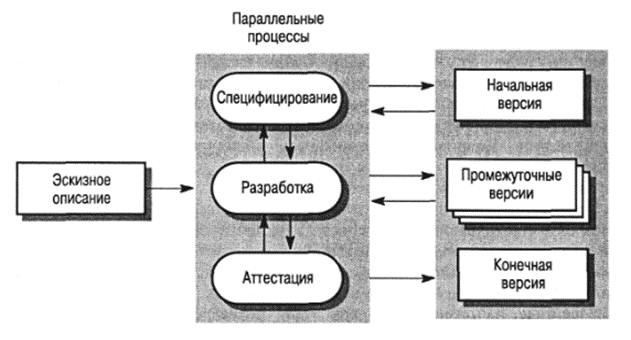
* при разработке проектов, для которых требования слишком сложны, неизвестны заранее, непостоянны или их следует уточнить;
* при разработке больших и долгосрочных проектов;

Достоинства:

* возможность уточнения и внесения новых требований в процессе разработки;
* пригодность для использования промежуточного продукта;
* возможность обеспечения управления рисками;
* реализация преимуществ инкрементной стратегии (выпуск версий, повторное использование результатов, сокращение графика работ);
* обеспечение широкого участия пользователя в проекте, начиная с ранних этапов, что минимизирует возможность разногласий между заказчиками и разработчиками и обеспечивает создание продукта высокого качества;
* уменьшение общих затрат на разработку за счет раннего решения проблем и меньшего объема доработок.

Недостатки:

* сложность определения критериев для продолжения процесса разработки на следующей итерации, неизвестность точного количества необходимых итераций; это затрудняет планирование проекта и может вызвать задержку реализации конечной версии системы или программного средства
* сложность управления проектом; необходимость грамотного управления с целью ограничения количества итераций до разумного;
* необходимость активного участия пользователей в проекте, что не всегда возможно в реальной жизни;
* необходимость мощных инструментальных средств и методов прототипирования;
* возможность отодвигания решения трудных проблем на последующие циклы, что может привести к несоответствию полученных продуктов требованиям заказчиков.

****

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

Содержание этапов создания ПС:

* 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.

Область применения:

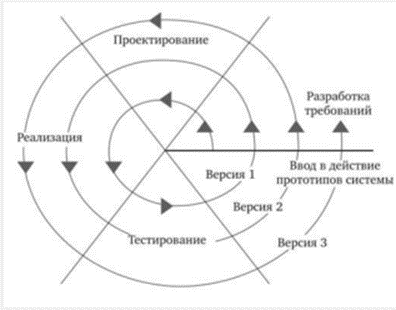
* для решения критически важных бизнес-задач

Достоинства:

* 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.

****

**29. Инкрементальная модель разработки ПО. Развитие инкрементального подхода. XP процессы.**

Инкрементная модель разработки программного обеспечения:

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

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.

Достоинства:

* быстрый выпуск минимального продукта;
* ошибка обходится дешевле;
* постоянное тестирование пользователями.

Недостатки:

* требуется переработка проекта;
* отсутствие фиксированного бюджета и сроков.



Развитием инкрементального подхода явилось создание некоторых новых технологий, пользующихся в настоящее время достаточным успехом:

1.Методология быстрой разработки приложений – Rapid Application Development (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.

Область применения:

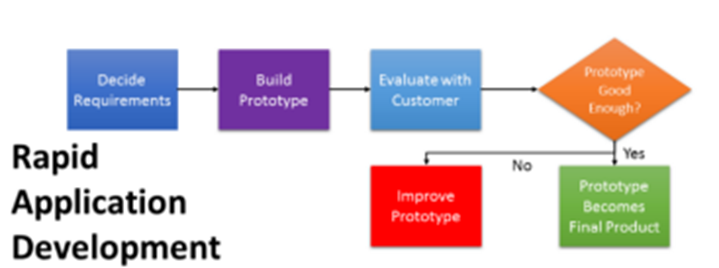
* для проектов, которые легко разделить на независимые или слабосвязанные модули;
* если требования к программному обеспечению быстро меняются;
* в условиях ограниченного бюджета;
* нет ясного представления, как должен выглядеть и работать продукт;
* разработка ведется командой профессионалов;
* если пользователь готов активно участвовать в проекте на протяжении всей работы.

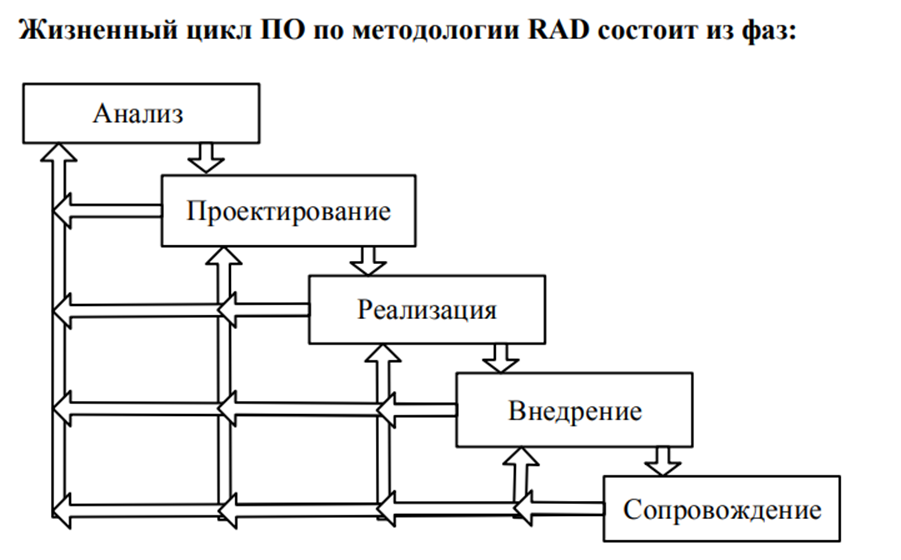
Достоинства:

* 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.





2.Extreme Programming (XP)

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.

Область применения:

* из-за недостатка структуры и документации не подходит для крупных проектов.

Достоинства:

* 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.



**30.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.

Инструментарий технологии программирования – совокупность программ и программных комплексов, обеспечивающих технологию разработки, отладки и внедрения создаваемых программных продуктов.

Управление требованиями - процесс, включающий: идентификацию, выявление, документацию, анализ, отслеживание, приоритизацию требований, достижение соглашений по требованиям и затем управление изменениями и уведомление заинтересованных лиц. Управление требованиями – непрерывный процесс на протяжении всего жизненного цикла продукта.

Техническое задание, как термин в области информационных технологий – это юридически значимый документ, содержащий исчерпывающую информацию, необходимую для постановки задач исполнителям на разработку, внедрение или интеграцию программного продукта, информационной системы либо прочего ИТ сервиса.

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

Методологии быстрой разработки ПО – это один из подходов к разработке ПО, который оптимизирует время и затраты на разработку.

Жизненный цикл ПО по методологии RAD (Rapid Application Development) основан на принципах и инструментах, которые максимально сокращают процесс разработки ПО. 1.Методология быстрой разработки приложений – Rapid Application Development (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.

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.

Достоинства:

* 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.

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

Инкрементная модель разработки программного обеспечения:

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

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.

Достоинства:

* быстрый выпуск минимального продукта;
* ошибка обходится дешевле;
* постоянное тестирование пользователями.

Недостатки:

* требуется переработка проекта;
* отсутствие фиксированного бюджета и сроков.

Итерационная модель разработки ПО представляет собой повторяющиеся циклы разработки, которые повторяются несколько раз до достижения окончательной цели.

Цикл включает в себя 4 основных этапа:

1. Определение и анализ требований
2. Дизайн и проектирование согласно требованиям
3. Разработка и тестирование (кодирование, интеграция и тестирование нового компонента)
4. Фаза ревью: оценка, пересмотр текущих требований

Достоинства:

* быстрый выпуск минимального продукта;
* постоянное тестирование пользователями.

Недостатки:

* требуется переработка проекта;
* отсутствие фиксированного бюджета и сроков.
* отсутствие четкой конечной цели для заказчика

Область применения:

* требования к конечной системе заранее четко не определены
* удобна для работы над большими проектами с инновационными подходами

Отличие итерационной модели от инкрементной модели заключается в том, что итерационная модель предполагает доработку продукта после каждой итерации, что обеспечивает лучшее качество продукта.

**33. 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.**

Методологии гибкой разработки ПО - это различные методологии, такие как Scrum, Kanban, Extreme Programming, которые предполагают более гибкую модель разработки, где изменения допускаются по мере их необходимости

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

Scrum кратко:

1. деление работы на части, которые называются спринтами;
2. спринты планируются исходя из требований для данного момента;
3. относительная оценка времени выполнения работ;
4. ревью каждого спринта;
5. фидбек (обратная связь) по поставляемому продукту;
6. ежедневные собрания.

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)

Плэннинг (определяют какие задачи должны быть выполнены за следующий спринт)

Ретроспектива (проводится в конце спринта)

Недостатки:

1. успех проекта во многом зависит от скрам-мастера и квалификации команды;
2. требует регулярной коммуникации с заказчиком;
3. сложность внедрения в масштабных и сложных проектах.

Scrum of scrums — это методология, предлагающая способ объединения нескольких команд, которые должны работать вместе для поставки сложных решений.

Kanban методология предполагает обсуждение производительности в режиме реального времени и полную прозрачность рабочих процессов. Рабочие задачи визуально представлены на доске Kanban, что позволяет участникам команды видеть состояние каждой задачи в любой момент времени.

Экстремальное программирование — гибкая методология, нацеленная на гибкую реакцию на изменения требований к продукту (один программист занимается написанием кода, а его напарник в это же время непрерывно просматривает только что написанный код).

Традиционный подход к разработке ПО основывается на последовательном, планомерном процессе разработки. Гибкие методологии разработки позволяют скорее отвечать на требования рынка и изменения в условиях, в которых разрабатывается продукт. Однако они также имеют минусы, такие как более высокие расходы и более высокий уровень нестабильности, которые могут привести к просрочке проектов.

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

Модель компетентного разработчика определяет требования к компетенциям разработчика.

PSP помогает разработчикам:

* улучшить оценку 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.
* стандарты кодирования.
* формы.
* оценки. Включают 4 основных элемента:

-размер — оценка размера для части продукта. Например, количество строк кода.

-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.

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

Конструирование программного обеспечения - дисциплина программной инженерии. Это детальное создание работающего программного обеспечения посредством комбинации кодирования, верификации, тестирования и отладки.

В процессе конструирования программного изделия осуществляют:

- функциональную декомпозицию задачи, на основе которой определяется архитектура системы, представляющей задачу;

- внешнее проектирование программного обеспечения, выражающееся в форме его внешнего взаимодействия с пользователем;

- проектирование базы данных, если это необходимо;

- проектирование архитектуры программного обеспечения, т. е. определение множества объектов или модулей, функционально связанных с решаемой задачей, включая сопряжения между ними и требования к ним.

Данная область знаний связана с другими областями. Наиболее сильная связь существует с проектированием и тестированием. Причиной этого является то, что сам по себе процесс конструирования программного обеспечения затрагивает важные аспекты деятельности по проектированию и тестированию. Кроме того, конструирование отталкивается от результатов проектирования, а тестирование (в любой своей форме) предполагает работу с результатами конструирования. Достаточно сложно определить границы между проектированием, конструированием и тестированием, так как все они связаны в единый комплекс процессов жизненного цикла

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

Технологии разработки ПО - это методы и инструменты, которые используются для разработки программного обеспечения. Технологии включают методики, методологии, средства и процедуры разработки ПО.

Управление требованиями - процесс, включающий: идентификацию, выявление, документацию, анализ, отслеживание, приоритизацию требований, достижение соглашений по требованиям и затем управление изменениями и уведомление заинтересованных лиц. Это непрерывный процесс на протяжении всего жизненного цикла продукта.

Требования представляют собой функциональные и нефункциональные требования, которые должны быть выполнены для создания качественного продукта.

1. Бизнес-требования содержат высокоуровневые задачи и цели организации-разработчика или заказчиков системы (зачем?)
2. Требования пользователей описывают цели и задачи, которые пользователям позволит решить система (кто и что?)
3. Системные требования определяют функциональность и характеристики системы, которую должны построить разработчики, для того чтобы пользователи смогли выполнить свои задачи.

Функциональные требования – «Что делает?».

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

1. формулируются заказчиками
2. описывают цели, которые требуется достичь с данной системой
3. какие задачи можно решить с помощью системы
4. определяются функциональность, которую необходимо реализовать

Нефункциональные требования – «Как делает?»

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.

1. требования к надежности
2. требования к совместимости
3. требования к эффективности
4. требования к гибкости

Разработка требований – это первый из основных процессов создания программных систем. Этот процесс состоит из следующих основных этапов:

1. Предварительные исследования
2. Проверка требований

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

Формализация функциональных требований - это процесс преобразования требований проекта в форму, понятную для компьютера.

Диаграммы вариантов использования (Use Case Diagrams) используются для представления функциональных требований. Они показывают взаимодействие пользователя с системой, а также взаимодействие между различными акторами и системой. Они состоят из актёра, который является пользователем системы и использует ее, и сценария, которые описывают действия актера.



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

Тестирование программного обеспечения (Software Testing) – проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом.

1. Верификация (Verification) – это процесс оценки системы или её компонентов с целью определения удовлетворяют ли результаты текущего этапа разработки условиям.
2. Валидация (Validation) – это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе..
3. План Тестирования (Test Plan) – это документ, описывающий весь объем работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.
4. Тест дизайн (Test Design) – это этап процесса тестирования ПО, на котором проектируются и создаются тестовые случаи (тест кейсы), в соответствии с определёнными ранее критериями качества и целями тестирования.
5. Тестовый случай (Test Case) – это артефакт (побочный продукт, созданный в процессе тестирования ПО), описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части.
6. Баг/Дефект Репорт (Bug Report) – это документ, описывающий ситуацию или последовательность действий, приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.
7. Тестовое Покрытие (Test Coverage) – это одна из метрик оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода.
8. Детализация Тест Кейсов (Test Case Specification) – это уровень детализации описания тестовых шагов и требуемого результата, при котором обеспечивается разумное соотношение времени прохождения к тестовому покрытию.
9. Время Прохождения Тест Кейса (Test Case Pass Time) – это время от начала прохождения шагов тест кейса до получения результата теста.

Все виды тестирования программного обеспечения, в зависимости от преследуемых целей, можно условно разделить на следующие группы:

1. Функциональное (тестирование безопасности и взаимодействия).
2. Нефункциональное (тестирование производительности, тестирование установки, тестирование удобства пользования, тестирование на отказ и восстановление, конфигурационное тестирование).
3. Связанное с изменениями (дымовое тестирование, регрессионное тестирование, тестирование сборки (Build Verification Test), санитарное тестирование или проверка согласованности/исправности (Sanity Testing)

Цели тестирования убедиться, что ПО отвечает заявленным требованиям и 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 включают в себя раннее тестирование, объективное тестирование, тестирование на основе требований и автоматизированное тестирование.

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

1. Тестирование по степени подготовленности к тестированию:

* Тестирование по документации – тестирование проводится по заранее подготовленным тестовым случаям.
* Интуитивное тестирование – тестирование проводится без какой-либо подготовки, без цели и плана.
* Исследовательское тестирование – тестирование с целью изучения проекта и документации на него, но без подготовленной заранее тестовой документации

1. Тестирование по степени автоматизации:

* Ручное тестирование – все тестирование проводится тестировщиком вручную без помощи скриптов.
* Полуавтоматизированное тестирование – часть тестирования проводится вручную, а часть автоматизирована.
* Автоматизированное тестирование – тестирование полностью автоматизировано.

1. Тестирование по знанию системы:

* Тестирование черного ящика – тестировщик не имеет доступа к коду.
* Тестирование белого ящика – тестировщик имеет доступа к коду.
* Тестирование серого ящика – тестировщик знает общую структуру приложения.

Автоматизация – ручное тестирование (или иное). Последовательность действий:

1. Тестирование начинается после получения спецификаций на разрабатываемое ПО.
2. Подготовлен первый прототип. Необходимо провести дымовое тестирование, по результатам которого делается вывод о возможности дальнейшего тестирования.
3. В случае если «smoke test failed!!!», переходим к п.2 (приложение отправляется на доработку).
4. В случае если «smoke test passed!!!», переходим к следующему виду тестирования – регрессионное тестирование (Regression testing) и санитарное тестирование (Sanity testing).

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

Разработка программной документации - это процесс создания документов, которые описывают, как работает программное обеспечение.

Назначение документирования программного обеспечения состоит в том, чтобы обеспечить понятное и доступное понимание того, как работает программное обеспечение.

Стандарты документирования формируют требования к документированию программного обеспечения, такие как структура, формат и содержание.

Правила хорошей документации:

1. Документация не должна быть избыточной и объемной. Избыточное количество

текста – раздражает и затрудняет восприятие.

1. Вся схема документирования проекта должна быть взаимоувязанной и

логичной. Если в схеме существует документ, который не связан ссылкой с

каким бы то ни было другим документом, то его можно безболезненно из схемы

исключить.

1. Вся оценка трудозатрат должна производиться только на основании описанных

атомарных задач. Чем мельче оцениваемый элемент – тем точнее будет

агрегированная оценка.

1. Всегда необходимо формировать списки оповещения заинтересованных

участников

**41.Управление командой проекта**

Управление командой проекта - это управление процессом работы команды над проектом. Это включает в себя распределение задач между участниками проекта, постановку целей и планирование сроков, мониторинг выполнения и контроль качества. Также могут потребоваться мероприятия по улучшению командной работы, внесение изменений в процессы и принятие решений.

42. Понятия