***ВОПРОСЫ для подготовки к экзамену по дисциплине «Основы программной инженерии»***

Оглавление

[1. Понятия: инженер, инженерия, программная инженерия. 1](#_Toc124110006)

[2. Основные стандарты программной инженерии. 1](#_Toc124110007)

[3. Образовательный стандарт программной инженерии SWEBOOK. Структура и содержание SWEBOK. 2](#_Toc124110008)

[4. Базовые принципы программного управления и построения компьютеров. 4](#_Toc124110009)

[5. История развития вычислительных машин и информационных технологий. 5](#_Toc124110010)

[6. Технические средства реализации информационных процессов: устройства хранения и обработки информации. 6](#_Toc124110011)

[7. Алгоритмы и их программная реализация. 7](#_Toc124110012)

[8. Понятие программного обеспечения. Классификация программного обеспечения. 8](#_Toc124110013)

[9. Программное обеспечение как продукт. Основные характеристики программных продуктов. 10](#_Toc124110014)

[10. IТ-проекты. Критерии успешности и причины неудач IТ-проектов. 10](#_Toc124110015)

[11. Процесс создания программного обеспечения. Основные стадии типового процесса создания программного обеспечения. 11](#_Toc124110016)

[12. Понятие жизненного цикла. Этапы и процессы жизненного цикла программного продукта. 13](#_Toc124110017)

[13. Модели организации жизненного цикла программного продукта. 14](#_Toc124110018)

[14. Методологии разработки программного продукта. Основные характеристики методологий. 21](#_Toc124110019)

[15. Основные организационные задачи при планировании проекта. Инструменты управления проектами. 27](#_Toc124110020)

[16. Общие сведения о Project Management Body of Knowledge (PMBOK). 32](#_Toc124110021)

[17. Временное планирование проекта. Диаграмма Ганта. 33](#_Toc124110022)

[18. Документация программного проекта. Цели и задачи документирования. 34](#_Toc124110023)

[19. Типы и виды программной документации. Организация документооборота в проекте. 35](#_Toc124110024)

[20. Понятие требований к программному обеспечению. Свойства требований. 37](#_Toc124110025)

[21. Понятие требований к программному обеспечению. Классификация требований. 38](#_Toc124110026)

[22. Процесс сбора требований к программному обеспечению. Разработка требований. Анализ требований. 39](#_Toc124110027)

[23. Функциональная спецификация программного продукта. Типовое техническое задание на разработку программы или программного обеспечения. 40](#_Toc124110028)

[24. Спецификация требований к программному. Единая система программной документации. 41](#_Toc124110029)

[25. Фазы конструирования программных продуктов. Описание этапов конструирования. 42](#_Toc124110030)

[26. Понятие интерфейса программы. Классификация интерфейсов. 45](#_Toc124110031)

[27. Проектирование пользовательского интерфейса. 46](#_Toc124110032)

[28. Инструменты конструирования программных продуктов. 48](#_Toc124110033)

[29. Управление сборками. 49](#_Toc124110034)

[30. Управление версиями. Основные понятия версионного контроля. 50](#_Toc124110035)

[31. Системы контроля версий. Классификация СКВ. 51](#_Toc124110036)

[32. Обзор современных систем контроля версий. 52](#_Toc124110037)

[33. Понятие дефекта. Системы управления дефектами. 54](#_Toc124110038)

## 1. Понятия: инженер, инженерия, программная инженерия.

*Инженер* – дипломированный специалист, имеющий высшее техническое образование. Нетрудно догадаться, что *программный инженер*– инженер в области разработки программного обеспечения.

*Инженерия* - деятельность по созданию машин для предоставления полезных для потребителя услуг и изделий.

**Программная инженерия (Software Engineering)** - это инженерная дисциплина, в которой инженеры применяют теоретические идеи, методы и средства при разработке ПО, создают продукты в соответствии со стандартами, регламентирующими процессы их проектирования и разработки.

Программная инженерия описывает методы управления программным проектом, качеством и рисками. Применение таких методов позволяет достичь высокого качества программных продуктов.

Эта инженерная дисциплина предоставляет всю необходимую информацию и стандарты для выбора наиболее подходящего метода и процессов жизненного цикла ПО для реализации конкретного проекта.

// Программная инженерия - применение систематизированного, дисциплинированного и оцениваемого по количественным параметрам подхода к разработке, функционированию и сопровождению программного обеспечения, то есть применение инженерного подхода к созданию ПО.

## 2. Основные стандарты программной инженерии.

Стандарт (standard) - норма, образец, мерило:

– утверждаемый компетентным органом нормативно-технический документ, устанавливающий комплекс норм и правил по отношению к объекту стандартизации

– типовой образец, эталон, модель, принимаемые за исходные для сопоставления с ними других предметов

■ 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). Разрабатываются с учетом требований мирового опыта и специфики отрасли. Подлежат сертификации

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

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

{РАЗРАБОТЧИКИ СТАНДАРТОВ

■ ISO - The International Standards Organization - международная организация по стандартизации, работающая в сотрудничестве с IEC - The International Electrotechnical Commission - международной электротехнической комиссией

■ IEEE Computer Society - профессиональное объединение специалистов в области программной инженерии

■ ACM - Association for Computing Machinery – Ассоциация по вычислительной технике}

## 3. Образовательный стандарт программной инженерии SWEBOOK. Структура и содержание SWEBOK.

SWEBOK – Software Engineering Body of Knowledge – Свод знаний по программной инженерии – содержит описания состава знаний по разделам (областям знаний) программной инженерии.

■ ACM/IEEE CC 2001 – Computing Curricula 2001 – Академический образовательный стандарт в области компьютерных наук. Выделены 4 основных раздела компьютерных наук:

– Computer science,

– Computer engineering,

– Software engineering,

– Information systems,

по каждому из которых описаны области знаний соответствующего раздела, состав и планы рекомендуемых курсов.

■ Ядру знаний SWEBOK соответствует стандарт ISO/IEC TR 19759:2005

SWEBOK также включает обзор смежных дисциплин связь с которыми представлена как фундаментальная, важная и обоснованная для программной инженерии:

1. Computer engineering – разработка компьютеров.

2. Computer science – информатика.

3. Management – общий менеджмент.

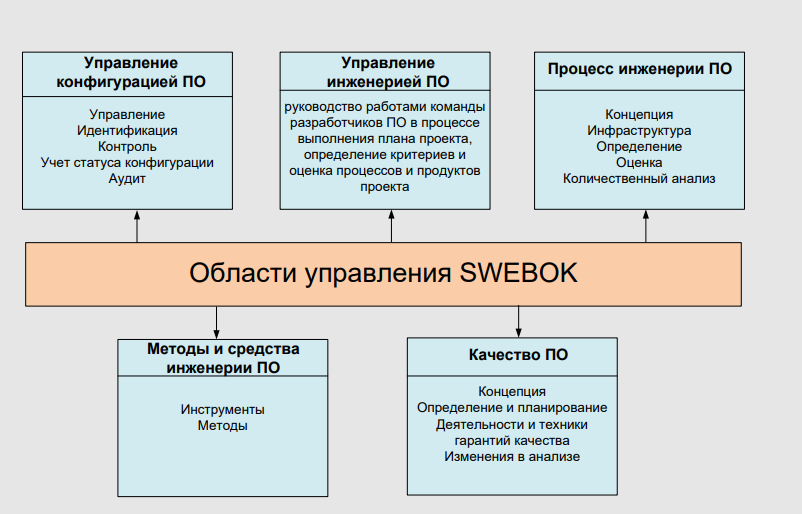
4. Mathematics – математика.

5. Project management – управление проектами.

6. Quality management – управление качеством.

7. Systems engineering – системное проектирование.





## 4. Базовые принципы программного управления и построения компьютеров.

Базовые принципы построения компьютеров:

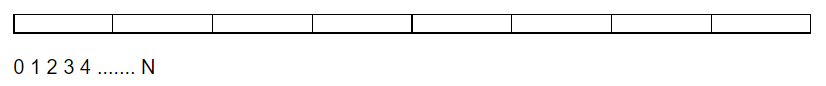
1. Принцип двоичного представления данных.
2. Принцип адресности.
3. Принцип программного управления.
4. Устройства ввода-вывода.

Принцип двоичного представления данных:

Вся информация, циркулирующая в компьютере, представлена двоичным кодом, то есть для ее представления используется двоичная система исчисления (вся информация, поступающая в ЭВМ, кодируется с помощью двоичных сигналов).

Принцип адресности:

Данные, которые необходимо обработать, и программа для их обработки хранятся в оперативной памяти компьютера по точно и однозначно определенным адресам. Оперативная память – это последовательность ячеек памяти. Каждый ячейка имеет свой номер.



Номер ячейки – это ее адрес. Обращение к памяти осуществляется по ее адресам. Адреса ячеек памяти представлены также в двоичном коде. Чем больше ячеек в оперативной памяти, тем больше двоичных разрядов необходимо для записи (представления) адреса.

Минимальная ячейка памяти, к которой можно обратиться – это **байт**или 8 двоичных разрядов.

Принцип программного управления.

Процесс решения задачи в ПК однозначно определяется и автоматически выполняется программой, предварительно занесенной в оперативную память компьютера. Программа является последовательностью команд компьютера. Она определяет очередность выполнения операций над данными.

**Команда** – это управляющее слово или инструкция, определяющее операцию вычислительной машины и данные, которые принимают участие в этой операции. Команды выполняются в процессоре компьютера.

## 5. История развития вычислительных машин и информационных технологий.

Историю развития вычислительной техники условно делят на 5 поколений:

***1-е поколение*** (1945-1954 гг.) — время становления машин с архитектурой, основанной на записывании программы и ее данных в память вычислительной машины. В этот период формируется типовой набор структурных элементов, входящих в состав ЭВМ.

***2-е поколение*** (1955-1964 гг.). Были разработаны команды для вызова подпрограмм. Появились языки высокого уровня — Algol, FORTRAN, COBOL, — создавшие предпосылки для появления переносимого программного обеспечения, не зависящего от типа ЭВМ. С появлением языков высокого уровня возникли компиляторы для них; библиотеки стандартных подпрограмм и другие хорошо знакомые нам сейчас вещи: Важное новшество — это появление процессоров ввода-вывода.

***3-е поколение*** (1965-1970 гг.). Увеличение мощности ЭВМ сделало возможным одновременное выполнение нескольких программ на одной ЭВМ. Для этого нужно было научиться координировать между собой одновременно выполняемые действия, для чего были расширены функции операционной системы.

***4-е поколение*** (1970-1984 гг.). Снижение размеров и стоимости ЭВМ. В начале 70-х годов фирмой Intel был выпущен микропроцессор (МП) i4004. И если до этого в мире вычислительной техники были только три направления (суперЭВМ, большие Э.ВМ (мэйнфреймы) и мини-ЭВМ), то теперь к ним прибавилось еще одно — микропроцессорное.

***5-е поколение*** можно назвать микропроцессорным. На базе разработанных в этот период микропроцессоров в 1984 году появились первые компьютеры.

Этапы развития ИТ:

**Первый этап развития ИТ** — «ручная» информационная техноло­гия (до второй половины XIX в.). Инструментарий: перо, чернильни­ца, бухгалтерская книга. Форма передачи информации — почта.

**Второй этап развития ИТ** — «механическая» информационная технология (с конца XIX в.). Инструментарий: пишущая машинка, те­лефон, фонограф. Передается информация с помощью усовершенст­вованной почтовой связи, идет поиск удобных средств представления и передачи информации.

**Третий этап развития ИТ** начался с конца 40-х гг. XX в. — с соз­дания первых ЭВМ.

**Четвертый этап развития ИТ** — «электронная» информационная технология (с начала 1970-х гг). Ее инструментарием становятся большие ЭВМ и создаваемые на их базе АСУ, оснащенные широким программным обеспечением. Цель — формирование содержательной части информации.

**Пятый этап развития ИТ** — компьютерная («новая») информационная технология (с середины 80-х гг.). Инструментарий — персональный компьютер (ПК) с большим количеством программных продуктов различного назначения. Развивается система поддержки принятия решений, искусственный интеллект реализуется на ПК, используется телекоммуникационная связь. Применяются микропроцессоры. Цель — содержание и доступность для широкого потребителя миниатюризированных технических средств бытового, культурного и прочего назначения.

## 6. Технические средства реализации информационных процессов: устройства хранения и обработки информации.

Основным устройством обработки информации является процессор.

* Процессор является “мозгом“ компьютера. Он состоит из двух основных устройств - устройства управления (УУ) и арифметико-логического устройства (АЛУ).

УУ формирует сигналы управления АЛУ, памятью, внешними устройствами, выдает эти сигналя на шину управления. Кроме этого, оно принимает с шины управления сигналы от внешних устройств и обрабатывает их. УУ следит за ходом выполнения арифметических и логических операций.

АЛУ выполняет арифметические и логические операции над числовой и символьной информацией.

* Внутренние регистры – служат для кратковременного хранения, записи и выдачи информации, непосредственно используемой в вычислениях в ближайшее время работы ПК.
* Регистры – быстродействующие ячейки памяти различной длины (8¸64 разрядов).

## 7. Алгоритмы и их программная реализация.

Алгоритм – последовательность чётко определенных действий, выполнение которых ведёт к решению задачи. Алгоритм, записанный на языке машины, есть программа решения задачи.

Существует несколько способов записи алгоритмов. На практике наиболее распространены следующие формы представления алгоритмов:

1. словесная (запись на естественном языке);
2. псевдокоды (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);
3. графическая (изображения из графических символов – блок-схема);
4. программная (тексты на языках программирования – код программы).

Программная реализация алгоритма – это компьютерная программа, написанная на каком-либо алгоритмическом языке программирования, например: С++, Pascal, Basic и т.д. Программа состоит из команд определенного языка программирования. Отметим, что одна и та же блок-схема может быть реализована на разных языках программирования. Ответ при этом получает ЭВМ, а не человек. Более подробно о составлении программ на языке программирования С++ смотреть Лекцию 3.

Различают три основных вида алгоритмов:

1. линейный алгоритм,
2. разветвляющийся алгоритм,
3. циклический алгоритм.

Линейный алгоритм – это алгоритм, в котором действия выполняются однократно и строго последовательно.

Разветвляющийся алгоритм – это алгоритм, в котором в зависимости от условия выполняется либо одна, либо другая последовательность действий.

Циклический алгоритм – это алгоритм, команды которого повторяются некое количество раз подряд.

## 8. Понятие программного обеспечения. Классификация программного обеспечения.

Программное обеспечение (software) — совокупность компьютерных программ и необходимых для их эксплуатации документов.

В зависимости от назначения, а также области действия, принято выделять три основных класса ПО:

* системное;
* прикладное;
* инструментарий для программирования.

Системное программное обеспечение

Данный вид ПО считается наиболее важным, поскольку именно от него зависит запуск и выполнение любых других процедур. Главным системным ПО в устройстве является его операционная система, определяющая глобальную среду со своими правилами, на основе чего происходит вся последующая работа компьютера.

Операционная система (ОС) – это сложный программный комплекс, реализующий управление аппаратной составляющей компьютера, благодаря которому осуществляется взаимодействие устройства с пользователем и внешним миром, а также выполнение прикладных программ. Кроме того, довольно важной частью ОС является ее файловая система, производящая регулярный контроль над памятью устройства, что проявляется в создании, копировании, перемещении и удалении необходимых файлов с информацией.

Наиболее известными ОС считаются:

* Windows;
* MS-DOS;
* Unix;
* MacOS;
* OS/2.

Прикладное программное обеспечение

Данная разновидность ПО существует для выполнения определенных пользовательских задач, управление которыми происходит из-под установленной ОС. Довольно часто такой класс программ называют приложениями. Обычно с их помощью создаются, редактируются и обрабатываются некие виртуальные объекты и модели, необходимые для решения домашних, офисных, корпоративных либо научных нужд. Современные программы имеют продвинутый графический интерфейс, позволяющий сделать работу пользователя в приложении максимально удобной и эффективной.

Существует множество различных видов прикладного ПО, активно используемого сегодня:

* текстовые редакторы;
* графические пакеты;
* системы баз данных;
* табличные процессоры;
* компьютерные игры;
* веб-браузеры;
* обучающие программы;
* медиаплееры.

Инструментарий для программирования

Последний класс ПО используется для проектирования, написания и тестирования новых программ. Современные наборы инструментов разработки предоставляют создателю приложений огромное количество функциональных возможностей по написанию кода. В большинстве случаев данные  программные решения требуют от человека знаний специального языка, который используется для написания ПО. Однако современные интегрированные среды разработки обладают массой дополнительных настраиваемых параметров, позволяющих в значительной мере облегчить процесс написания программ с помощью автоматического создания отдельных частей кода.

Среди наиболее популярных языков программирования следует выделить следующие:

* C++;
* Java;
* PHP;
* C#;
* JavaScript;
* Python;
* Delphi;
* Pascal.

## 9. Программное обеспечение как продукт. Основные характеристики программных продуктов.

Программный продукт (software product) — совокупность компьютерных программ, процедур и связанных с ними документации и данных, предназначенная для удовлетворения потребностей пользователей, широкого распространения и продажи.

Под программным продуктом (ПП) мы понимаем программное обеспечение (ПО) как результат человеческой деятельности, выставленный на рынке массового покупателя в качестве товара и имеющий ненулевую потребительную стоимость.

Основными характеристиками программных продуктов являются:

* алгоритмическая сложность (логика алгоритмов обработки информации);
* состав и глубина проработки реализованных функций обработки;
* полнота и системность функций обработки;
* объем файлов программ;
* требования к операционной системе и техническим средствам обработки со стороны
* программного средства;
* объем дисковой памяти;
* размер оперативной памяти для запуска программ;
* тип процессора;
* версия операционной системы;
* наличие вычислительной сети и др.

## 10. IТ-проекты. Критерии успешности и причины неудач IТ-проектов.

IT-проект:

* Проект имеет конкретную ограниченную во времени цель
* Выполнение проекта ограничено четкими временными рамками
* Каждый проект по-своему неповторим (уникален)
* На выполнение проекта выделено ограниченное количество ресурсов
* На время выполнения проекта создаются специальные проектные команды

1. Успешные проекты – завершены вовремя в полном объеме и в рамках бюджета;
2. Спорные проекты – завершены не в полном объеме и/или с нарушением сроков, и/или перерасходом бюджета;
3. Проваленные проекты – не доведены до конца (перерасход средств, низкое качество)

Основными причинами неудачи IT-проектов признаются:

1. Нереалистичные временные рамки проекта
2. Недостаток количества исполнителей
3. Размытые границы проекта
4. Недостаток финансирования
5. Нехватка квалифицированных кадров

## 11. Процесс создания программного обеспечения. Основные стадии типового процесса создания программного обеспечения.

Этапы создания ПО:

Анализ требований

Самым первым этапом разработки программного обеспечения по праву называется процедура проведения всестороннего анализа выдвинутых заказчиком требований к создаваемому ПО, чтобы определить ключевые цели и задачи конечного продукта. В рамках этой стадии происходит максимально эффективное взаимодействие нуждающегося в программном решении клиента и сотрудников компании-разработчика, в ходе обсуждения деталей проекта помогающих более четко сформулировать предъявляемые к ПО требования. Результатом проведенного анализа становится формирование основного регламента, на который будет опираться исполнитель в своей работе — технического задания на разработку программного обеспечения. ТЗ должно полностью описывать поставленные перед разработчиком задачи и охарактеризовать конечную цель проекта в понимании заказчика.

Проектирование

Следующий ключевой этап в разработке программного обеспечения — стадия проектирования, то есть моделирования теоретической основы будущего продукта. Самые современные средства программирования позволяют частично объединить этапы проектирования и кодирования, то есть технической реализации проекта, будучи основанными на объектно-ориентированном подходе, но полноценное планирование требует более тщательного и скрупулезного моделирования. Качественный анализ перспектив и возможностей создаваемого продукта станет основой для его полноценного функционирования и выполнения всего комплекса возлагаемых на ПО задач. Одной из составных частей этапа проектирования, к примеру, является выбор инструментальных средств и операционной системы, которых сегодня на рынке присутствует очень большое количество.

Кодирование

Следующим шагом становится непосредственная работа с кодом, опираясь на выбранный в процессе подготовки язык программирования. Описывать особенности и тонкости самого трудоемкого и сложного этапа вряд ли стоит, достаточно указать, что успех реализации любого проекта напрямую зависит от качества предварительного анализа и оценки конкурирующих решений, с которыми создаваемой программе предстоит “бороться” за право называться лучшей в своей нише. Если речь идет о написании кода для выполнения узкоспециализированных задач в рамках конкретного предприятия, то от грамотного подхода к этапу кодирования зависит эффективность работы компании, заказавшей разработку. Кодирование может происходить параллельно со следующим этапом разработки — тестированием программного обеспечения, что помогает вносить изменения непосредственно по ходу написания кода. От слаженности действий программистов, тестировщиков и проектировщиков зависит качество реализации проекта.

Тестирование и отладка

После достижения задуманного программистами в написанном коде следуют не менее важные этапы разработки программного обеспечения, зачастую объединяемые в одну фазу — тестирование продукта и последующая отладка, позволяющая ликвидировать огрехи программирования и добиться конечной цели — полнофункциональной работы разработанной программы. Процесс тестирования позволяет смоделировать ситуации, при которых программный продукт перестает функционировать. Отдел отладки затем локализует и исправляет обнаруженные ошибки кода, “вылизывая” его до практически идеального состояния. Эти два этапа занимают не меньше 30% затрачиваемого на весь проект времени, так как от их качественного исполнения зависит судьба созданного силами программистов программного обеспечения.

Внедрение

Процедура внедрения программного обеспечения в эксплуатацию является завершающей стадией разработки и нередко происходит совместно с отладкой системы. Как правило, ввод в эксплуатацию ПО осуществляется в три этапа:

1. первоначальная загрузка данных;
2. постепенное накопление информации;
3. вывод созданного ПО на проектную мощность.

Ключевой целью поэтапного внедрения разработанной программы становится постепенное выявление не обнаруженных ранее ошибок и недочетов кода.

## 12. Понятие жизненного цикла. Этапы и процессы жизненного цикла программного продукта.

ЖИЗНЕННЫЙ ЦИКЛ ПО – это непрерывный процесс (совокупность процессов), который начинается с момента принятия решения о необходимости его создания и заканчивается в момент его полного изъятия из эксплуатации

В основе практически всех современных промышленных технологий создания ПС лежит международный **стандарт ISO/IEC 12207**

Стадии (этапы) жизненного цикла:

- формирование требований на основе анализа предметной области

- проектирование

- реализация

- внедрение (ввод системы в эксплуатацию)

- эксплуатация (сопровождение проекта)



Для каждой стадии определяют:

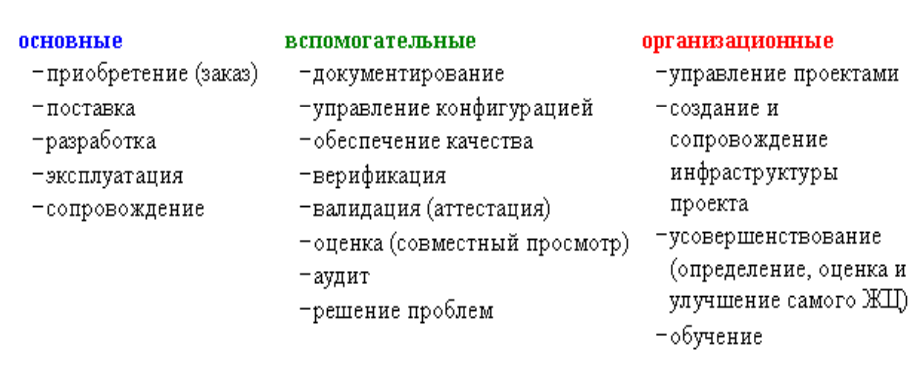
■ состав и последовательность выполняемых работ,

■ получаемые результаты

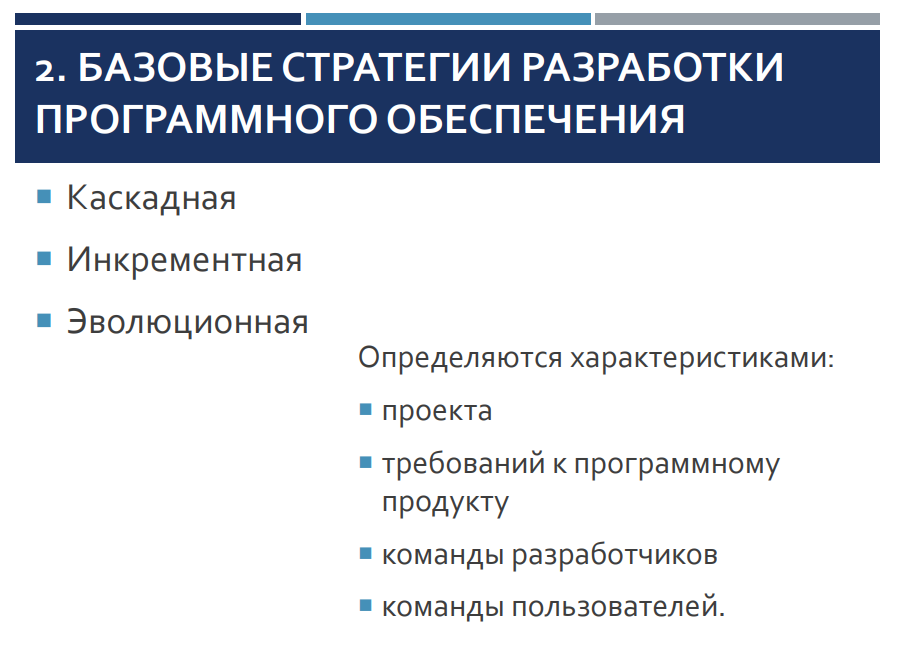
■ методы и средства, необходимые для выполнения работ,

■ роли и ответственность участников и т. Д

Процессы:



## 13. Модели организации жизненного цикла программного продукта.



КАСКАДНАЯ СТРАТЕГИЯ (1970, УИНСТОН РОЙС)

Характеристика:

* последовательное выполнение входящих в ее состав этапов;
* окончание каждого предыдущего этапа до начала последующего;
* отсутствие (или определенным ограничением) возврата к предыдущим этапам;
* наличие результата только в после завершения всех этапов.

ИНКРЕМЕНТНАЯ СТРАТЕГИЯ:

ХАРАКТЕРИСТИКА

* многократное выполнение этапов разработки с запланированным улучшением результата;
* полное определение всех требований к программному средству (системе) в начале процесса разработки;
* полный набор требований реализуется постепенно в соответствии с планом в последовательных циклах разработки.

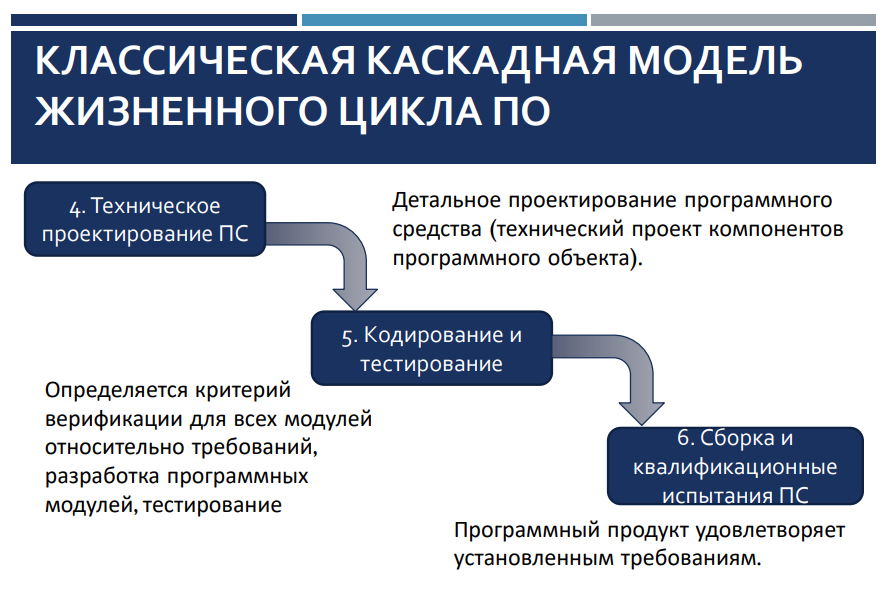
Результат каждого цикла называется инкрементом.

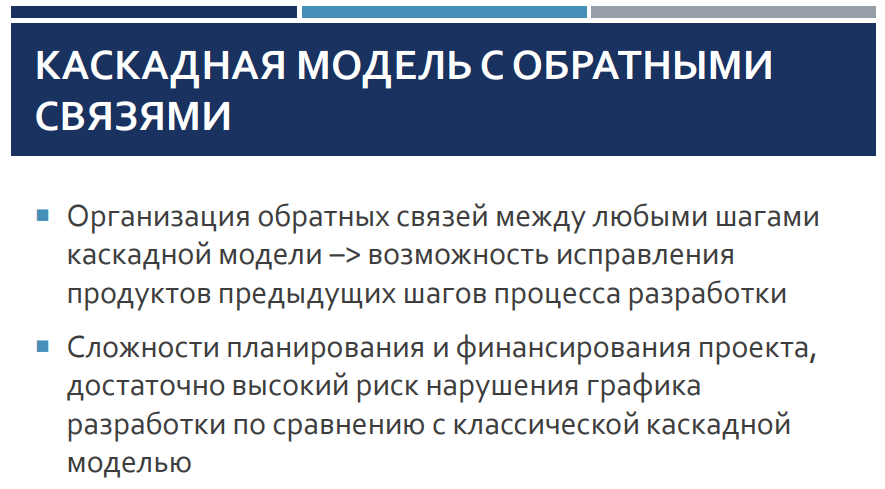
ЭВОЛЮЦИОННАЯ СТРАТЕГИЯ:

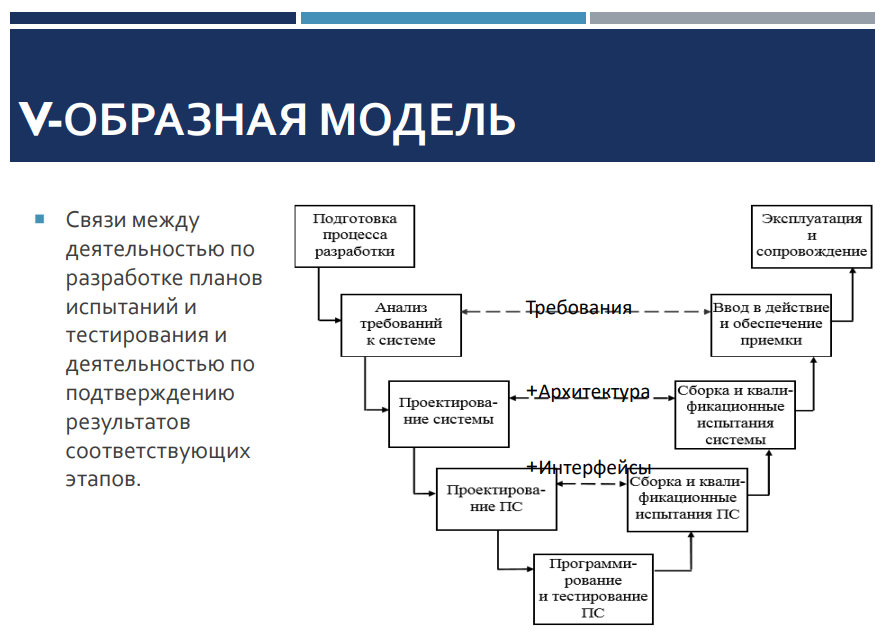
ХАРАКТЕРИСТИКА

* представляет собой многократный проход этапов разработки
* основана на частичном определении требований к разрабатываемому программному средству в начале процесса разработки
* требования постепенно уточняются в последовательных циклах разработки
* результат каждого цикла разработки представляет собой очередную поставляемую версию программного средства.
* Классическая каскадная модель
* Каскадная модель с обратными связями
* V-образная модель
* Спиральная модель

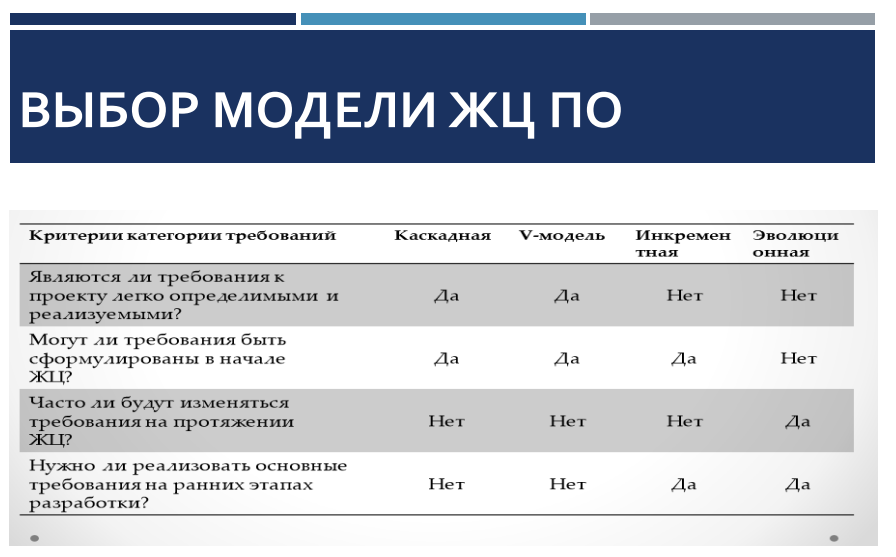
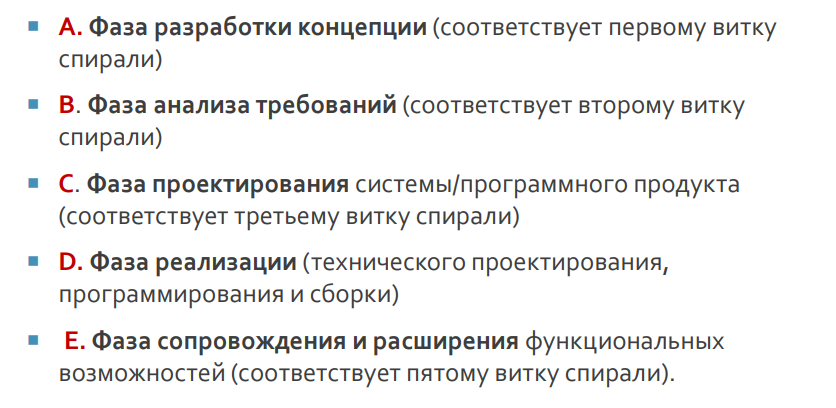


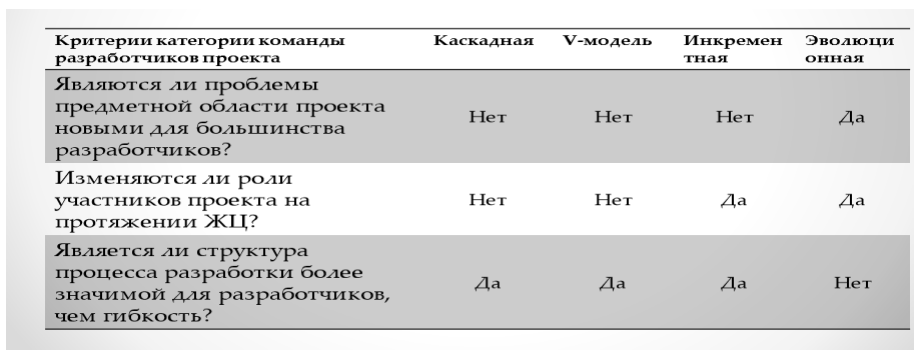






**** 





## 14. Методологии разработки программного продукта. Основные характеристики методологий.

Методология - совокупность методов, правил и принципов управления процессом разработки, которые делают её более эффективной

В зависимости от условий разработки выделяют:

* **Прогностические («тяжеловесные») процессы**:

1. Предполагают планирование всего объема работ, достаточно большой объем документации Основная цель – отделить успешные практики разработки и сопровождения ПО от конкретных людей, умеющих их применять
2. Многочисленные вспомогательные действия имеют целью выполнение успешной разработки с помощью имеющихся работников, не обязательно являющихся суперпрофессионалами

* **Адаптивные или облегченные, гибкие, «живые» (agile) процессы разработки**

1. Не требуют столь жесткой регламентации, допускают возможность частых и существенных изменений требований заказчиков
2. Делают упор на использование хороших разработчиков, а не хорошо отлаженных процессов разработки
3. Избегают фиксации четких схем действий, чтобы обеспечить большую гибкость в каждом конкретном проекте и не требуют создания дополнительных промежуточных документов



*1. RAD (Rapid Application Development)*

Быстрая разработка приложений:

характеристика

• небольшая команда разработчиков (до 10 человек);

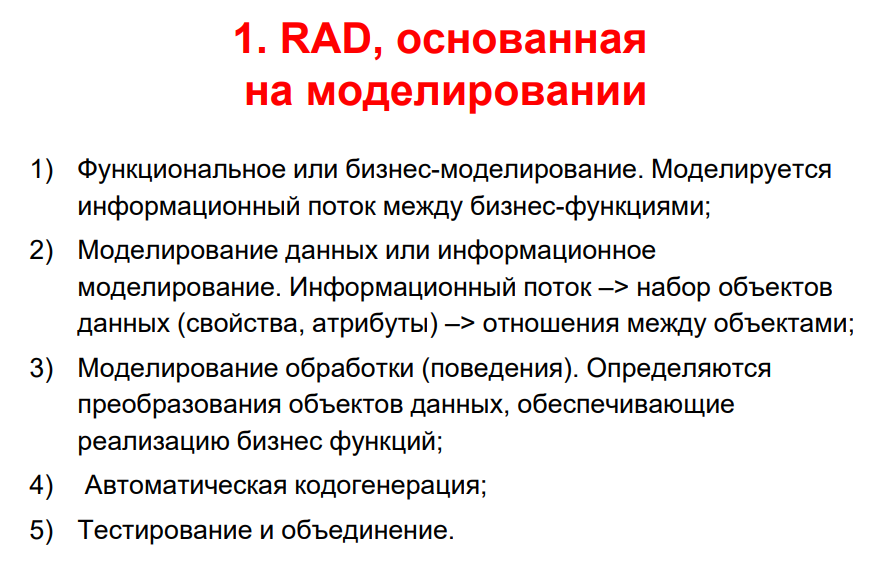
• короткий, но тщательно проработанный производственный график (от 2 до 6 месяцев);

• повторяющийся цикл

• эволюционная или инкрементная стратегии разработки

• мощные инструментальные средства разработки – визуальные среды проектирования и программирования

• прототипирование



*2. Rational Unified Process, RUP (РАЦИОНАЛЬНЫЙ УНИФИЦИРОВАННЫЙ ПРОЦЕСС)*

• Цель RUP: гарантировать высокое качество программного продукта, отвечающего потребностям конечных пользователей, в пределах предсказуемого временного графика и бюджета.

• RUP обеспечивает строгий подход к назначению задач и ответственности в пределах группы разработки

Характеристика

• Заключается в создании и обслуживании моделей

• Сосредотачивает внимание на первоначальной разработке и компоновке устойчивой архитектуры программы

• Поддерживает объектно-ориентированный подход к разработке ПО с перестраиваемой конфигурацией

• Поддерживается инструментальными средствами, которые автоматизируют большинство действий процесса





*3. XP-процесс (ЭКСТРЕМАЛЬНОЕ ПРОГРАММИРОВАНИЕ)*

XP-процесс ориентирован на разработку качественного продукта группами малого и среднего размера в условиях неопределенных или быстро меняющихся требований

• Основная идея XP-процесса – устранить высокую стоимость внесения изменений. Это достигается путем резкого (до двух недель) сокращения длительности отдельных итераций

• Базовыми действиями являются:

* кодирование,
* тестирование,
* выслушивание заказчика,
* проектирование

Принципы и практики ХР

• Короткий цикл обратной связи

– Разработка через тестирование

– Игра в планирование

– Заказчик всегда рядом

– Парное программирование

• Непрерывный, а не пакетный процесс

– Непрерывная интеграция

– Рефакторинг (Design Improvement, Refactor)

– Частые небольшие релизы

• Понимание, разделяемое всеми

– Простота

– Метафора системы

– Коллективное владение кодом или выбранными шаблонами проектирования

– Стандарт кодирования

• Социальная защищенность программиста:

– 40-часовая рабочая неделя

**Scrum**

Ещё одна гибкая методология, которая ориентирована на клиента, ведь он принимает непосредственное участие в процессе работы над проектом. Работа делится на короткие отрезки, которые называют «спринтами». Один такой спринт включает в себя несколько задач. Спринты длятся 1–2 недели и подходят для работы команд численностью до 10 человек.

**Как работать:**определить несколько спринтов, которые помогут достигнуть целей проекта и назначить ответственных за исполнение. Разбить спринты на подзадачи для большей точности контроля за выполнением.

**Плюсы:** задачи не зависят друг от друга, а значит, могут претерпевать изменения по ходу разработки и управления проектом.

**Минусы:** высокие требования к квалификации команды.

Кому подходит: для команд разного размера, в особенности для тех, кто использует методологию Agile.

**5. Kanban**

Методология, которая основана на визуализации задач. Они располагаются на так называемых дорожках канбан-доски, при этом каждая задача представляет собой отдельную карточку. Подход делит все задачи по группам, что значительно упрощает управление проектом.

**Как работать:** создать несколько дорожек для задач. Распределить задачи по дорожкам: необходимо сделать, в работе, на проверку руководителю, выполнено. Назначить ответственного за каждую задачу. Перемещать карточки с задачами по дорожкам по мере выполнения.

**Плюсы:** удобное отслеживание прогресса по задачам, которые могут выполняться параллельно друг другу.

**Минусы:** метод не имеет строгого процесса, поэтому его эффективность зависит от организованности команды.

**Кому подходит:** для работы в офисе и для удалённой работы над проектами, для команд любого размера.

**6. Lean**

Подход, который нацелен на экономию ресурсов и получение большей эффективности из меньшего количества ресурсов. Руководитель команды предотвращает возникновение перегрузки, неравномерности и расточительности при реализации проекта.

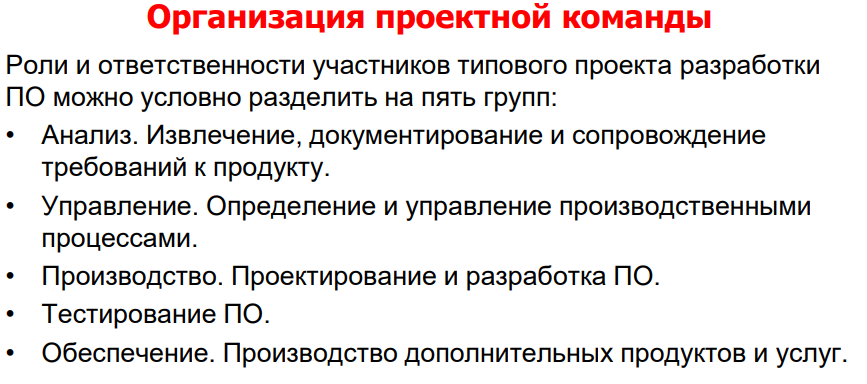
**Как работать:** лучше всего использовать подход в связке с Agile, Scrum или Kanban. Так вы достигнете большей эффективности в работе.

**Плюсы:** повышает эффективность команды, сокращает затраты на проект.

**Минусы:** возможное сопротивление со стороны команды и клиента.

**Кому подходит:** крупным компаниям и командам, у которых наблюдаются проблемы с эффективным использование ресурсов.

## 15. Основные организационные задачи при планировании проекта. Инструменты управления проектами.

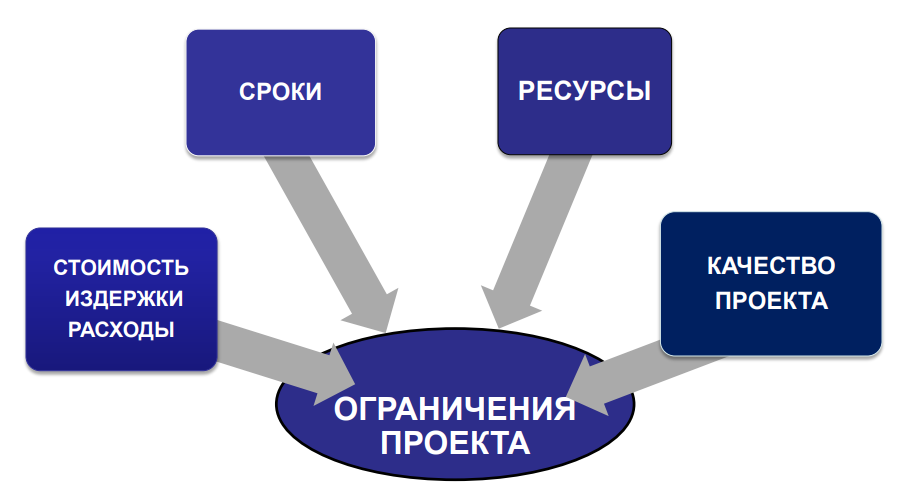


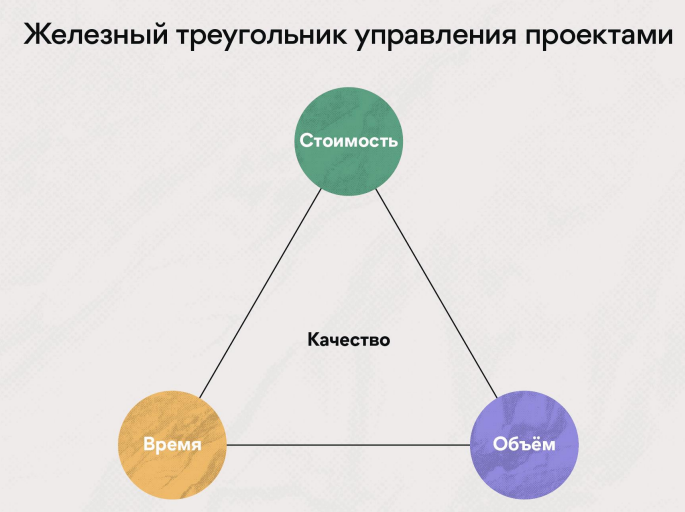
Задачи планирования проекта

На этапе планирования проекта решается ряд задач:

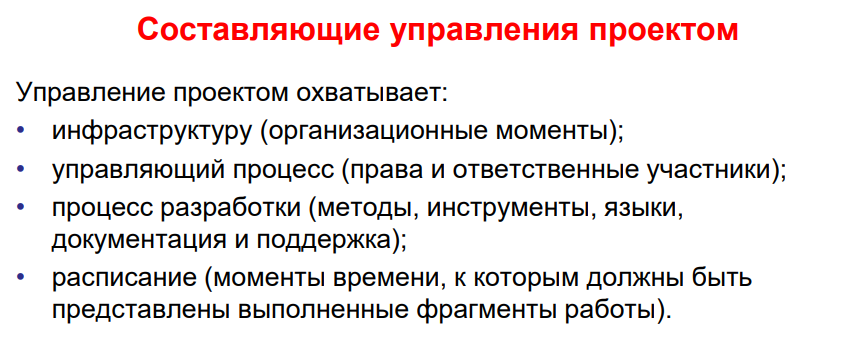
* утверждение плана проекта;
* согласование плана основными участниками проекта;
* создание и уточнение процедур управления проектом;
* уточнение и детализация целей и задач проекта;
* уточнение объема работ и состава проекта;
* уточнение порядка взаимодействия в проектной команде, а также между внешней средой и проектной командой;
* оценка рисков и формирование плана реагирования на них;
* уточнение потребности в ресурсах, план ресурсного обеспечения проекта (или отдельных этапов проекта);
* разработка бюджета проекта и реального расписания (или отдельных его этапов).

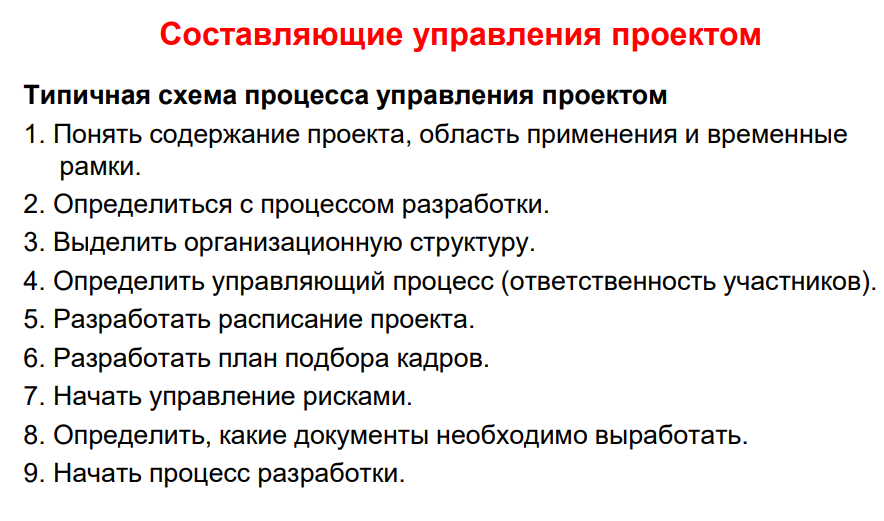
Управление проектом заключается в управлении производством программного продукта в рамках отведенных средств и времени











**Программы для управления проектами:**

**1. GanttPRO**

**Функции и преимущества**

Представитель методологии Waterfall. Позволяет создавать диаграммы Ганта в онлайн-режиме для управления проектами.

* Управление проектами вместе с командой — можно создать рабочее пространство для команды, назначить роли и распределить задачи, а также делиться файлами, обсуждать задачи прямо в инструменте и выставлять прогресс.
* Несколько проектов в наглядной структуре — можно следить за множеством проектов сразу и делегировать управление над ними коллегам.
* Автоматическое планирование — инструмент строит графики в автоматическом режиме, необходимо только задать сроки задач и проектов.
* Управление ресурсами — можно распределять ресурсы между задачами и выставлять их стоимость, из этих данных программа рассчитает стоимость задач и проекта в целом.

**2. Trello**

**Функции и преимущества**

Представитель методологии Kanban. Фактически является канбан-доской, на которой можно удобно расположить необходимые дорожки и карточки с задачами.

* Эффективная командная работа — интуитивно понятный интерфейс и функции, которые позволяют грамотно настроить работу в команде. Можно проводить совещания, ставить цели, устраивать мероприятия.
* Несколько режимов просмотра — доска с колонками и карточками, хронология событий, отслеживание показателей продуктивности, структурирование работы в календарях.
* Удобное управление карточками — можно легко отслеживать задачи и управлять ими, делиться информацией с командой проекта, вести списки задач, видеть даты выполнения, беседы и вложения.
* Автоматизация монотонных процессов — доверьте простые задания роботам и освободите свою команду от рутины.
* Интеграция с другими инструментами для работы в команде.

**3. Jira**

Представитель методологии Agile и Scrum одновременно. Удобное решения для достижения целей при планировании и разработке проектов.

* Настраиваемая Scrum-доска — кастомизируйте так, как удобно вашей команде.
* Отчеты о прогрессе — отслеживайте прогресс команды и редактируйте задачи.
* Интеграция со сторонними сервисами — сделайте работу более эффективной.
* Мобильное приложение — работайте на телефоне или планшете.
* Отслеживание ошибок, настраиваемая панель мониторинга, отслеживание времени.

## 16. Общие сведения о Project Management Body of Knowledge (PMBOK).

PMBoK – это максимально полное изложение знаний по управлению проектами, которое признается всеми специалистами данной области. Это изложение является основной для более узких и практических методологий, но вот сам PMBoK назвать методологией, подходящей для практического применения, нельзя. Основная его ценность заключается в концепции, пригодной для создания других методологий. Является Американским национальным стандартом.

В PMBoK выделяются 44 главных процесса, происходящих при управлении проектами. Эти процессы разделены на пять основных групп:

* Процессы инициации (необходимы для определения и авторизации проекта или его фазы)
* Процессы планирования (необходимы для определения и уточнения целей, планирования действий по достижению этих целей)
* Процессы исполнения (необходимы для объединения человеческих и прочих ресурсов для выполнения плана)
* Процессы мониторинга и управления (необходимы для регулярной оценки прогресса проекта, обнаружения отклонений и корректировки действий)
* Процессы завершения (необходимы для формализации приемки результата проекта, подведения проекта или его фазы к завершению)

В общем случае для документирования проекта требуются следующие шаблоны, описанные ниже.

1. Для инициации проекта:

* ***Экономическое обоснование***
* ***Устав проекта***

1. Для планирования проекта:

* ***Расписание проекта.*** Результатом согласования и утверждения сроков проекта является ***Базовое расписание***.
* ***Требования к ресурсам операций***
* ***Базовый план выполнения стоимости***
* ***Базовый план по содержанию***
* ***План управления коммуникациями***
* ***План управления качеством***
* ***Реестр рисков***.

1. Для исполнения проекта:

* ***Акты подтверждения/принятия результатов***
* ***Информация об исполнении работ***.
* ***Журнал регистрации проблем***

1. Для мониторинга и контроля проекта:

* ***Запросы на изменения***
* ***Бюджетные прогнозы***
* ***Отчеты об исполнении***.

1. Для завершения проекта:

* ***Акты выполненных работ*** формально завершают проект, фазу проекта или контракт.
* **Документация по накопленным знаниям** содержит причины проблем и отклонений, обоснования в пользу выбора того или иного корректирующего воздействия и другие знания, накопленные в процессе реализации проекта.
* **Историческая информация по завершенному проекту** оформляется во время анализа успешности проекта и документирования результатов для Спонсора.

## 17. Временное планирование проекта. Диаграмма Ганта.

***Управление проектом по временным параметрам - это функция, обеспечивающая своевременную реализацию отдельных этапов и всего проекта в целом.*** Под временными параметрами обычно понимают периоды времени, в течение которых будут выполнены работы или группы работ, а также моменты контроля за ходом реализации проекта. Время является одним из ключевых факторов, влияющих на эффективность реализации проекта. Сроки реализации проекта в целом, его составных частей, а также отдельных процессов должны планироваться заранее. У минимизации сроков реализации проекта есть три естественных ограничения – технические возможности, технологические требования и качество работ. Эти и некоторые другие параметры необходимо учитывать при выполнении данной функции.

Диаграмма Ганта — это визуальное представление графика работ, построенное согласно плану проекта. На ней отражены задачи и последовательность их выполнения.

График работ состоит из ряда отрезков, размещённых вдоль временной оси. Каждый из них соответствует отдельной задаче или подзадаче. Начало и конец отрезка соответствуют моменту начала и завершения работы по задаче. Длина отрезка — продолжительность работ.

Диаграмма Ганта нужна, чтобы наглядно представить все этапы работы. Она показывает:

* задачи, включённые в проект;
* их продолжительность;
* даты начала и окончания проекта;
* время, которое занимает каждая задача;
* исполнителей, работающих над задачами;
* способы объединения задач.

Всё это позволяет оценить все ресурсы и взаимосвязи задач. А значит, запланировать работу так, чтобы не пришлось глобально пересматривать подход, менять команду или инструменты.

## 18. Документация программного проекта. Цели и задачи документирования.

**Документирование ПО** – процесс подготовки документации. Целью документирования является обеспечение заинтересованных читателей о функциях и структуре системы на том уровне, который обеспечивает их понимание на том уровне, который необходим применительно к решаемым им задачам.

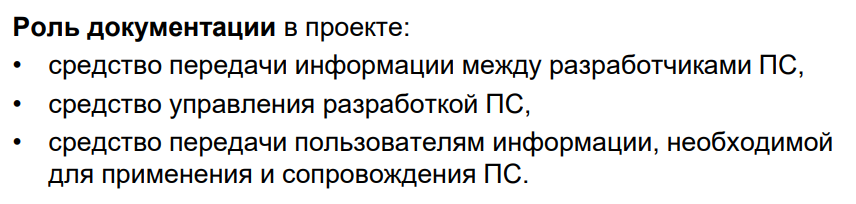
**Требования к документации**

1. Актуальность.

2. Полнота.

3. Универсальность. Возможность использования на различных этапах разработки и эксплуатации ПО.

4. Стандартизованность. Использование во всей документации на ПО общих стандартов оформления и терминологии.



Цели документирования:

▶ посредничество между разработчиками ПО;

▶ упрощение сопровождения и эволюции;

▶ информация для планирования и оценки затрат в процессе разработки;

▶ инструкции по использованию и управлению программной системой;

▶ основание для сертификации системы.

## 19. Типы и виды программной документации. Организация документооборота в проекте.

Типы документации

Документация на процесс разработки (англ. process documentation):

▶ планы разработки;

▶ расписания;

▶ документы оценки качества процессов разработки;

▶ организационные и проектные стандарты.

Документация на продукты разработки (англ. product documentation):

▶ системная (техническая) документация описание программной системы с точки зрения разработчика;

▶ пользовательская документация описание ПО с точки зрения конечного пользователя.

Виды документации:

* Документирование процессов разработки

Виды документации:

▶ планы, оценки затрат и расписания: составляются менеджерами для управления процессом разработки;

▶ отчеты: использование ресурсов на различных этапах;

▶ стандарты: ограничения на процесс разработки (специфичные для организации или национальные / международные);

▶ рабочие документы (working paper): особенности архитектуры системы, стратегии имплементации;

▶ общение между разработчиками и менеджерами

* Пользовательская документация



* Системная документация

Виды системной документации:

▶ документ спецификации требований;

▶ описание общей архитектуры системы;

▶ описание отельных компонентов (архитектура, предоставляемая функциональность и интерфейсы);

▶ исходный код и комментарии в нем;

▶ документы, касающиеся валидации системы;

▶ руководство по сопровождению системы (известные проблемы, направления эволюции, внешние зависимости, …).

* Генерация документации

Примеры генераторов документации:

▶ Javadoc (основной для Java);

▶ Sphinx (основной для Python);

▶ Doxygen (основной для С / С++).

Документооборот — это система создания, обработки, приема, передачи, хранения и архивирования документов на предприятии.

Основные требования к организации документооборота должны соответствовать задачам, которые решает система. Это бесперебойное движение документов по кратчайшим маршрутам с минимальными затратами труда и времени:

1. Четкая маршрутизация. Установленные для разных типов документов схемы движения, обеспечивающие оперативное и беспрепятственное прохождение документов в нужном направлении.
2. Непрерывность. Ее достигают за счет контроля времени, установленного для обработки документа исполнителями и равномерного распределения нагрузки между ними.
3. Регламентация всех процедур исполнения документа. Все процедуры от создания или приема, регистрации и последующей обработки документов регламентируют. Это позволит обеспечить необходимую ритмичность и непрерывность.
4. Исключение дублирования. Это разработка оптимальных маршрутов, исключающая выполнение дублирующих операций при обработке и исполнении документов.

## 20. Понятие требований к программному обеспечению. Свойства требований.

**Требования к программному обеспечению** — совокупность утверждений относительно атрибутов, свойств или качеств программной системы, подлежащей реализации. Создаются в процессе разработки требований к программному обеспечению, в результате анализа требований.

Требования могут выражаться в виде текстовых утверждений и графических моделей.

В классическом техническом подходе совокупность требований используется на стадии проектирования ПО. Требования также используются в процессе проверки ПО, так как тесты основываются на определённых требованиях.

Этапу разработки требований, возможно, предшествовало технико-экономическое обоснование, или концептуальная фаза анализа проекта. Фаза разработки требований может быть разбита на выявление требований (сбор, понимание, рассмотрение и выяснение потребностей заинтересованных лиц), анализ (проверка целостности и законченности), спецификация (документирование требований) и проверка правильности.

Свойства:

1. Полнота.
2. Ясность.
3. Корректность и согласованность (непротиворечивость)
4. Верифицируемость. 3-4
5. Необходимость и полезность при эксплуатации.
6. Осуществимость. 3-6
7. Упорядоченность по важности и стабильности. 3-9

## 21. Понятие требований к программному обеспечению. Классификация требований.

Требования к программному обеспечению — совокупность запросов/утверждений относительно атрибутов, свойств или качеств программной системы, подлежащей реализации. Создаются в процессе проработки (анализа и синтеза) задания на разработку/модернизацию программного обеспечения (ПО).

* Бизнес -требования (Business Requirements) определяют высокоуровневые цели организации или клиента (потребителя) – заказчика разрабатываемого ПО. ( ПОЧЕМУ ? )
* Пользовательские требования (User Requirements) описывают цели /задачи пользователей системы, которые должны выполняться пользователями при помощи создаваемого ПО. ( КТО и ЧТО ? )
* Функциональные требования (Functional requirements) определяют функциональность ПО, которая должна быть создана разработчиками для предоставления возможности выполнения пользователями своих обязанностей в рамках бизнес -требований. ( ЧТО ? )
* Нефункциональные требования включают эксплуатационные характеристики и описание атрибутов качества.

– Атрибуты качества (Quality Attributes) – описывают дополнительные характеристики продукта в различных измерениях, важных для пользователя или разработчиков. Эти характеристики включают практичность, портабельность, целостность, эффективность, устойчивость.

– Ограничение – это формулировка условия, которое модифицирует требование или набор требований сужая выбор возможных решений.

– Бизнес-правила (business rules) включают корпоративную политику, законодательные акты, индустриальные стандарты, учетную практику и алгоритмы вычислений.

* Системные требования (system requirements) описывают высокоуровневые требования для ПО, которое содержит много подсистем (IEEE Std 1233-1998).

## 22. Процесс сбора требований к программному обеспечению. Разработка требований. Анализ требований.

Сбор требований — это один из самых важных этапов процесса создания любой информационной системы, будь то десктопное, веб или мобильное приложение или же просто доработка уже существующего решения. Прежде, чем начать собирать требования, необходимо выявить всех заинтересованных лиц (стейкхолдеров), которые будут пользоваться системой.

Варианты сбора требований:

1. Анкетирование
2. Интервью
3. Изучение существующей документации
4. Представитель заказчика в компании разработчика
5. Мозговой штурм (совещание)

Разработка требований определяется как процесс определения, документирования и поддержания требований. Дисциплина включает в себя все приемы, методы и процедуры, связанные с определением и управлением потребностями пользователей, относящимися к изучаемой системе.

Анализ требований — часть процесса разработки программного обеспечения, включающая в себя сбор требований к программному обеспечению (ПО), их систематизацию, выявление взаимосвязей, а также документирование.

## 23. Функциональная спецификация программного продукта. Типовое техническое задание на разработку программы или программного обеспечения.

Функциональная спецификация в системной инженерии и разработке программного обеспечения — это документ, описывающий требуемые характеристики системы (функциональность). Документация описывает необходимые для пользователя системы входные и выходные параметры (например, программная система).

Функциональная спецификация состоит из трех частей:

* • описания внешней информационной среды, к которой должны применяться программы разрабатываемой ПС;
* • формулирование функций ПС, определенных на множестве состояний этой информационной среды (такие функции будем называть внешними функциями ПС);
* • описание нежелательных (исключительных) ситуаций, которые могут возникнуть при выполнении программ ПС, и реакций на эти ситуации, которые должны обеспечить соответствующие программы.

В первой части должны быть определены на концептуальном уровне все используемые каналы ввода и вывода и все информационные объекты, к которым будет применяться разрабатываемое ПС, а также существенные связи между этими информационными объектами. Примером описания информационной среды может быть концептуальная схема БД или описание сети датчиков и приборов, которой должна управлять разрабатываемая ПС.

Во второй части вводятся обозначения всех определяемых функций, специфицируются все входные данные и результаты выполнения каждой определяемой функции, включая указание их типов и заданий всех соотношений (или ограничений), которым должны удовлетворять эти данные и результаты. И, наконец, определяется семантика каждой из этих функций, что является наиболее трудной задачей функциональной спецификации ПС. Обычно эта семантика описывается неформально на естественном языке — примерно так, как это делается при описании семантики многих языков программирования. Эта задача может быть в ряде случаев существенно облегчена при достаточно четком описании внешней информационной среды, если внешние функции задают какие-либо манипуляции с ее объектами.

В третьей части должны быть перечислены все существенные случаи, когда ПС не сможет нормально выполнить ту или иную свою функцию (с точки зрения внешнего наблюдателя). Примером такого случая может служить обнаружение ошибки во время взаимодействия с пользователем или попытка применить какую-либо функцию к данным, не удовлетворяющим соотношениям, указанным в ее спецификации, или получение результата, нарушающего заданное ограничение. Для каждого такого случая должна быть определена (описана) реакция ПС.

Техническое задание (ТЗ) — исходный документ, который является основанием для разработки и испытания программы или автоматизированной системы. Техническое задание на программу и программное обеспечение разрабатывается в соответствии с требованиями ГОСТ 19.201-78. Основанием для разработки ТЗ чаще всего является договор.

## 24. Спецификация требований к программному. Единая система программной документации.

Спецификация требований к программному обеспечению (SRS) — это документ, в котором отражены все связанные с программным обеспечением аспекты системы. Сюда входят такие аспекты, как функциональность, производительность, масштабируемость и т. д. По сути, все, что влияет на работу программного обеспечения системы, подпадает под категорию SRS.

Спецификация требований к программному обеспечению (SRS):

1. Определяет функциональные и нефункциональные требования, которые присутствуют в программном обеспечении.
2. Создаётся системным аналитиком, системным архитектором или бизнес-аналитиком.
3. В нем также на высоком уровне описаны как технические, так и функциональные характеристики программного обеспечения.
4. Описывает, как бизнес функционирует при использовании программного обеспечения или приложения.
5. Имеет дело с ресурсами, которые предоставляет компания.

Единая система программной документации - комплекс государственных стандартов, устанавливающих взаимосвязанные правила разработки, оформления и обращения программ и программной документации.

В состав ЕСПД входят:

• основополагающие и организационно-методические стандарты;

• стандарты, определяющие формы и содержание программных документов, применяемых при обработке данных;

• стандарты, обеспечивающие автоматизацию разработки программных документов.

## 25. Фазы конструирования программных продуктов. Описание этапов конструирования.

Конструирование программного обеспечения заключается в создании рабочего программного обеспечения посредством комбинации кодирования, верификации (проверки), модульного тестирования, интеграционного тестирования и отладки.

Анализ требований

Самым первым этапом разработки программного обеспечения по праву называется процедура проведения всестороннего анализа выдвинутых заказчиком требований к создаваемому ПО, чтобы определить ключевые цели и задачи конечного продукта. В рамках этой стадии происходит максимально эффективное взаимодействие нуждающегося в программном решении клиента и сотрудников компании-разработчика, в ходе обсуждения деталей проекта помогающих более четко сформулировать предъявляемые к ПО требования. Результатом проведенного анализа становится формирование основного регламента, на который будет опираться исполнитель в своей работе — технического задания на разработку программного обеспечения. ТЗ должно полностью описывать поставленные перед разработчиком задачи и охарактеризовать конечную цель проекта в понимании заказчика.

Проектирование

Следующий ключевой этап в разработке программного обеспечения — стадия проектирования, то есть моделирования теоретической основы будущего продукта. Самые современные средства программирования позволяют частично объединить этапы проектирования и кодирования, то есть технической реализации проекта, будучи основанными на объектно-ориентированном подходе, но полноценное планирование требует более тщательного и скрупулезного моделирования. Качественный анализ перспектив и возможностей создаваемого продукта станет основой для его полноценного функционирования и выполнения всего комплекса возлагаемых на ПО задач. Одной из составных частей этапа проектирования, к примеру, является выбор инструментальных средств и операционной системы, которых сегодня на рынке присутствует очень большое количество.

В рамках данного этапа стороны должны осуществить:

оценку результатов проведенного первоначально анализа и выявленных ограничений;

поиск критических участков проекта;

формирование окончательной архитектуры создаваемой системы;

анализ необходимости использования программных модулей или готовых решений сторонних разработчиков;

проектирование основных элементов продукта — модели базы данных, процессов и кода;

выбор среды программирование и инструментов разработки, утверждение интерфейса программы, включая элементы графического отображения данных;

определение основных требований к безопасности разрабатываемого ПО.

Кодирование

Следующим шагом становится непосредственная работа с кодом, опираясь на выбранный в процессе подготовки язык программирования. Описывать особенности и тонкости самого трудоемкого и сложного этапа вряд ли стоит, достаточно указать, что успех реализации любого проекта напрямую зависит от качества предварительного анализа и оценки конкурирующих решений, с которыми создаваемой программе предстоит “бороться” за право называться лучшей в своей нише. Если речь идет о написании кода для выполнения узкоспециализированных задач в рамках конкретного предприятия, то от грамотного подхода к этапу кодирования зависит эффективность работы компании, заказавшей разработку. Кодирование может происходить параллельно со следующим этапом разработки — тестированием программного обеспечения, что помогает вносить изменения непосредственно по ходу написания кода. Уровень и эффективность взаимодействия всех элементов, задействованных для выполнения сформулированных задач компанией-разработчиком, на текущем этапе является самым важным — от слаженности действий программистов, тестировщиков и проектировщиков зависит качество реализации проекта.

Тестирование и отладка

После достижения задуманного программистами в написанном коде следуют не менее важные этапы разработки программного обеспечения, зачастую объединяемые в одну фазу — тестирование продукта и последующая отладка, позволяющая ликвидировать огрехи программирования и добиться конечной цели — полнофункциональной работы разработанной программы. Процесс тестирования позволяет смоделировать ситуации, при которых программный продукт перестает функционировать. Отдел отладки затем локализует и исправляет обнаруженные ошибки кода, “вылизывая” его до практически идеального состояния. Эти два этапа занимают не меньше 30% затрачиваемого на весь проект времени, так как от их качественного исполнения зависит судьба созданного силами программистов программного обеспечения. Нередко функции тестировщика и отладчика исполняет один отдел, однако самым оптимальным будет распределить эти обязанности между разными исполнителями, что позволит увеличить эффективность поиска имеющихся в программном коде ошибок.

Внедрение

Процедура внедрения программного обеспечения в эксплуатацию является завершающей стадией разработки и нередко происходит совместно с отладкой системы. Как правило, ввод в эксплуатацию ПО осуществляется в три этапа:

первоначальная загрузка данных;

постепенное накопление информации;

вывод созданного ПО на проектную мощность.

Ключевой целью поэтапного внедрения разработанной программы становится постепенное выявление не обнаруженных ранее ошибок и недочетов кода. В рамках этого этапа разработки программного обеспечения и заказчик, и исполнитель могут столкнуться с рядом достаточно узкого спектра ошибок, связанных с частичной рассогласованностью данных при их загрузке в БД, а также срывов выполнения программных процедур в связи с применением многопользовательского доступа. Именно на этой стадии выкристаллизовывается окончательная картина взаимодействия пользователя с программой, а также определяется степень лояльности последнего к разработанному интерфейсу. Если выход системы на проектную мощность после ряда проведенных доработок и улучшений произошел без особых осложнений, значит предварительная работа над проектом и реализация предыдущих стадий разработки осуществлялась правильно.

Заключение

Создание даже небольшого и технически простого ПО зависит от четкого выполнения каждой фазы, то есть деятельности всех отделов, задействованных в процессе разработки. Четкий план выполнения необходимых мероприятий с указанием конечных целей становится неотъемлемой частью работы разработчиков, планирующих оставаться широко востребованными на рынке труда специалистами. Только правильно составленное техническое задание позволит добиться нужного результата и осуществить разработку по-настоящему качественного и конкурентного ПО для любой платформы — серверной, стационарной или мобильной.

## 26. Понятие интерфейса программы. Классификация интерфейсов.

(Интерфейс- совокупность средств сопряжения и связи, обеспечивающие эффективное взаимодействие системы или их частях).

В общем, интерфейс – это совокупность логических и физических принципов взаимодействия компонентов технических средств вычислительной системы (ВС), т. е. совокупность правил алгоритмов и временных соглашений по обмену данными между компонентами ВС (логический интерфейс), а также совокупность физических, механических и функциональных характеристик средств подключения, реализующих такое взаимодействие (физический интерфейс).

Интерфейс — это «проводник» между человеком и программой, операционной системой, техническим устройством или способ взаимодействия приложений между собой. Человек дает команды с помощью интерфейса, устройство их анализирует и отвечает. Основные задачи, для решения которых он предназначен:

* ввод и отображение информации (звук, изображение);
* управление отдельными приложениями;
* обмен данными с другими устройствами;
* взаимодействие с операционной системой.

Интерфейс подразумевает взаимодействие не только человека и техники, но и компьютер-программа, программа-программа, компьютер-устройство. Например, когда устройства подключают к системному блоку компьютера, как способ взаимодействия используют разъем.

Интерфейсы можно разделить на следующие основные классы:

1) системные интерфейсы ЭВМ (внутренние интерфейсы)

2) интерфейсы периферийных устройств.

3) интерфейсы программно- управляющих модульных систем и приборов.

4) Интерфейсы сетей передачи данных.

Внешние интерфейсы — описание аспектов взаимодействия с другими системами и операционной средой. К ним относятся требования к API продукта или системы, а также требования к API других систем, с которыми осуществляется интеграция. • Примеры внешних интерфейсов: «Обеспечить запись в журнал операционной системы следующих событий: сообщения о запуске и остановке сервиса XX»; «Обеспечить запись в журнал параметров модулей программы: сканера, ядра, антивирусных баз (информация должна заноситься в журнал при запуске программы и при обновлении модулей)».

Интерфейсы ЭВМ и средств промышленной автоматизации по функциональному назначению подразделяют на локальные, мезонинные, системные, интерфейсы периферийных устройств, приборные интерфейсы, интерфейсы локальных вычислительных сетей.

Под приборным интерфейсом понимают совокупность неунифицированных сигналов, которая обеспечивает обмен информацией и управление некоторым конкретным прибором.

## 27. Проектирование пользовательского интерфейса.

Пользовательский интерфейс представляет собой совокупность программных и аппаратных средств, обеспечивающих взаимодействие пользователя и вычислительной системы.

Иными словами, это тот набор кнопок, ссылок, форм, диалоговых окон, иконок, пиктограмм, баннеров, ползунков и лент прокрутки, с помощью которого пользователь управляет продуктом.

Проектирование пользовательского интерфейса – это создание тестовой версии приложения. Это начальный этап разработки пользовательского интерфейса, когда распределяются функции приложения по экранам, определяются макеты экранов, содержимое, элементы управления и их поведение.

Пользовательский интерфейс (ПИ) делится на следующие компоненты:

– интерфейс командной строки – инструкции компьютеру даются путем ввода с клавиатуры текстовых строк (команд). В этом виде интерфейса человек подает «команды» компьютеру, а компьютер их выполняет и выдает результат человеку. Командный интерфейс реализован в виде пакетной технологии и технологии командной строки;

– графический интерфейс пользователя (или WIMPинтерфейс: Window – окно, Image – образ, Menu – меню, Pointer – указатель) – программные функции представляются графическими элементами экрана. Характерной особенностью этого вида интерфейса является то, что диалог с пользователем ведется не с помощью команд, а с помощью графических образов – меню, окон, других элементов. Хотя и в этом интерфейсе подаются команды машине, но это делается «опосредственно», через графические образы. Данный вид интерфейса реализован на двух уровнях технологий: простой графический интерфейс и «чистый» WIMPинтерфейс. Отличительные особенности простого графического интерфейса: выделение областей экрана; переопределение клавиш клавиатуры в зависимости от контекста; использование манипуляторов и серых клавиш клавиатуры для управления курсором. Соб- 7 ственно WIMP-интерфейс характеризуется следующими особенностями: вся работа с программами, файлами и документами происходит в окнах; все объекты представляются в виде значков; все действия с объектами осуществляются с помощью меню; применяется широкое использование манипуляторов для указания на объекты;

– естественно-языковой интерфейс (или SILK-интерфейс: Speech – речь, Image – образ, Language – язык, Knowlege – знание) – пользователь «разговаривает» с программой на родном ему языке. Этот вид интерфейса наиболее приближен к обычной, человеческой форме общения. В рамках этого интерфейса идет обычный «разговор» человека и компьютера. При этом компьютер находит для себя команды, анализируя человеческую речь и находя в ней ключевые фразы. Результат выполнения команд он также преобразует в понятную человеку форму. Этот вид интерфейса наиболее требователен к аппаратным ресурсам компьютера, и поэтому его применение началось для военных целей. Сейчас этот вид интерфейса широко осваивается, особенно для мобильных приложений.

Преимущества хорошего ПИ: 1) повышение конкурентоспособности; 2) снижение стоимости разработки; 3) увеличение аудитории продукта; 4) уменьшение затрат на обучение и поддержку пользователей; 5) уменьшение потерь продуктивности работников при внедрении системы и более быстрое восстановление утраченной продуктивности; 6) доступность функциональности системы для максимального количества пользователей; 7) снижение риска ошибок

## 28. Инструменты конструирования программных продуктов.

К инструментам конструирования относятся инструменты, используемые для производства и трансляции программного представления (например, исходного кода), достаточно детального и явного для машинного выполнения:

– Редакторы, применяемые для создания и модификации исходного кода программ и ассоциированной с ними документации. Это могут быть как редакторы “общего назначения” (что на протяжении многих лет наблюдается в UNIX и unix-подобных средах) или специализированные редакторы с поддержкой специфики целевого языка программирования (что является, в большинстве случаев, прерогативой интегрированных сред разработки – IDE). В то же время, документирование все же является не только и не столько частью редактора, сколько самостоятельной функциональностью, пусть часто и тесно интегрированной с редактором.

– Компиляторы и генераторы кода, традиционно выполнявшие покомандную трансляцию исходного кода. Однако существует тенденция интеграции компиляторов и редакторов в интегрированные среды программирования. К этому классу также относятся препроцессоры, линковщики/загрузчики, а также генераторы кода (за исключением, объектно-ориентированных средств проектирования).

– Интерпретаторы, обеспечивающие исполнение программ посредством эмуляции. Они могут поддерживать действия по конструированию программного обеспечения, предоставляя для исполнения программ окружение, более контролируемое и поддающееся наблюдению, чем это обычно способна сделать та или иная операционная система. Существует явная тенденция к интеграции функций компиляторов и интерпретаторов в единых инструментах. Например, компиляция just-in-time (компиляции «на лету»), когда промежуточный программный код, по мере исполнения или с опережением преобразуется в набор инструкций, исполняемых непосредственно средствами операционной системы, но под контролем среды исполнения. Такого рода подход стал родоначальником ряда современных программных платформ, например, Java и.NET.

– Отладчики, которые поддерживают процесс конструирования программного обеспечения, но, в то же время, функционально отличаются от редакторов и компиляторов.

Кроме того, необходимо выделить «интегрированные средства разработки» (IDE – integrated developers environment), программные библиотеки/библиотеки компонент, а также «программные платформы» (например, Java, J2EE и Microsoft.NET) и «платформы облачных вычислений» (например, Microsoft Azure, Amazon и др.), которые включают наравне с инструментами, как таковыми, и определенные модели конструирования, преобразования и выполнения кода.

## 29. Управление сборками.

Управление сборками – это автоматизированный процесс трансформации исходных текстов ПО в пакет исполняемых модулей, учитывающий многочисленные настройки проекта, настройки компиляции, и интегрируемый с процессом автоматического тестирования.

Процедуру сборки проекта часто автоматизируют, то есть выполняют не из среды разработки, а из специального build-скрипта. Этот скрипт используется тогда, когда разработчику требуется полная сборка всего проекта. А также он используется в процедуре непрерывной интеграции (continues integration) – то есть регулярной сборке всего проекта (как правило – каждую ночь).

Приватные сборки (библиотеки) поставляются с самим приложением, используются только им и храниться в его каталоге.

Локальные сборки видны только самому приложению и никому более, т.е. приложение изолируется от внешнего воздействия, как других программ, так и самой операционной системы.

Концепция приватных сборок сильно упрощает развёртывание (инсталляцию) приложений.

Приватная сборка должна иметь строгое имя: Имя самой сборки; Номер версии (позволяется одновременно использоваться разные версии одной и той же сборки); Открытый ключ (позволяет делать строгое имя уникальным и исключает вероятность подмены сборки); Культурные и региональные настройки

Разделяемые сборки хранятся в глобальном кэше сборок; Сборки, хранящиеся там, используются многими приложениями; Хранилище сборок располагается в каталоге <windir>assembly; В каталоге GAC (Global Assembly Cashe) есть подкаталоги, представляющие каждую сборку, в которых хранятся директории, разбивающие данную сборку по версиям. Таким образом, на компьютере может храниться любое количество версий одной и той же сборки.

## 30. Управление версиями. Основные понятия версионного контроля.

Системы управления версиями — это программное обеспечение, помогающее отслеживать изменения, внесенные в код с течением времени. Когда разработчик редактирует код, система управления версиями создает моментальный снимок файлов. Затем этот моментальный снимок сохраняется, чтобы при необходимости им было можно воспользоваться позже.

Без управления версиями разработчики заманчивы сохранять на своем компьютере несколько копий кода. Это опасно, так как легко изменить или удалить файл в неправильной копии кода, потенциально потеряв работу. Системы управления версиями решают эту проблему, управляя всеми версиями кода, но одновременно предоставляя команде одну версию.

Почему управление версиями имеет значение:

Есть много вещей, которые могут занять время в качестве разработчика. Воспроизведение ошибок, изучение новых инструментов и добавление новых функций или содержимого — лишь несколько примеров. По мере увеличения масштаба требований пользователей управление версиями помогает командам работать вместе и отправлять их вовремя.

За последние несколько десятилетий системы контроля версий (Version Control Systems, VCS) стали гораздо более совершенными, причем некоторым это удалось лучше других. Системы VCS иногда называют инструментами SCM (управления исходным кодом) или RCS (системой управления редакциями). Один из наиболее популярных на сегодняшний день инструментов VCS называется Git. Git относится к категории распределенных систем контроля версий, известных как DVCS (эта тема будет рассмотрена подробнее чуть позже). Git, как и многие другие популярные и доступные на сегодняшний день системы VCS, распространяется бесплатно и имеет открытый исходный код. Независимо от того, какую систему контроля версий вы используете и как она называется, основные ее преимущества заключаются в следующем.

1. Полная история изменений каждого файла за длительный период. Это касается всех изменений, внесенных огромным количеством людей за долгие годы. Изменением считается создание и удаление файлов, а также редактирование их содержимого. Различные инструменты VCS отличаются тем, насколько хорошо они обрабатывают операции переименования и перемещения файлов. В историю также должны входить сведения об авторе, дата и комментарий с описанием цели каждого изменения. Наличие полной истории позволяет возвращаться к предыдущим версиям, чтобы проводить анализ основных причин возникновения ошибок и устранять проблемы в старых версиях программного обеспечения. Если над программным обеспечением ведется активная работа, то «старой версией» можно считать почти весь код этого ПО.
2. Ветвление и слияние. Эта возможность полезна не только при одновременной работе участников команды: отдельные люди также могут извлечь из нее пользу и работать над несколькими независимыми направлениями. Создание «веток» в инструментах VCS позволяет иметь несколько независимых друг от друга направлений разработки, а также выполнять их слияние, чтобы разработчики могли проверить, что изменения, внесенные в каждую из веток, не конфликтуют друг с другом. Многие команды разработчиков программного обеспечения создают отдельные ветки для каждой функциональной возможности, для каждого релиза либо и для того, и для другого. Наличие множества различных рабочих процессов позволяет командам выбирать подходящий для них способ использования ветвления и слияния в VCS.
3. Отслеживаемость. Возможность отслеживать каждое изменение, внесенное в программное обеспечение, и связывать его с ПО для управления проектами и отслеживания ошибок, например Jira, а также оставлять к каждому изменению комментарий с описанием цели и назначения изменения может помочь не только при анализе основных причин возникновения ошибок, но и при проведении другого анализа. История с комментариями во время чтения кода помогает понять, что этот код делает и почему действие реализовано именно таким образом. Благодаря этому разработчики могут вносить корректные и совместимые изменения в соответствии с долгосрочным планом разработки системы. Это особенно важно для эффективной работы с унаследованным кодом, поскольку дает разработчикам возможность точнее оценить объем дальнейшей работы.

## 31. Системы контроля версий. Классификация СКВ.

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

1. Локальные системы контроля версий:

(простая база данных, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль ревизий)

Одной из популярных СКВ была система RCS, которая и сегодня распространяется со многими компьютерами. [RCS](https://www.gnu.org/software/rcs/) хранит на диске наборы патчей (различий между файлами) в специальном формате, применяя которые она может воссоздавать состояние каждого файла в заданный момент времени.

1. Централизованные системы контроля версий:

Такие системы, как CVS, Subversion и Perforce, используют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которые получают файлы из этого централизованного хранилища. Применение ЦСКВ являлось стандартом на протяжении многих лет.

1. Распределённые системы контроля версий:

В РСКВ (таких как Git, Mercurial, Bazaar или Darcs) клиенты не просто скачивают снимок всех файлов (состояние файлов на определённый момент времени) — они полностью копируют репозиторий. В этом случае, если один из серверов, через который разработчики обменивались данными, умрёт, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных.

## 32. Обзор современных систем контроля версий.

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

2. Популярные системы контроля версий:

|  |  |
| --- | --- |
| Название | Год создания |
| 1. GIT | 2005 |
| 2. SVN (Subversion) | 2000 |
| 3. Mercurial | 2005 |
| 4. CVS (Concurrent Versions System) | 1990 |
| 5. Bazaar [2] | 2007 |

Факты о Git:

1. Надежная система сравнения ревизий и проверки корректности данных.

2. Гибкая система ветвления проектов и слияния веток между собой.

3. Наличие локального репозитория, содержащего полную информацию обо всех изменениях, позволяет вести полноценный локальный контроль версий и заливать в главный репозиторий только полностью прошедшие проверку изменения.

4. Высокая производительность и скорость работы.

5. Удобный и интуитивно понятный набор команд.

6. Возможность делать контрольные точки, в которых данные сохраняются полностью. Благодаря этому скорость восстановления данных уменьшается, так как за основу берется ближайшая контрольная точка, следовательно, восстановление идет от нее.

7. Широкая распространенность, легкая доступность и качественная документация.

8. Unix – ориентированность. На данный момент отсутствует официальная полноценная реализация Git, совместимая с другими операционными системами, такими как Windows, Mac OS и тому подобные.

3. Отслеживается только изменение всего проекта целиком, а не отдельных файлов, что может быть неудобно при работе с большими проектами, содержащими множество несвязных между собой файлов.

Факты о SVN:

1. Поддерживается большинство возможностей CVS.

2. Разнообразные графические интерфейсы и удобная работа из консоли.

3. Отслеживается история изменения файлов и каталогов даже после их переименования и перемещения.

4. Высокая эффективность работы, как с текстовыми, так и с бинарными файлами.

5. Наличие удобного механизма создания меток и ветвей проектов.

6. Можно с каждым файлом и директорией связать определенный набор свойств, облегчающий взаимодействие с системой контроля версии.

7. Широкое распространение позволяет быстро решить большинство возникающих проблем, обратившись к данным, накопленным Интернет-сообществом.

8. Полная копия репозитория хранится на локальном компьютере в скрытых файлах, что требует достаточно большого объема памяти.

9. Слабо поддерживаются операции слияния веток проекта.

10. Сложности с полным удалением информации о файлах, попавших в репозиторий, так как в нем всегда остается информация о предыдущих изменениях файла, и не предусмотрено никаких штатных средств для полного удаления данных о файле из репозитория.

2.3 Mercurial

1. Первоначально, она была создана для эффективного управления большими проектами под Linux’ом, а поэтому была ориентирована на быструю и надежную работу с большими репозиториями. На данный момент mercurial адаптирован для работы под Windows, Mac OS X и большинство Unix систем.

2. Большая часть системы контроля версий написана на языке Python, и только отдельные участки программы, требующие наибольшего быстродействия, написаны на языке Си.

3. Так же, как и в Git’е, поддерживается возможность создания веток проекта с последующим их слиянием.

4. Быстрая обработка данных.

5. Простота в обращении.

6. Ориентирован на работу в консоли.

## 33. Понятие дефекта. Системы управления дефектами.

Дефект — изъян в компоненте или системе, который может привести компонент или систему к невозможности выполнить требуемую функцию. Дефект, обнаруженный во время выполнения, может привести к отказам компонента или системы.

1. Функциональные дефекты  — в этом случае приложение функционально работает не так, как задумано.
2. Визуальные дефекты  — в этом случае  приложение выглядит не так, как задумано.
3. Логические дефекты  — в этом случае приложение работает неправильно с точки зрения логики.
4. Дефекты контента
5. Дефекты удобства использования — в этом случае приложение неудобно в использовании.
6. Дефекты безопасности  — в этом случае могут быть затронуты пользовательские данные, есть риск падения системы и т.п.

Управление дефектами — это систематический процесс выявления и устранения ошибок. Цикл управления дефектами содержит следующие этапы: 1) Обнаружение дефекта, 2) Категоризация дефекта 3) Устранение дефекта разработчиками 4) Проверка тестерами, 5) Закрытие дефекта 6) Отчеты о дефектах в конце проекта

Системы управления дефектами:

Также известные как системы отслеживания ошибок — используемые в процессе разработки программного обеспечения инструменты для внесения и последующего контроля ошибок такого ПО.

Основные функции такого рода систем можно разделить на 2 основных пункта:

* документирование найденных багов в работе разрабатываемого продукта, а также обращений от пользователей, предложения для улучшений;
* прослеживание за процессом устранения и выполнения таких ошибок, обращений и улучшений.

В этом разделе мы собираем информацию по различным инструментам, с помощью которых можно создавать баги, управлять их жизненным циклом, назначать на исполнителей и так далее.

Если обобщить то этот инструментарий посвящен различным манипуляциям с информацией по ошибкам продукта.