Оглавление

[1.Цели и задачи технологий разработки ПО. Особенности современных проектов разработки ПО. 2](#_Toc92987224)

[2.Основные определения: программа, программный продукт, программное обеспечение, программная инженерия, жизненный цикл ПО. Стандарты программной инженерии. 4](#_Toc92987225)

[3.Классификация программного обеспечения. Определение и состав системы программирования. 5](#_Toc92987226)

[4.Интегрированная среда разработки Visual Studio. Представление символьной информации в кодировке Windows-1251. 6](#_Toc92987227)

[5.Компоненты классической системы программирования. Трансляторы, ассемблеры, интерпретаторы. Схема работы транслятора. 6](#_Toc92987228)

[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. 7](#_Toc92987229)

[7.Структура системы программирования. Модели памяти (классы памяти): код, статическая память, стек, динамическая память. 8](#_Toc92987230)

[8.Парадигмы программирования. 9](#_Toc92987231)

[9.Язык программирования: определение, назначение, примеры. Исходный код. Объектный код. Объектный модуль. Загрузочный модуль. Оформления кода. 10](#_Toc92987232)

[10.Кодирование информации: определение, назначение, данные, представление данных, кодировки. Примеры. 11](#_Toc92987233)

[11.Спецификация системы программирования. Кодировка ASCII, кодировка Windows-1251, стандарт кодирования символов Unicode. 11](#_Toc92987234)

[12.Кодирование информации: определение, назначение, примеры. Кодировка UNICODE: назначение, структура, UCS, UTF. Прямой (LE) и обратный (BE) порядок байт. BOM: определение, назначение, примеры. 12](#_Toc92987235)

[13.Этапы и цели разработки программы, трудоемкость этапов разработки программ. 13](#_Toc92987236)

[14.Алгоритм программы. Назначение и свойства алгоритмов. Способы описания алгоритмов. 15](#_Toc92987237)

[15.Системы программирования Microsoft, Linux, Unix, IBM. Стандарты языков программирования. Парадигмы (стили) программирования. 16](#_Toc92987238)

[16.Интегрированная среда разработки: определение. Примеры IDE. Назначение, основные возможности. Понятие отладки кода на языке программирования. 16](#_Toc92987239)

[17.Интегрированная среда разработки MS Visual Studio 2019. Назначение, возможности. 17](#_Toc92987240)

[18.Среда разработки: назначение и основные возможности отладчика. Точки остановки. Отображение и модификация локальных данных. Пошаговая отладка. Понятие и назначение дизассемблера. 17](#_Toc92987241)

[19.Методология разработки программного обеспечения. Структурный подход к проектированию ПО. Сущность структурного подхода. Методы структурного программирования 18](#_Toc92987242)

[20. Методология разработки программного обеспечения. Модульное программирование. Сущность структурного подхода. Методы модульного программирования 19](#_Toc92987243)

[21. Системы контроля версий. Классификация. Назначение, разновидности систем контроля версий. Система контроля версий Git: основные возможности. Фиксирование состояния. Ветвления. Слияния веток. Конфликты при слиянии веток. Ветвление проектов. Распределенная разработка. 20](#_Toc92987244)

[22. Понятие веб-сервиса Github. Назначение и основные возможности GitHub. Совместная работа над проектом. 22](#_Toc92987245)

[23. Этапы создания программного продукта. Понятие жизненного цикла разработки программного обеспечения. Назначение модели жизненного цикла ПО. Структура процессов жизненного цикла программного обеспечения. 22](#_Toc92987246)

[24. Каскадная модель жизненного цикла ПС: содержание этапов, область применения, достоинства и недостатки. 24](#_Toc92987247)

[25. Эволюционная модель жизненного цикла ПС: последовательность действий, область применения, достоинства и недостатки. 26](#_Toc92987248)

[26. Спиральная модель разработки ПО: содержание этапов создания ПС, область применения, достоинства и недостатки. 27](#_Toc92987249)

[27. Инкрементальная модель разработки ПО. Развитие 29](#_Toc92987250)

[инкрементального подхода. XP-процессы. 29](#_Toc92987251)

[28.Методологии разработки программного обеспечения. Инструментарий технологии программирования. Управление требованиями. Техническое задание на разработку программного продукта. 30](#_Toc92987252)

[29. Методологии быстрой разработки ПО. Жизненный цикл ПО по методологии RAD. Преимущества, недостатки, область применения. 31](#_Toc92987253)

[30. Инкрементальная модель разработки ПО. Итерационная модель разработки ПО. Отличие итерационной модели от инкрементной модели. 32](#_Toc92987254)

[31. Методологии гибкой разработки ПО: Scrum, Kanban, Extreme Programming. Различия между Agile и традиционным подходом к разработке ПО. Преимущества и недостатки технологий быстрой разработки программного обеспечения. 35](#_Toc92987255)

[32. Модель компетентного разработчика (Personal Software Process). 37](#_Toc92987256)

[33. Этапы конструирования. Подходы к конструированию программных средств. 37](#_Toc92987257)

[34. Технологии разработки ПО. Управление требованиями. Понятие требования к ПО. Виды и уровни требований, классификация требований. Функциональные требования и нефункциональные требования. Разработка требований. 38](#_Toc92987258)

[35. Формализация функциональных требований: диаграммы вариантов использования. Назначение и компоненты диаграмм вариантов использования. Примеры. 40](#_Toc92987259)

[36.Тестирование ПО: основные понятия и определения. Классификация видов тестирования. Цели, задачи и принципы тестирования. 42](#_Toc92987260)

[37. Тестирование ПО: методы тестирования. Ручное тестирование. 44](#_Toc92987261)

[38. Разработка программной документации. Назначение документирования программного обеспечения. Стандарты документирования. 44](#_Toc92987262)

[39.Управление командой проекта. 45](#_Toc92987263)

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

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

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

* сложность описания (достаточно большое количество функций, процессов, элементов данных и сложные взаимосвязи между ними), требующая тщательного моделирования и анализа данных и процессов;
* наличие совокупности тесно взаимодействующих компонентов (подсистем), имеющих свои локальные задачи и цели функционирования (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 неосязаемым объект разработки, доступный пониманию ЭВМ, для которой написан.

Свойства хорошей программы:

* Выполнение функциональных требований
* Соответствие нефункциональным требованиям
* Сопровождаемость (maintainability)
* Надежность (dependability)
* Эффективность (efficiency)
* Удобство использования (usability)
* программный продукт – программа, работающая без авторского присутствия. Программный продукт исполняется, тестируется, конфигурируется без присутствия автора и сопровождается документацией.
* программное обеспечение – совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ
* Программная инженерия (Software Engineering) ориентирована на разработку программного обеспечения прикладных и информационных систем разного назначения.
* Программная инженерия – это применение систематического, дисциплинированного и измеряемого подхода к разработке, эксплуатации и сопровождению 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.

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

**Основные типы стандартов**

**Корпоративные стандарты** разрабатываются крупными фирмами с целью повышения качества своей продукции. Создаются на основе собственного опыта компании, но с учетом требований мировых стандартов

**Отраслевые стандарты** действуют в пределах организаций некоторой отрасли. Разрабатываются с учетом требований мирового опыта и специфики отрасли.

**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.

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

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

**Классификация программного обеспечения**

В зависимости от функций, выполняемых ПО делится на три типа:

1. **Системное ПО** — комплекс программ, обеспечивающих работу компьютера и компьютерных сетей в целом.

2. **Прикладное ПО** — программы, предназначенные для решения определенного круга задач в различных областях человеческой деятельности.

3. **Инструментальное ПО** — средства для разработки и отладки программ. Все системное и прикладное программное обеспечение создается с помощью инструментального ПО.

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

**Состав системы программирования:**

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

и пр.

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

Интегрированная среда разработки Visual Studio — это стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений. Помимо стандартного редактора и отладчика, которые есть в большинстве сред IDE, Visual Studio включает в себя компиляторы, средства автозавершения кода, графические конструкторы и многие другие функции для улучшения процесса разработки.

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

в **Windows** **1251** можно закодировать всего 256 символов. Так как все сводится к двоичной системе исчисления, а байт – это 8 бит (0 и 1), то и максимальное число сочетаний составляет 28 = 256.

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

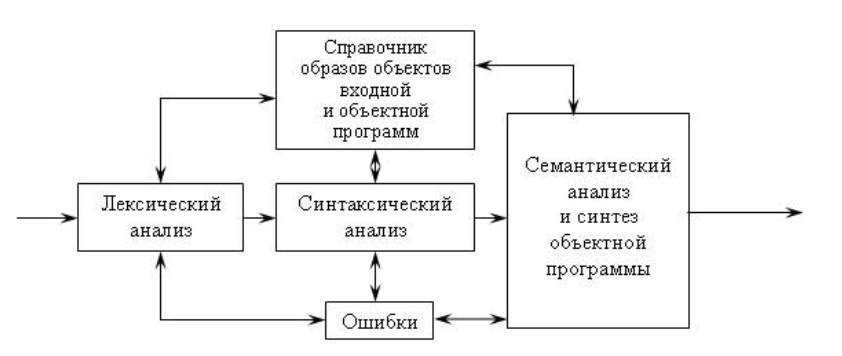


**Трансля́тор** — программа или техническое средство, выполняющее трансляцию программы. Трансля́ция програ́ммы — преобразование программы, представленной на одном из языков программирования, в программу на другом языке. **Транслятор** обычно выполняет также диагностику ошибок, формирует словари идентификаторов, выдаёт для печати текст программы и т. д.

Ассемблер - язык программирования, понятия которого отражают архитектуру электронно-вычислительной машины.

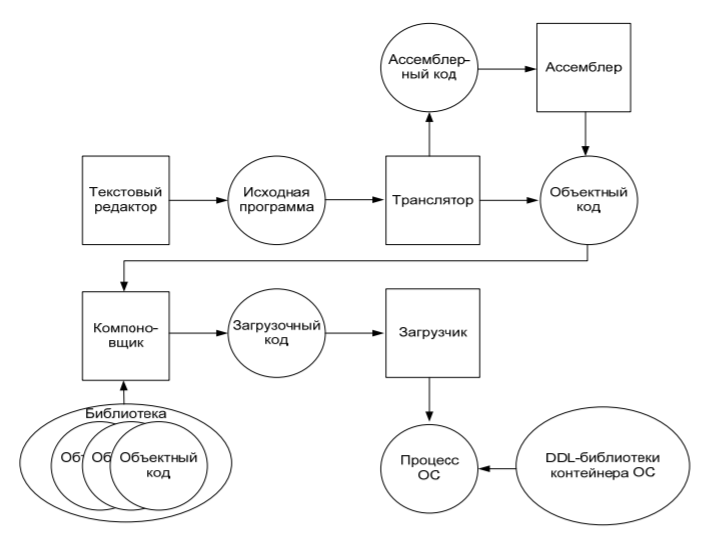
Язык ассемблера — символьная форма записи машинного кода, использование которого упрощает написание машинных программ.

**ИНТЕРПРЕТАТОР** — компьютерная программа, которая выполняет программу, написанную программным языком без ее предварительного перевода 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.

**Структура классической системы программирования**



**От исходного кода к исполняемому модулю, основные этапы преобразования**: Классическая схема создания исполняемого файла выполняется для компилируемых языков: (1) обработка исходного кода препроцессором,

(2) компиляция в объектный код и

(3) компоновка объектных модулей, включая модули из объектных библиотек, в исполняемый файл.

**Язык программирования** – формальная знаковая система, предназначенная для записи компьютерных программ.

Основные элементы: Алфавит, лексемы, синтаксис, стиль  программирования

**Алфавит языка программирования** — это все символы или комбинации символов, которые используются при программировании на этом языке.

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

**Система** **программирования (структура)** обычно включает в себя следующие компоненты:

1. Компилятор или интерпретатор
2. Интегрированная среда разработки
3. Средства создания и редактирования текстов программ
4. Библиотеки стандартных программ и функций
5. Отладочные программы, помогающие находить и устранять ошибки
6. Диалоговая среда
7. Многооконный режим работы
8. Мощные графические библиотеки
9. Утилиты для работы с библиотеками
10. Ассемблер

Все переменные в программе характеризуются не только типом, но и классом памяти. В языке Си существует четыре класса памяти:

* Автоматический (automatic)
* Регистровый (register)
* Статический (static)
* Внешний (external)

**Код** программы – это текст, написанный на каком либо языке **программирования**.

**Статическая** **память** — это область **памяти**, выделяемая при запуске программы до вызова функции main из свободной оперативной **памяти** для размещения глобальных и **статических** объектов, а также объектов, определённых в пространствах имён.

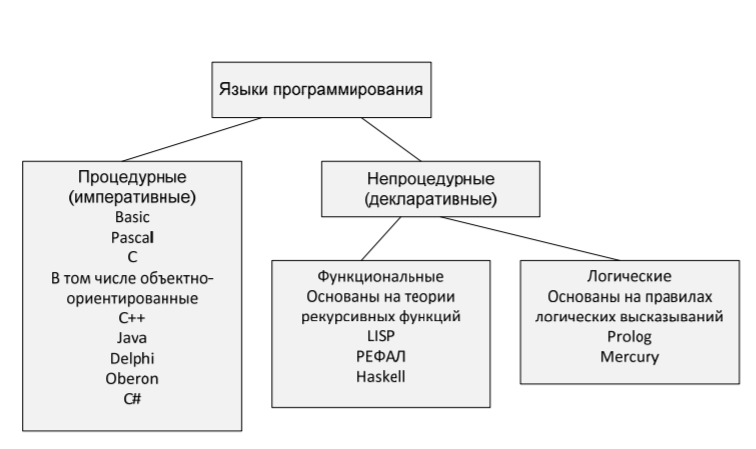
**Стек** (от англ. stack — стопка) — структура данных, представляющая из себя упорядоченный набор элементов, в которой добавление новых элементов и удаление существующих производится с одного конца, называемого вершиной **стека**. Притом первым из **стека** удаляется элемент, который был помещен туда последним, то есть в **стеке** реализуется стратегия «последним вошел — первым вышел»

**Динамическое** выделение **памяти** — это способ запроса **памяти** из операционной системы запущенными программами по мере необходимости. Эта **память** не выделяется из ограниченной **памяти** стека программы, а выделяется из гораздо большего хранилища, управляемого операционной системой — кучи.

# 8.Парадигмы программирования.

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

**Парадигмы (стили) программирования:**



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

**Язык программирования** – формальная знаковая система, предназначенная для записи компьютерных программ.

Назначение – 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.

Примеры компилируемых языков: Pascal, Basic, Visual Basic, Cobol, Swift, Go, Haskell, C#, Kotlin. Интерпретируемые: PHP, JavaScript, Python, Lava, Perl.

Исходный код – Текст компьютерной программы на каком-либо языке программирования или языке разметки, который может быть прочтён человеком.

Объектный код – это последовательность операторов или инструкций на компьютерном языке, обычно это язык машинного кода (т. Е. Двоичный) или промежуточный язык, такой как язык передачи регистров (RTL).

Объектный модуль –  файл с промежуточным представлением отдельного **модуля** программы, полученный в результате обработки исходного кода компилятором.

Загрузочный модуль – это готовая программа в машинных кодах, которая может быть записана во внутреннюю память микроконтроллера.

Оформления кода – набор правил и соглашений, используемых при написании исходного **кода** на некотором языке программирования. Наличие общего стиля программирования облегчает понимание и поддержание исходного **кода**, написанного более чем одним программистом, а также упрощает взаимодействие нескольких человек при разработке программного обеспечения.

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

Кодирование информации – то процесс преобразования данных из исходной формы представления в код.

Назначение:

* компактное хранение, удобство при обработке информации;
* удобство при обмене данными;
* четкое отображение информации;
* шифровка конфиденциальной информации.

Да́нные (калька от лат. data) — это представление фактов и идей в формализованном виде, пригодном для передачи и обработки в некотором информационном процессе.

**Кодировка** **-** **это** **правила** **перевода** **одного** **набора** **символов** **в** **другой.**

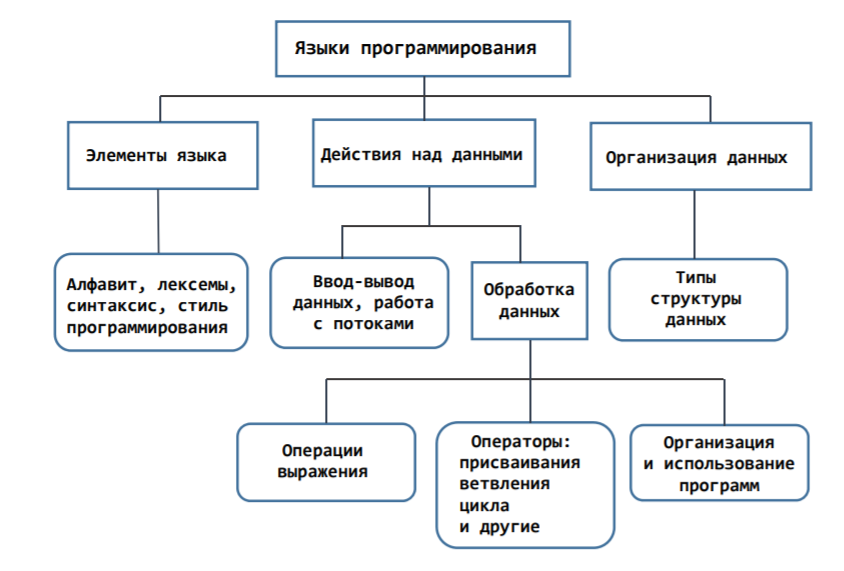
# 11.Спецификация системы программирования. Кодировка ASCII, кодировка Windows-1251, стандарт кодирования символов Unicode.

**ASCII** — **American** **Standard** **Code** for **Information** **Interchange**. **ASCII** была разработана (1963 год) для **кодирования** символов, коды которых помещались в 7 бит (128 символов). Со временем **кодировка** была расширена до 8-ми бит (256 символов), коды первых 128-и символов не изменились.

**Windows**-**1251** (cp1251) — это стандартная 8-битная **кодировка**, разработанная компанией Microsoft. Она содержит практически все символы, которые Вы можете встретить на стандартной русской клавиатуре.

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

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



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

Например: Кодирование текстовой информации, кодирование цвета, графической информации, числовой информации и тд.

**UNICODE: назначение:** применение этого стандарта позволяет закодировать очень большое число символов из разных письменностей: в документах Unicode могут соседствовать китайские иероглифы, математические символы, буквы греческого алфавита, латиницы и кириллицы, при этом становится ненужным переключение кодовых страниц.

UNICODE: UTF-8 — представление Юникода, обеспечивающее наибольшую компактность и обратную совместимость с 7-битной системой [ASCII](https://ru.wikipedia.org/wiki/ASCII); текст, состоящий только из символов с номерами меньше 128, при записи в UTF-8 превращается в обычный текст [ASCII](https://ru.wikipedia.org/wiki/ASCII) и может быть отображён любой программой, работающей с ASCII; и наоборот, текст, закодированный 7-битной ASCII может быть отображён программой, предназначенной для работы с UTF-8.

UTF-16 — кодировка, позволяющая записывать символы Юникода в диапазонах U+0000…U+D7FF и U+E000…U+10FFFF (общим количеством 1 112 064). При этом каждый символ записывается одним или двумя словами (суррогатная пара).

UTF-32 — способ представления Юникода, при котором каждый символ занимает ровно 4 байта. Главное преимущество UTF-32 перед кодировками переменной длины заключается в том, что символы Юникод в ней непосредственно индексируемы, поэтому найти символ по номеру его позиции в файле можно чрезвычайно быстро, и получение любого символа *n*-й позиции при этом является операцией, занимающей всегда одинаковое время.

**UCS** (англ. **Universal** **Coded** Character Set) представляет собой стандартный набор символов, определенный международным стандартом ISO/IEC 10646, который является основой многих символьных кодировок.

**Порядок следования байтов:**

− LE (Little endian order, прямой порядок, от младшего к старшему);

− BE (Big endian order, обратный порядок, от старшего к младшему).

BOM (Byte Order Mark) — маркер последовательности байтов.

BOM назначения: Для определения формата представления Юникода в начало текстового файла записывается сигнатура (обозначение) — символ U+FEFF

Пример BOM



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

Этапы и цели разработки программы:

* 1. Постановка задачи.

• определение функциональных возможностей программы;

• подготовка технического задания

* 1. Выбор метода решения.

• определение исходных и выходных данных, ограничений на них;

• выполнение формализованного описания задачи;

• построение математической модели, для решения на компьютере.

* 1. Разработка алгоритма решения задачи.

• выполняется на основе ее математического описания;

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

* 1. Написание программы на языке программирования (кодирование)

• запись алгоритма на языке программирования.

* 1. Ввод программы в компьютер

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

* 1. Трансляция

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

• получение объектного модуля.

* 1. Компоновка

• объединение одного или нескольких объектных модулей программы и объектных модулей статических библиотек в исполняемую программу;

• связывание вызовов функций и их внутреннего представления (кодов), расположенных в различных модулях;

• получение исполняемого (загрузочного) файла.

* 1. Выполнение

• выполнение исполняемого файла программы на целевой машине.

* 1. Отладка

• обнаружение, локализация и устранение ошибок.

* 1. Тестирование

• подготовка тестовых наборов данных для проверки поведения программы на соответствие предъявляемым к ней требованиям.

* 1. Документирование

• создание пользовательской документации.

* 1. Эксплуатация

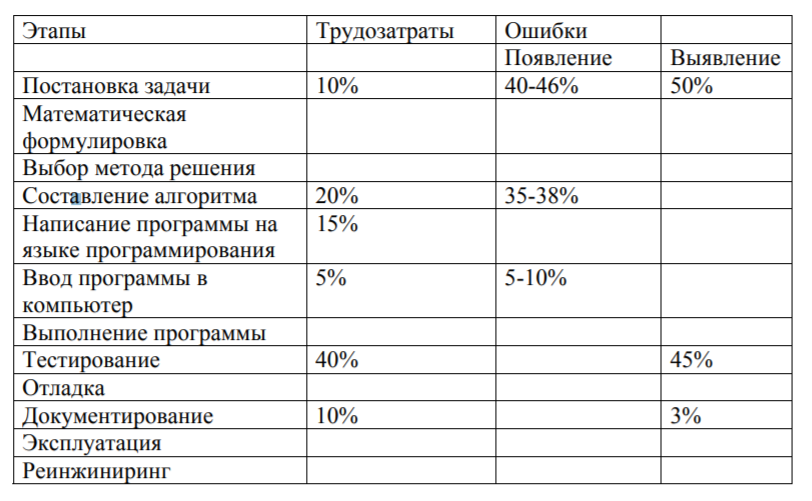
• выполнение в предназначенной для этого среде в соответствии с пользовательской документацией

* 1. Модификация (Реинжиниринг)

• внесение изменений в целях повышения производительности или адаптации к изменившимся условиям работы или требованиям.

* 1. Снятие с эксплуатации

• завершение жизненного цикла ПП и изъятие его из эксплуатации.



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

**Алгоритм** **программы** — **это** точное предписание (совокупность последовательных шагов, схема действий), которое определяет процесс перехода от первичных данных к желаемому результату.

**Свойства:**

* **Дискретность – свойство алгоритма, которое означает, что** **алгоритм состоит из отдельных команд, каждая из которых выполняется за конечное число шагов.**
* **Понятность**
* **Эффективность - процесс должен через конечное число шагов остановиться и выдать искомый результат;**
* **Конечность - свойство алгоритма, которое означает, что для корректного набора данных алгоритм должен завершиться через конечное время с вполне определенным результатом.**
* **Определенность - свойство алгоритма, указывающее на то,** **что каждый шаг алгоритма должен быть строго определен и не допускать различных толкований;**
* **Массовость - Это свойство предполагает, что** **алгоритм должен быть пригоден для решения всех задач данного типа.**

**Способы описания:**

* + словесно-формульный;
  + графический;
  + использование псевдокода;
  + программный.
  + Сети петри

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

Microsoft: Visual C++,  Visual Basic и тд

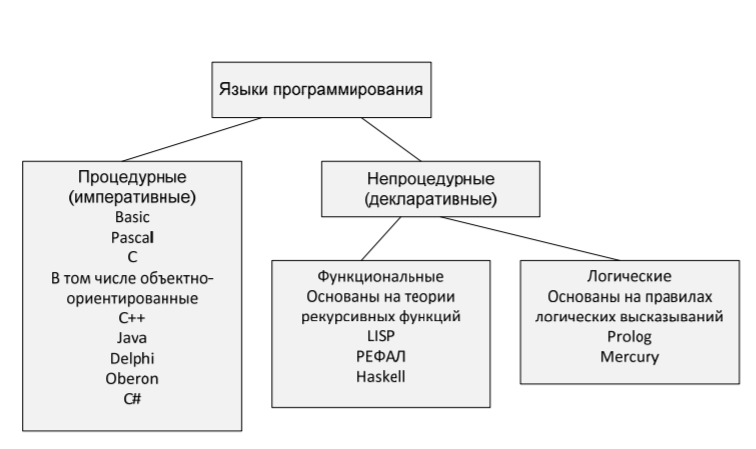
Linux (дистрибутивы): Debian**,** Gentoo**,** Kali Linux

Unix:

**Стандарты языков программирования**

Язык программирования обычно представляется в виде набора спецификаций, определяющих его синтаксис и семантику. Например: Visual C++ 2017 версия 15.3 – это реализация стандарта С++17 или ISO/IEC 14882:2017.

**Парадигмы (стили) программирования**.



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

**Интегри́рованная среда́ разрабо́тки** (*Integrated development environment* *—****IDE)***— комплекс программных средств, используемый [программистами](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%81%D1%82) для разработки [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) (ПО).

Примеры IDE: Microsoft Visual Studio, Eclipse, Xcod, Clion

**Назначение:**  нужен для **разработки** и отладки программ, имеющий общую интерактивную графическую оболочку, поддерживающую выполнение всех основных функций жизненного цикла **разработки** программы - набор и редактирование исходного кода, компиляцию (сборку), исполнение, отладку, профилирование и др.

**Отладка** (**debugging**) — это процесс нахождения ошибок и их устранения

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

Visual Studio — это стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений. Можно использовать для различных аспектов разработки программного обеспечения.

Возможности: разработка, расширение, отладка, тестирование, сотрудничество (система контроля версий).

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

Среда разработки (назначение) – для **разработки** программного обеспечения (ПО)

С помощью **отладчика** можно прерывать или приостанавливать выполнение программы с целью проверки кода, вычислять и редактировать значения переменных программы, отслеживать состояние регистров, просматривать инструкции, созданные из исходного кода, а также просматривать область памяти, используемую приложением.

**то́чка** **остано́ва** (англ. breakpoint) — это преднамеренное прерывание выполнения программы, при котором выполняется вызов отладчика

**Отладка** — это процесс, при котором код пошагово выполняется в некой программе, например, в Visual Studio.

**дизассемблер** - это компьютерная программа, которая переводит машинный язык в язык ассемблера - операция, обратная операции ассемблер.

Дизассемблирование, вывод **дизассемблера**, часто форматируется для удобства чтения человеком, а не для ввода ассемблера, что делает его главным образом инструментом обратного проектирования.

В окнах Видимые и Локальные отображаются значения переменных во время отладки. Окна доступны только во время сеанса отладки. В окне Видимые отображаются переменные, используемые вокруг текущей точки останова. В окне Локальные отображаются переменные, определенные в локальной области, которая обычно является текущей функцией или методом.

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

**Методология** **разработки** ПО – это процесс описания того, как определенный продукт будет разрабатываться, то есть один из способов организации коллективной **разработки**.

Например: каскадные, Инкрементные, итерационные, поэтапные, спиральные; Waterfall, Rational Unified Process (RUP),Agile – общая методология гибкой разработки 4) Spiral 5) Extreme Programming (XP) 6) Structured Analysis and Design Technique (SADT) 7) Microsoft Solutions Framework (MSF) Microsoft Operations Framework (MOF) 8) Rapid Application Development (RAD) 9) Personal Software Process 10) Scrum – концепция работы в условиях сорванных сроков и идеологического кризиса.

Структурный подход заключается в выделении основных функций или действий и базируется на нисходящем проектировании, основанном на структурах и алгоритмах управления. Основное понятие этого подхода - алгоритмы. В объектном или объектно-ориентированном подходе в первую *очередь* выделяется множество основных объектов системы и впоследствии определяется множество операций над объектами. Такой подход базируется на абстрактных типах, и решение задачи выражается в терминах выделенных объектов.

(Сущность)Структурный подход применяется в организациях для обеспечения основных элементов деятельности и взаимосвязей между ними. Он предполагает использование разделения труда, охвата контролем, децентрализацию и департаментализацию. Разделение и специализация труда должны применяться, чтобы предоставить возможность организации произвести большее количество продукции лучшего качества, затратив при этом столько же усилий.

Методология SADT(Structured Analysis and Design Technique):

разработана специально для описания искусственных систем средней сложности. На методологии SADT основана группа методологий IDEF:

− IDEF0 – для создания функциональной модели, отражающей структуру и функции системы, а также потоки информации и материальных объектов, преобразуемые этими функциями;

− IDEF1 – для построения информационной модели, отражающей структуру и содержание информационных потоков, необходимых для поддержки функций системы;

− IDEF2 – позволяет построить динамическую модель изменяющихся во времени функций, информации и ресурсов системы.

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

**Методология** **разработки** ПО – это процесс описания того, как определенный продукт будет разрабатываться, то есть один из способов организации коллективной **разработки**.

**Мо́дульное программи́рование** — это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам.

Структурный подход заключается в выделении основных функций или действий и базируется на нисходящем проектировании, основанном на структурах и алгоритмах управления. Основное понятие этого подхода - алгоритмы. В объектном или объектно-ориентированном подходе в первую *очередь* выделяется множество основных объектов системы и впоследствии определяется множество операций над объектами. Такой подход базируется на нисходящем проектировании, основанном на структурах и алгоритмах управления. Основное понятие этого подхода - алгоритмы. В объектном или объектно-ориентированном подходе в первую *очередь* выделяется множество основных объектов системы и впоследствии определяется множество операций над объектами. Такой подход базируется на абстрактных типах, и решение задачи выражается в терминах выделенных объектов.

(Сущность)Структурный подход применяется в организациях для обеспечения основных элементов деятельности и взаимосвязей между ними. Он предполагает использование разделения труда, охвата контролем, децентрализацию и департаментализацию. Разделение и специализация труда должны применяться, чтобы предоставить возможность организации произвести большее количество продукции лучшего качества, затратив при этом столько же усилий.

**Методы модульного программирования**

Метод восходящей разработки

Сначала строится древовидная модульная структура программы. Затем поочередно проектируются и разрабатываются модули программы, начиная с модулей самого нижнего уровня, затем предыдущего уровня и т. д. То есть модули реализуются в таком порядке, чтобы для каждого программируемого модуля были уже запрограммированы все модули, к которым он может обращаться.

Метод нисходящей разработки

Как и в предыдущем методе, сначала строится модульная структура программы в виде дерева. Затем проектируются и реализуются модули программы, начиная с модуля самого верхнего уровня — головного, далее разрабатываются модули уровнем ниже и т. д. При этом переход к программированию какого-либо модуля осуществляется только в том случае, если уже запрограммирован модуль, который к нему обращается. Затем производится их поочередное тестирование и отладка в таком же нисходящем порядке.

# 21. Системы контроля версий. Классификация. Назначение, разновидности систем контроля версий. Система контроля версий Git: основные возможности. Фиксирование состояния. Ветвления. Слияния веток. Конфликты при слиянии веток. Ветвление проектов. Распределенная разработка.

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

Системы контроля версий классифицируются на:

* Локальные системы контроля версий
* Централизованные системы контроля версий
* Распределенные системы контроля версий

Назначение:

* автоматическое создание архива (бэкап) для синхронизации кодовой базы;
* отслеживание изменений (кто, когда и зачем сделал изменения)
* совместная работа над одним и тем же проектом;
* отслеживание ошибок (Bug трекинговая система).

Разновидности:

* локальные системы контроля версий (Version  
  Control System, VCS, Revision Control System, RCS);
* централизованные системы контроля версий  
  (Centralized Version Control System, CVCS);
* распределенные системы контроля версий  
  (Distributed Version Control System, DVCS).

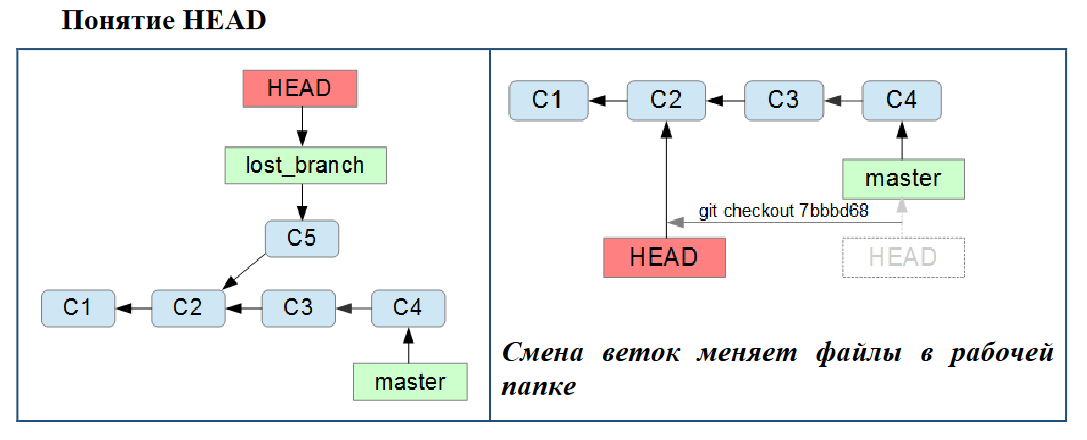
Гит - распределённая система управления версиями. Инструмент, позволяющий реализовать распределённую систему контроля версий.  
Дата запуска: 2005 г. Git хранит информацию в структуре данных,  
называемой репозиторий (repository)

Основные свойства:

* быстродействие и размер;
* безопасность и целостность (хэш SHA);
* достоверность;
* гибкость (нелинейные рабочие процессы – слияние, ветвление);
* производительность (простое ветвление);
* функциональность

В Гит все данные – это набор снимков состояния миниатюрной файловой системы.

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



Процедура объединения веток называется **слияние** (merge). В результате выполнения команды слияния в текущей ветке появится новый коммит. Этот коммит будет иметь два предка – последние коммиты обоих веток, участвующих в слиянии. Содержимое этого коммита будет включать содержимое коммитов обоих веток.

При слиянии может возникнуть ситуация, когда фрагмент в каком-либо файле проекта в различных ветках отредактирован по-разному. Такая ситуация называется **конфликт** (conflict). В случае возникновения конфликтов git заносит в создаваемый при объединении коммит файл, содержащий текст обоих версий. Такой файл получает статус **не объединенный** (unmerged).

При возникновении конфликта пользователь должен в ручном режиме его устранить. Посмотреть список не объединенных файлов можно с помощью команды git status.

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

GitHub — сервис для совместной разработки и хостинга проектов. Основан на системе контроля версий Git.

C **помощью** **GitHub** над кодом проекта может работать неограниченное количество программистов из любых точек мира.

Проект в GitHub хранится в репозитории —место, где хранятся и поддерживаются какие-либо данные.

Коммит – это пакет изменений, хранящий информацию с добавленными, отредактированными или удалёнными файлами кода.

Ветка (branch) — указатель на коммит с определенными изменениями. Можно создать неограниченное количество веток, чтобы вносить новые изменения, не мешая основному проекту. Часто разработчики делают параллельные изменения кода. По окончании работы ветки каждого из них можно объединить в одну. Для этого в Git используют функцию pull request (pr).

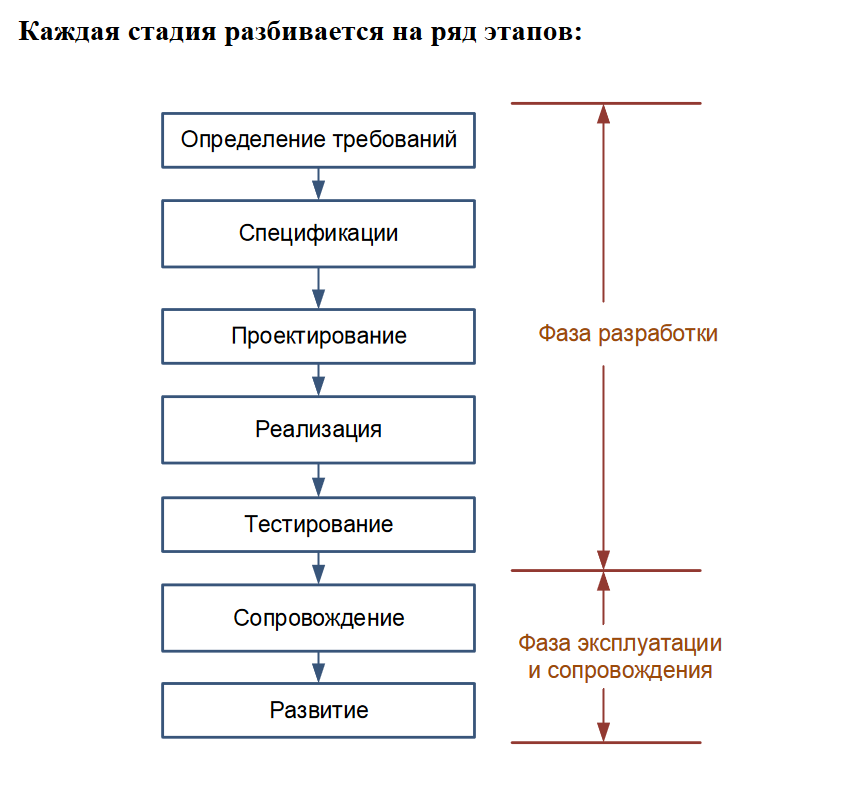
Pull request — это заявка на слияние кода из разных веток.

Ревью кода (code review) — процесс обсуждения изменений кода после совместного создания коммита и перед окончательным слиянием.

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

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

* разработка
* сопровождение



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. Это ряд событий, происходящих с ПО  
в процессе его создания и  
использования.  
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



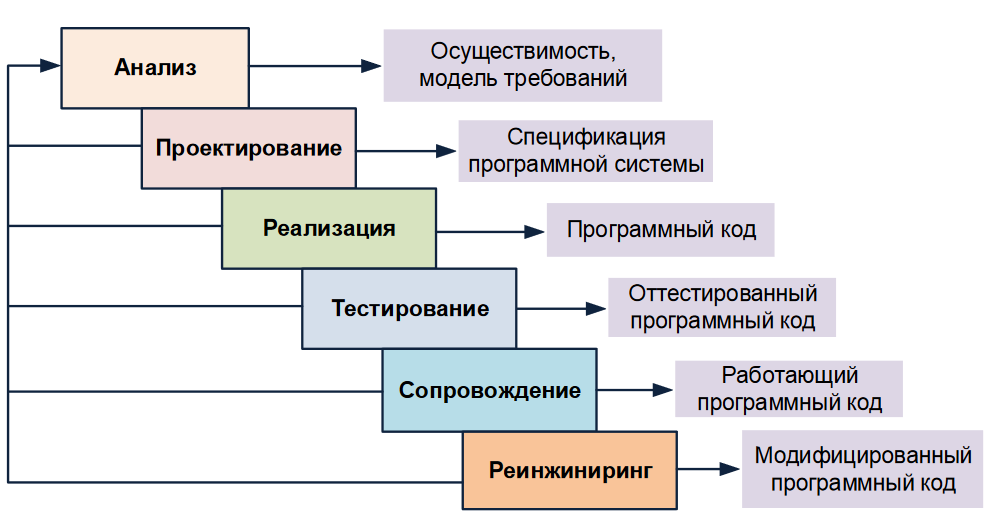
# 24. Каскадная модель жизненного цикла ПС: содержание этапов, область применения, достоинства и недостатки.

Каскадная (водопадная) модель это:

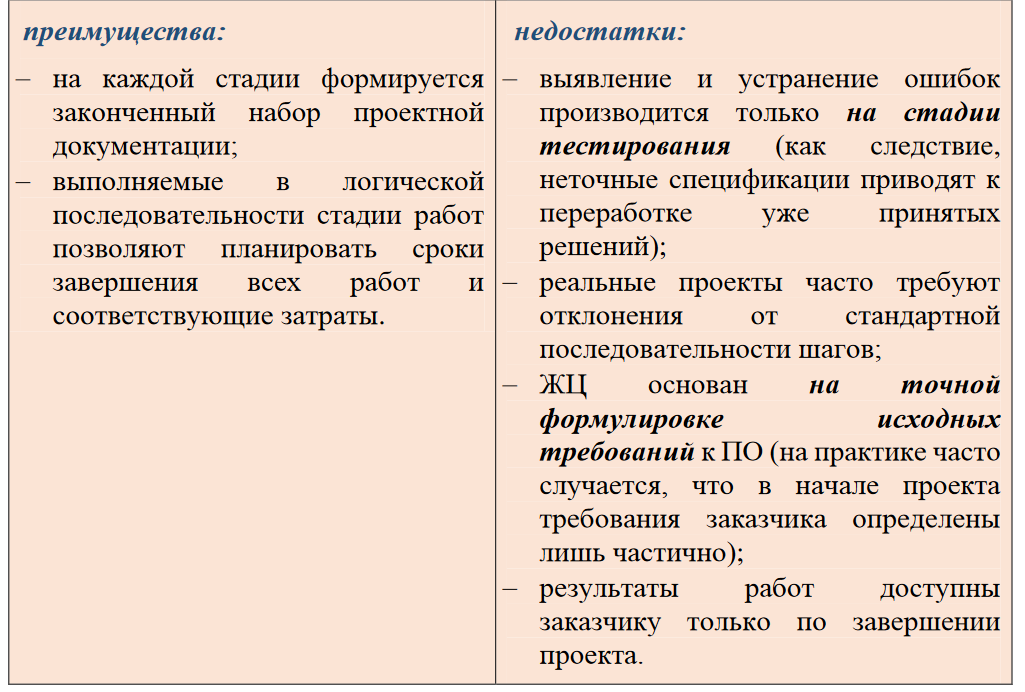
− последовательное и однократное выполнение всех этапов проекта в  
строго фиксированном порядке;

− переход на следующий этап после полного завершения работ на  
предыдущем этапе.

Этапы:



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



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

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

Достоинством процесса создания ПО, построенного на основе эволюционного подхода, является то, что спецификация может разрабатываться постепенно, по мере того как заказчик (или пользователи) осознает и сформулирует те задачи, которые должно решать программное обеспечение.

Вместе с тем данный подход имеет и некоторые недостатки.

* Многие этапы процесса создания ПО не документированы. Менеджерам проекта создания ПО необходимо регулярно документально отслеживать выполнение работ. Но если система разрабатывается быстро, то экономически не выгодно документировать каждую версию системы.
* Система часто получается плохо структурированной. Постоянные изменения в требованиях приводят к ошибкам и упущениям в структуре ПО. Со временем внесение изменений в систему становится все более сложным и затратным.
* Часто требуются специальные средства и технологии разработки ПО. Это вызвано необходимостью быстрой разработки версий программного продукта. Но, с другой стороны, это может привести к несовместимости некоторых применяемых средств и технологий, что, в свою очередь, требует наличия в команде разработчиков специалистов высокого уровня.

**Модель** применяется для разработки несложных и не критических систем, для которых главным требованием является реализация функций системы.

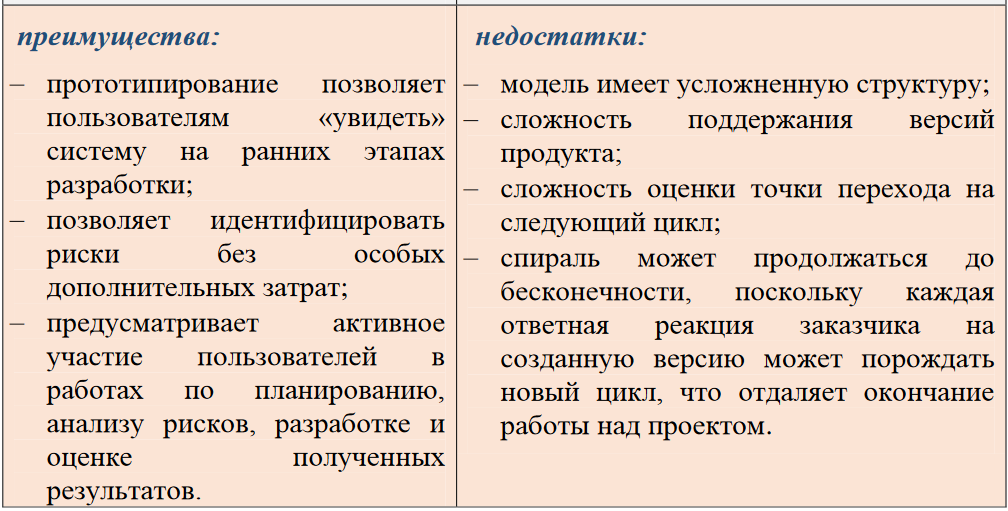
# 26. Спиральная модель разработки ПО: содержание этапов создания ПС, область применения, достоинства и недостатки.

Спиральная модель − на каждом витке спирали создается очередная версия продукта, уточняются требования проекта, определяется его качество и планируются работы следующего витка.

Область применения спиральной модели:

* при разработке систем, требующих большого объема вычислений  
  (например, систем принятия решений);
* при выполнении бизнес-проектов;
* при выполнении бизнес-проектов;  
  при выполнении проектов в области аэрокосмической промышленности, обороны и инжиниринга, где уже имеется позитивный опыт ее использования





# 27. Инкрементальная модель разработки ПО. Развитие

# инкрементального подхода. XP-процессы.

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



Развитием инкрементального подхода явилось создание некоторых новых технологий, пользующихся в настоящее время достаточным успехом: - экстремальное программирование XP (Кент Бек, 1999). Оно ориентировано на очень малые приращения функциональности.

В XP процесс создания системы делится на очень маленькие ступеньки, по сравнению с планируемыми процессами. Это приводит к тому, что первые шаги могут занимать дни или недели вместо месяцев или даже лет для каждой ступени в модели «каскад».

Сначала пишутся автоматические тесты, (тесты, которые представляют собой выполняемые фрагменты кода для автоматической проверки корректности частей (модулей) программного обеспечения) чтобы описать цели разработки.

Потом идёт кодирование, которое заканчивается в тот момент, когда все тесты проходят, и программисты не могут придумать новых тестов.

Дизайн делается теми же людьми, которые пишут код (только последняя ступень — соединение дизайна и кода является общим для всех гибких процессов). Незаконченная, но функционирующая система показывается узкому кругу пользователей (чаще всего это сами разработчики). В этот момент начинают писать тесты для следующей наиболее важной части системы.

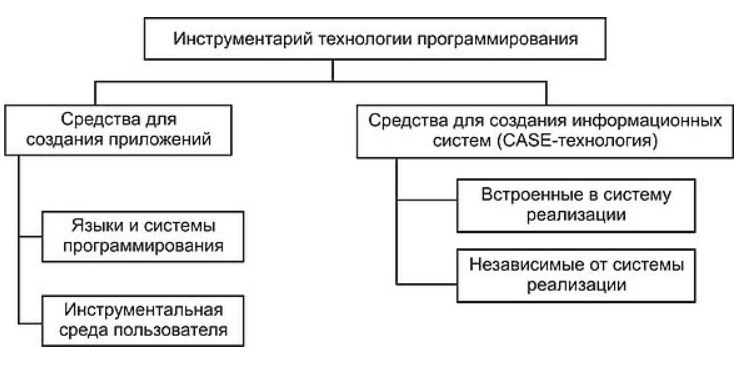
После того, как заканчивается работа на ступени, процесс переходит к следующей. Продукт не выпускается до того, как не будут завершены все ступени разработки.

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

Существуют следующие виды методологии разработки:

* Waterfall(Каскадная модель)
* Rational Unified Process (RUP) (методология “Рациональный унифицированный процесс”)
* Agile – общая методология гибкой разработки
* Rapid Application Development (RAD) (методология быстрой разработки приложений)
  + Scrum – концепция работы в условиях сорванных сроков   
    и идеологического кризиса.
* Spiral Model (Спиральная модель)
* Extreme Programming (XP)
* V-образная модель (разработка через тестирование, это усовершенствованная каскадная модель)
* Structured Analysis and Design Technique (SADT)
  + Microsoft Solutions Framework (MSF) Microsoft Operations Framework   
    (MOF)
* Personal Software Process (Модель компетентного разработчика)

Инструментарий технологии программирования — это программные продукты, предназначенные для поддержки технологии программирования



Средства для создания приложений — совокупность языков и систем программирования, инструментальные среды пользователя, а также различные программные компоненты для отладки и поддержки создаваемых программ.

Средства для создания информационных систем (ИС) и технологий поддерживают полный цикл проектирования сложной информационной системы или технологии от исследования объекта автоматизации до оформления проектной и прочей документации на информационную систему или технологию.

Требование – это утверждение, которое идентифицирует эксплуатационные, функциональные параметры, характеристики или ограничения проектирования продукта или процесса, которое однозначно, проверяемо и измеримо.   
Управление требованиями это 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 включает в себя:  
• словарь терминов предметной области;  
• описание предметной области;  
• описание ролевой системы (действующие лица);  
• описание функциональных требований;  
• описание нефункциональных требований.  
Все требования формируются на основании описания бизнес-процессов заказчика (функциональные требования).  
Описание требований в ТЗ фиксирует необходимые функции на верхнем уровне. Требования оптимально разбивать на смысловые группы по подсистемам.

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

RAD (Rapid Application Development) — методология быстрой разработки приложений, которая предполагает применение инструментальных средств визуального моделирования (прототипирования) и разработки. RAD предусматривает небольшие команды разработки, сроки до 4 месяцев и активное привлечение заказчика с ранних этапов.

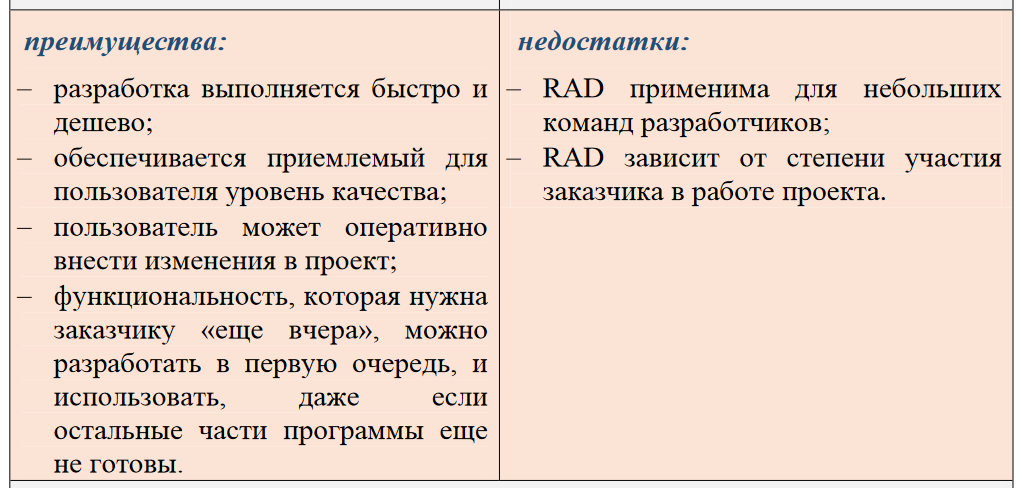
При использовании методологии быстрой разработки приложений жизненный цикл информационной системы состоит из четырех фаз:

· фаза анализа и планирования требований;

· фаза проектирования;

· фаза построения;

· фаза внедрения.

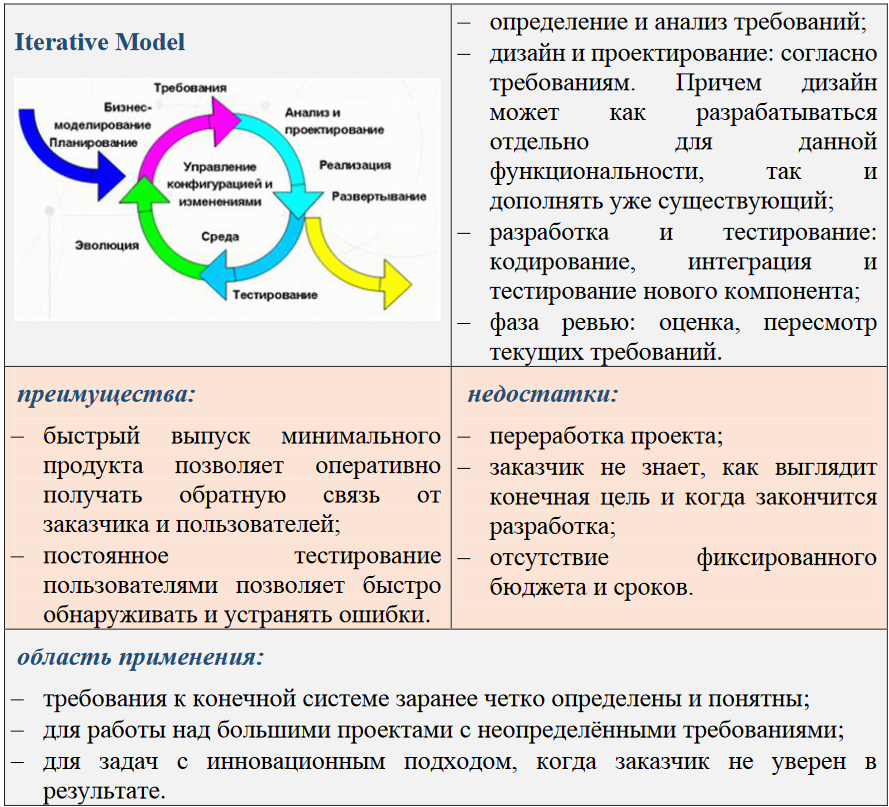


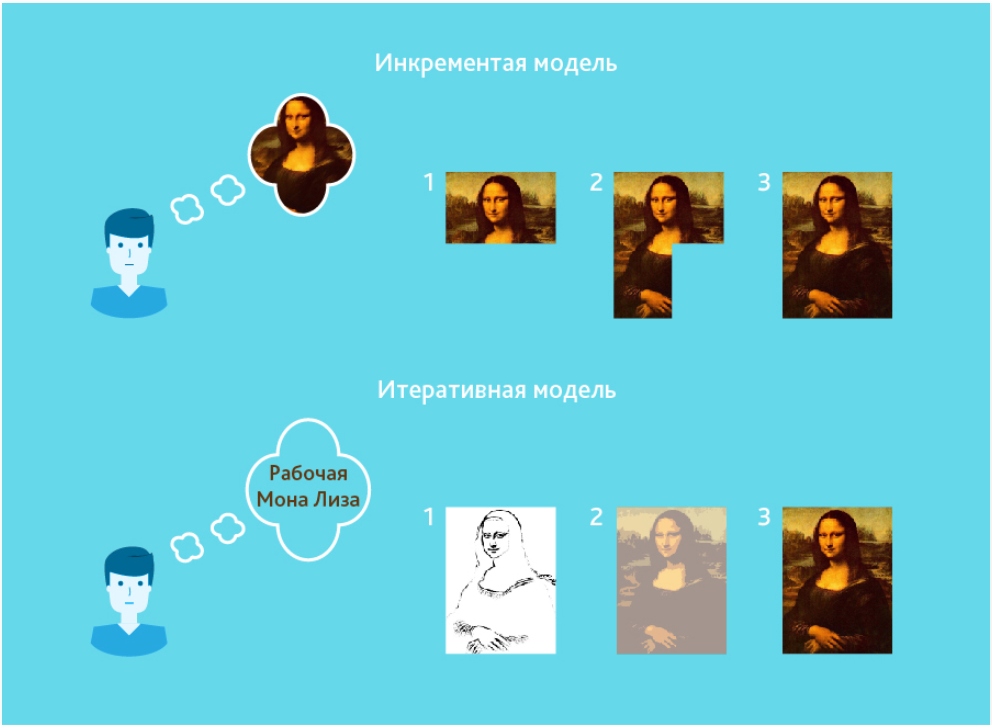
Область применения:  
− для проектов, которые легко разделить на независимые или слабосвязанные модули;  
− если требования к программному обеспечению быстро меняются;  
− в условиях ограниченного бюджета;  
− нет ясного представления, как должен выглядеть и работать продукт;  
− разработка ведется командой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

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

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

Итеративная разработка ПО — это процесс создания программного обеспечения, который осуществляется небольшими этапами, в ходе которых ведется анализ полученных промежуточных результатов, выдвигаются новые требования и корректируются предыдущие этапы работы.





На диаграмме показана итерационная «разработка» Мона Лизы. Как видно, в первой итерации есть лишь набросок Джоконды, во второй — появляются цвета, а третья итерация добавляет деталей, насыщенности и завершает процесс. В инкрементной же модели функционал продукта наращивается по кусочкам, продукт составляется из частей. В отличие от итерационной модели, каждый кусочек представляет собой целостный элемент.

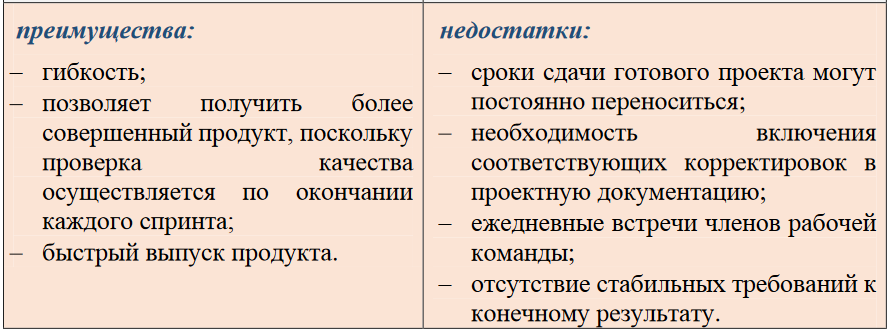
# 31. Методологии гибкой разработки ПО: Scrum, Kanban, Extreme Programming. Различия между Agile и традиционным подходом к разработке ПО. Преимущества и недостатки технологий быстрой разработки программного обеспечения.

Scrum кратко: agile-подход к разработке и управлению проектами:  
− деление работы на части, которые называются спринтами (две недели);  
− спринты планируются исходя из требований для данного момента;  
− относительная оценка времени выполнения работ;  
− ревью каждого спринта, чтобы понять, как он прошёл и что можно  
было бы улучшить;  
− фидбек (обратная связь) по поставляемому продукту;  
− ежедневные собрания (15 мин.).

Agile/Kanban (с японского переводится, как «карточка»)  
Kanban кратко: agile-подход к разработке и управлению проектами, ориентированный на следующие задачи задачи:  
− еженедельные собрания;  
− непрерывная разработка;  
− визуализация процесса на доске;  
− решение сначала самых важных задач;  
− поэтапные улучшения.  
Цель: анализ рабочего процесса и поиск точек для улучшения.

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.



****

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

Personal Software Process определяет требования к компетенциям разработчика  
Цели PSP. PSP помогает разработчикам:  
− улучшить оценку и планирование навыков;  
− управлять качеством проектов;  
− снизить количество ошибок в своих разработках.

Один из основных аспектов PSP — использование накопленной статистики для анализа и улучшения показателей процесса разработки. Сбор статистики включает 4 элемента:  
− скрипты.  
− оценки. Оценки включают 4 основных элемента:  
 • размер — оценка размера для части продукта. Например,  
количество строк кода (LOC — Lines Of Code).  
 • качество — количество ошибок в продукте.  
 • усилия — оценка времени, требующегося для завершения  
задачи, обычно записываемое в минутах.  
 • планирование — оценка хода проекта, перемещаемая между  
планируемыми и завершенными пунктами.  
− стандарты кодирования. Применение стандартов к процессу может  
обеспечить точные и постоянные данные.  
− формы.

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

КонструированиеПО - создание работающего ПО с привлечением методов верификации, кодирования и тестирования компонентов. К инструментам конструирования ПО отнесены языки программирования и конструирования, а также программные методы и инструментальные системы (компиляторы, СУБД, генераторы отчетов, системы управления версиями, конфигурацией, тестированием и др.). К формальным средствам описания процесса конструирования ПО, взаимосвязей между человеком и компьютером и с учетом среды окружения отнесены, например, структурные диаграммы Джексона.

Существуют три стратегии конструирования ПО:  
1. Однократный проход (водопадная стратегия) – линейная  
последовательность этапов конструирования.  
2. Инкрементная стратегия. В начале процесса определяются все  
пользовательские и системные требования, оставшаяся часть  
конструирования выполняется в виде последовательности версий. Первая версия реализует часть запланированных  
возможностей, следующая версия реализует дополнительные  
возможности и т. д., пока не будет получена полная система.  
3. Эволюционная стратегия. Система также строится в виде  
последовательности версий, но в начале процесса определены не  
все требования. Требования уточняются в результате разработки  
версий.

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

Технология разработки программного обеспечения(ПО) представляет собой комплекс организационных мер, операций и приемов, направленных на разработку программных продуктов высокого качества в рамках отведенного бюджета и в срок. Технологии включают методики, методологии, средства и процедуры разработки ПО.

В настоящее время существует достаточно много различных методик разработки программного обеспечения

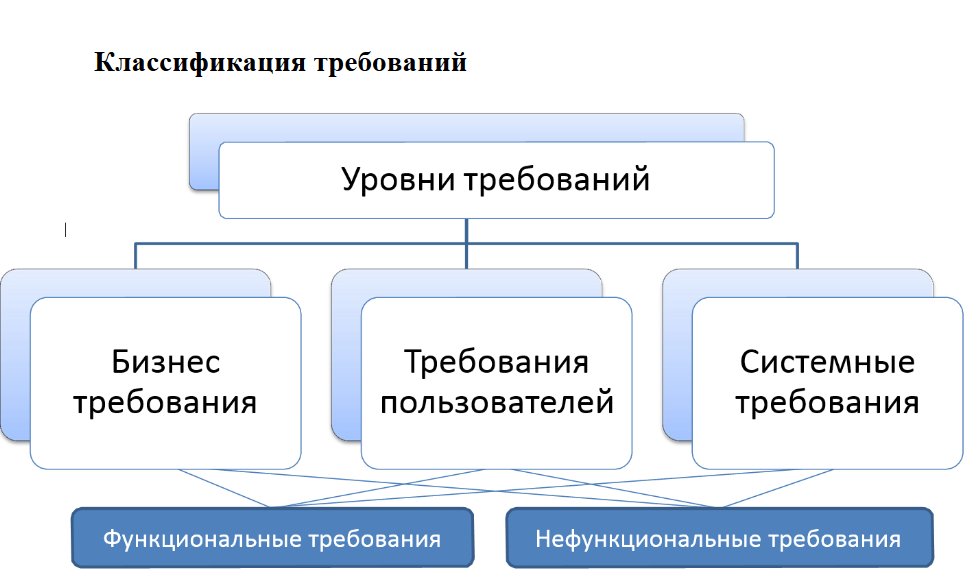
Методики, базирующиеся на каскадной модели, характеризуются тем, что переход на следующую стадию проектирования осуществляется только после того, как будет завершена работа на текущей стадии.

Спиральная модель жизненного цикла ПО лежит в основе методологии создания ИТ-решений компании Microsoft - MSF (MicrosoftSolutionFramework). В данной методологии компания Microsoft отразила свое видение на процессы создания программных систем различного назначения.

К итеративным методам разработки ПО относится методология, созданная компанией RationalSoftware - RationalUnifiedProcess (RUP).

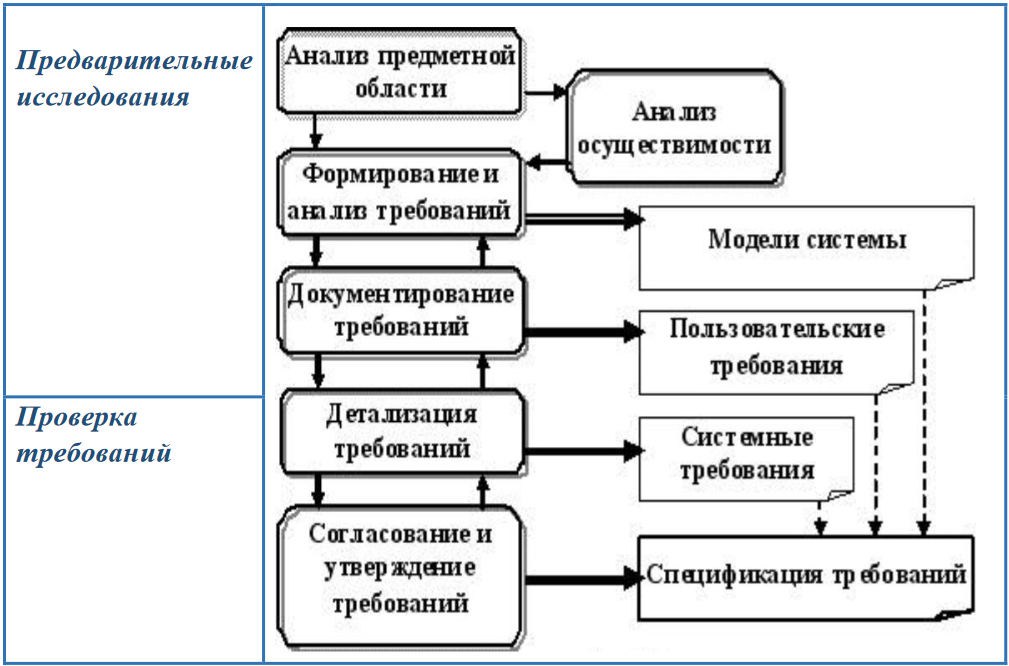
Требование – это утверждение, которое идентифицирует эксплуатационные, функциональные параметры, характеристики или ограничения проектирования продукта или процесса, которое однозначно, проверяемо и измеримо.   
Управление требованиями это процесс, включающий:  
идентификацию, выявление, документацию, 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.



Функциональные требования определяют функции, которые выполняет  
система, и зависят от потребностей пользователей и типа решаемой задачи. Нефункциональные требования определяют характеристики и ограничения системы и не связаны непосредственно с функциональными требованиями. Они формируются на основе имеющихся атрибутов качества, требований к внешнему интерфейсу и ограничений.

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



Цели разработки требований

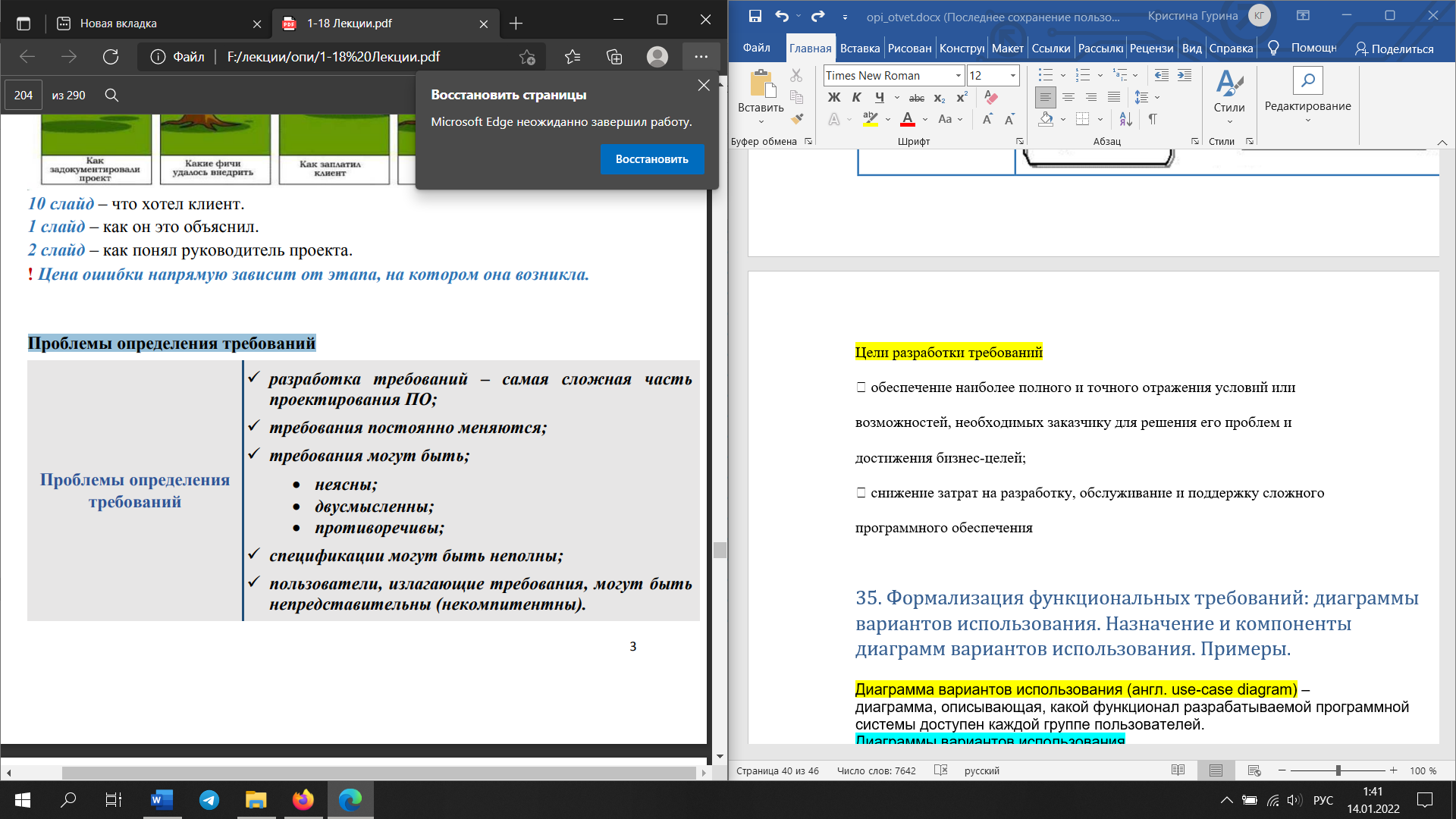
 обеспечение наиболее полного и точного отражения условий или

возможностей, необходимых заказчику для решения его проблем и

достижения бизнес-целей;

 снижение затрат на разработку, обслуживание и поддержку сложного

программного обеспечения



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

Свойства требований:

 корректность (correct);

 однозначность (unambiguous);

 полнота (complete);

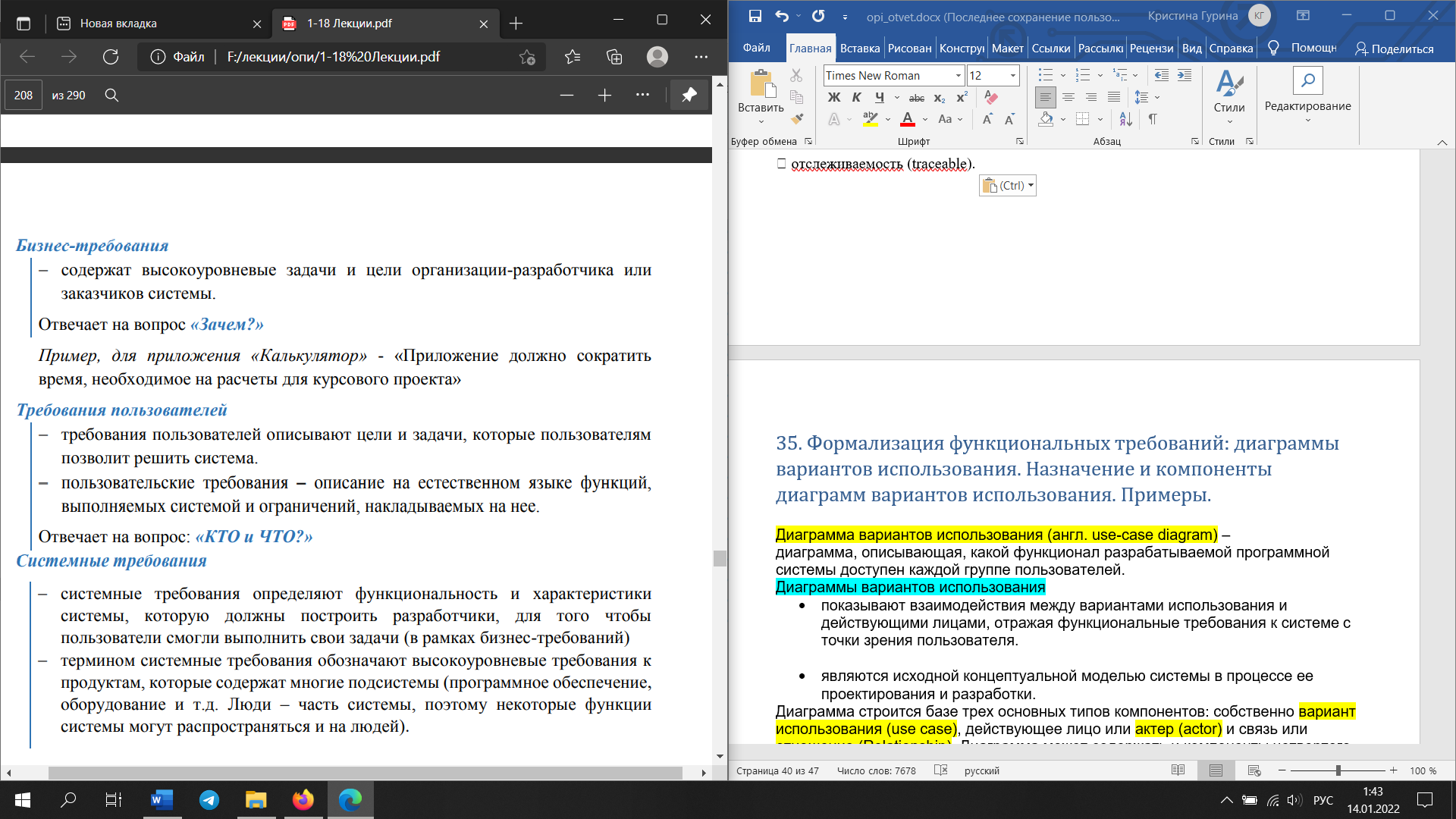
 непротиворечивость (consistent);

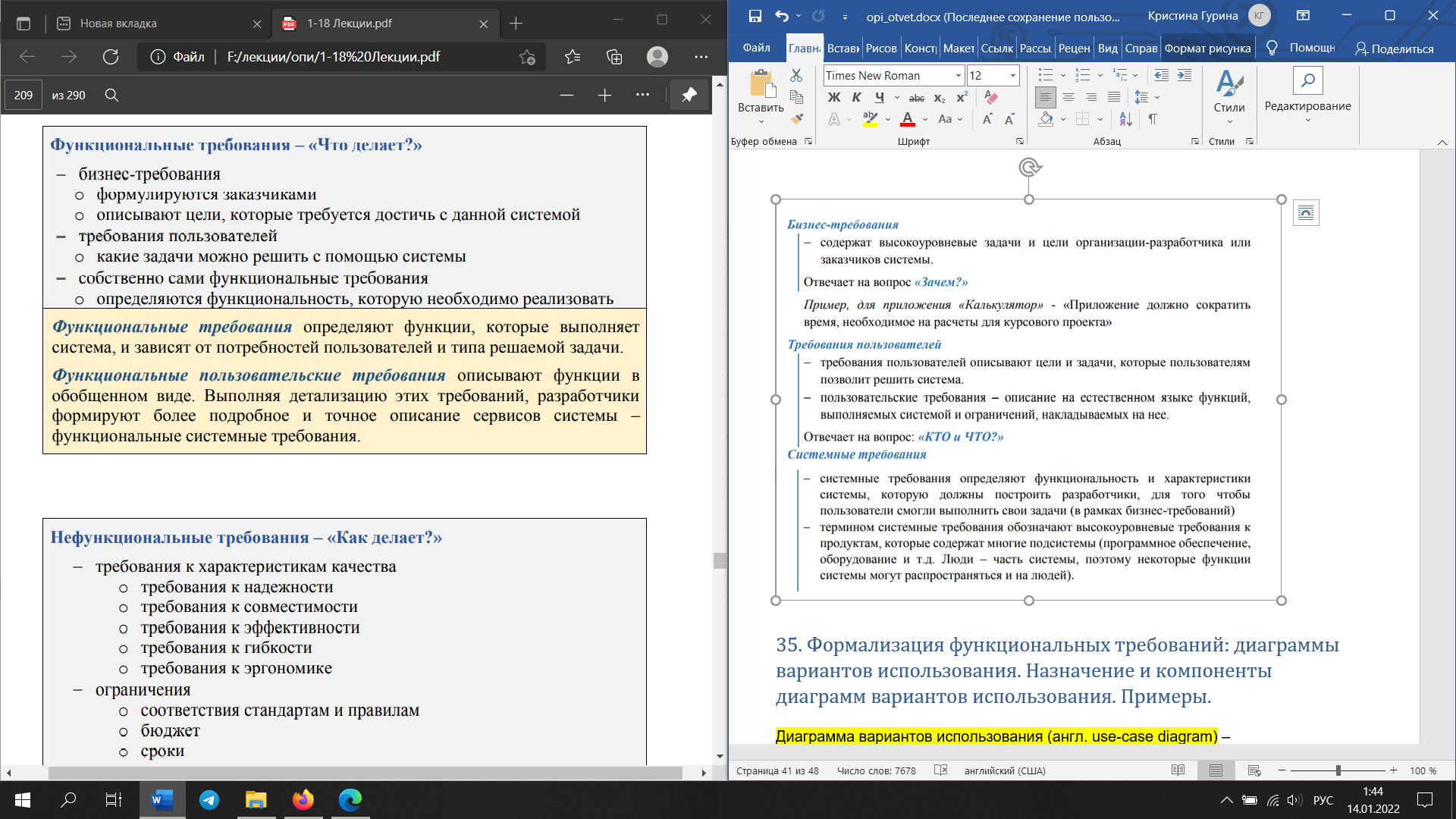
 приоритезация (prioritized);

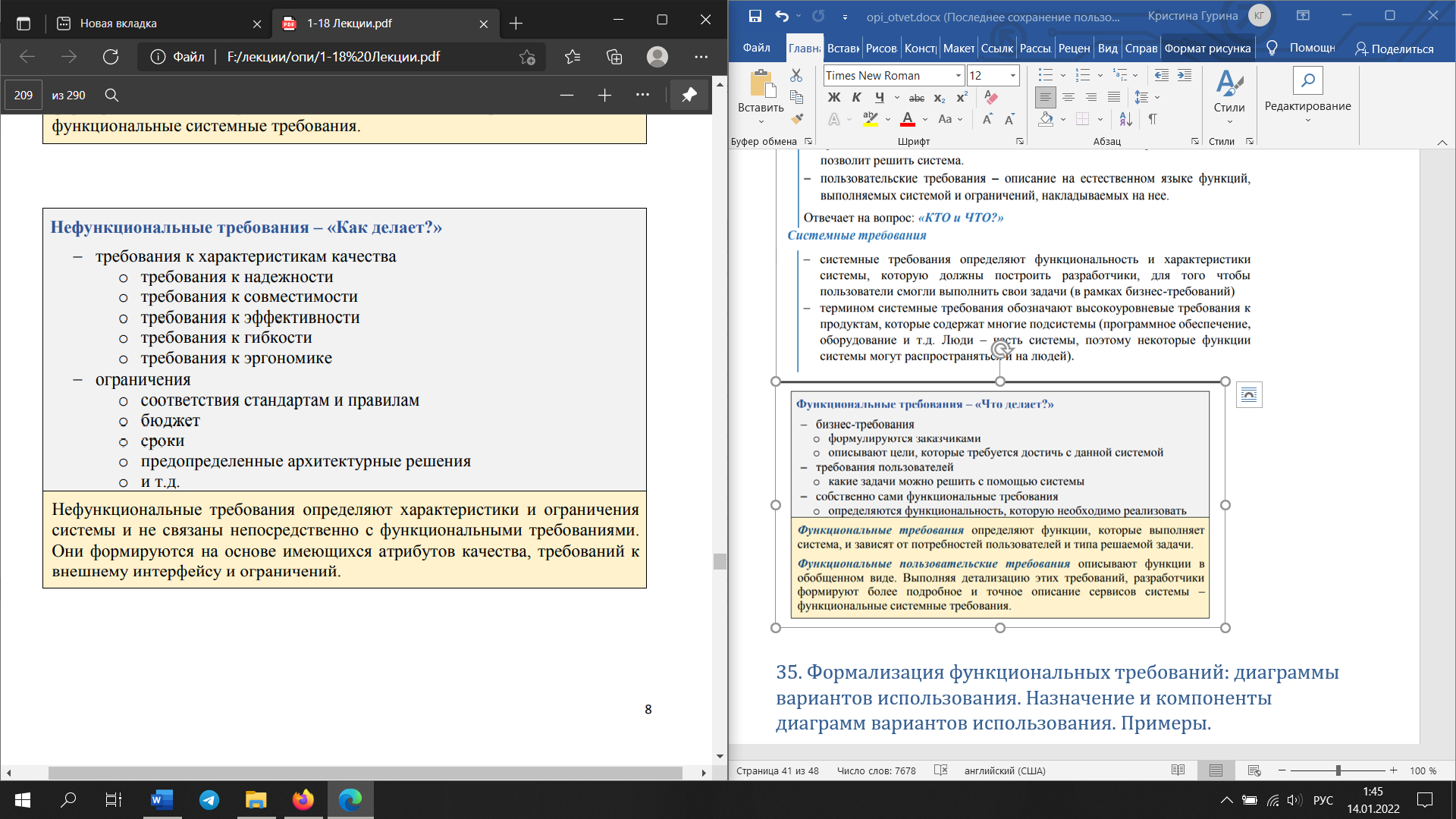
 проверяемость (verifiable);

 модифицируемость (modifiable);

 отслеживаемость (traceable).







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

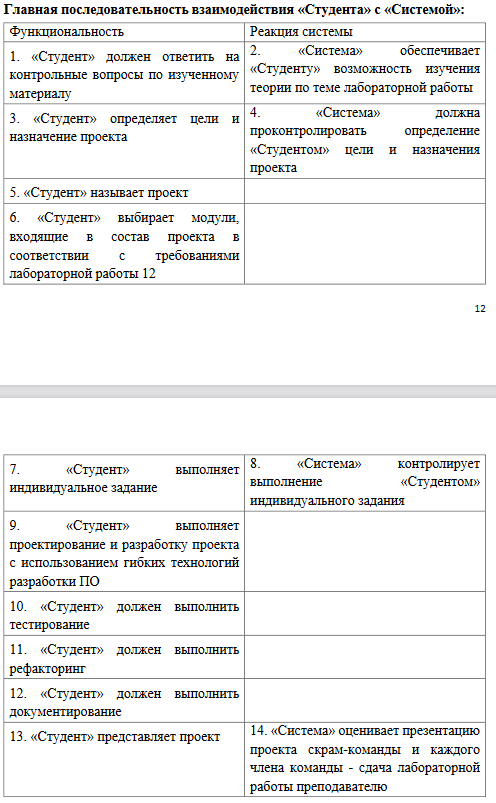
Диаграмма вариантов использования (англ. use-case diagram) –  
диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей.

Диаграммы вариантов использования

* показывают взаимодействия между вариантами использования и  
  действующими лицами, отражая функциональные требования к системе с точки зрения пользователя.
* являются исходной концептуальной моделью системы в процессе ее проектирования и разработки.

Диаграмма строится базе трех основных типов компонентов: собственно вариант использования (use case), действующее лицоилиактер (actor)исвязьили отношение (Relationship)*.* Диаграмма может содержать и компоненты четвертого типа – интерфейсы (interface)*,* а такжепримечания(notes) - произвольные текстовые комментарии разработчика, имеющие отношение к компонентам UseCase-диаграммы*.*

Актером будем называть внешнюю по отношению к ПС сущность, которая может взаимодействовать с системой. При взаимодействии актера с системой последняя выполняет ряд работ, которые образуют вариант использования системы (use case). Таким образом, каждый ВИ, по существу, есть некоторое функциональное требование к системе (которое может быть разбито на несколько более мелких). Отношения между ВИ служат для извлечения из ВИ характерных фрагментов, которые могут рассматриваться как отдельные абстрактные ВИ.

Пример  
Рассмотрим выполнение 12-ой лабораторной работы студентами.  
Моделируемая система – «Ознакомление с технологиями разработки ПО».  
Цель – см. цель лабораторной работы 12.  
Актер – «Студент» (взаимодействует с системой).  
Функциональность:  
• «Студент» должен ответить на контрольные вопросы по изученному  
материалу;  
• «Студент» определяет цели и назначение проекта;  
• «Студент» выбирает название проекта;  
• «Студент» выбирает модули, входящие в состав проекта в соответствии с требованиями лабораторной работы 12;  
• «Студент» выполняет индивидуальное задание;  
• «Студент» выполняет проектирование и разработку проекта с  
использованием гибких технологий разработки ПО;  
• «Студент» должен выполнить тестирование;  
• «Студент» должен выполнить рефакторинг;  
• «Студент» должен выполнить документирование;  
• «Студент» представляет проект.  
При взаимодействии с актером «Студент» система должна позволять выполнять набор функциональных требований, при этом важна реакция системы.  
  


Альтернативная последовательность  
7а: «Студент» не выполняет индивидуальное задание.  
7в: Реакция системы: «Студент» не допускается до экзамена



# 36.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.

* Верификация (Verification) – это процесс оценки системы или её компонентов с целью определения удовлетворяют ли результаты текущего этапа разработки условиям, сформированным в начале этого этапа. Т.е. выполняются ли наши цели, сроки, задачи по разработке проекта, определенные в начале текущей фазы.
* Валидация (Validation) – это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе.
* 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.
* 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.
* 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.
* Баг/Дефект Репорт (Bug Report) – это документ, описывающий ситуацию или последовательность действий, приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.
* Тестовое Покрытие (Test Coverage) – это одна из метрик оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода.
* 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
* Время Прохождения Тест Кейса (Test Case Pass Time) – это время от начала прохождения шагов тест кейса до получения результата теста

  
Цели тестирования  
• убедиться, что ПО отвечает заявленным требованиям.  
• выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим требованиям.

Задачи тестирования  
• убедиться, что ПО отвечает заявленным требованиям.  
• выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим требованиям.  
• предотвратить как можно больше дефектов  
• проверить, что известные дефекты устранены  
• проверить, что при устранении известных дефектов, не было внесены новые дефекты  
• информировать всех заинтересованных лиц о качестве системы.  
Принципы тестирования  
1. Тестирование демонстрирует наличие дефектов.  
2. Исчерпывающее тестирование недостижимо.  
3. Раннее тестирование.  
4. Скопление дефектов.  
5. Парадокс пестицида.  
6. Тестирование зависит от контекста.  
7. Заблуждение, что ошибки отсутствуют.

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

Методы тестирования ПО:

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

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.

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.

Ручное тестирование (manual testing) — часть процесса тестирования на этапе контроля качества в процессе разработки программного обеспечения. Оно проводится тестировщиками или обычными пользователи путем моделирования возможных сценариев действия пользователя.

Ручное тестирование заключается в выполнении задокументированной процедуры, где описана методика выполнения тестов. Методика задает порядок тестов и для каждого теста – список значений параметров, который подается на вход со список результатов на выходе. Так как процедура предназначена для выполнения человеком, в ее описании для краткости могут использоваться некоторые значения по умолчанию, ориентированные на здравый смысл, или ссылки на информацию, хранящуюся в другом документе.

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

Документация на программное обеспечение — это документы, сопровождающие некоторое программное обеспечение (ПО) — программу или программный продукт. Эти документы описывают то, как работает программа и/или то, как её использовать

Существует четыре основных типа документации на ПО:  
• архитектурная/проектная – обзор программного обеспечения, включающий описание рабочей среды и принципов, которые должны быть использованы при создании ПО;  
• техническая – документация на код, алгоритмы, интерфейсы, API;  
• пользовательская – руководства для конечных пользователей, администраторов системы и другого персонала;  
• маркетинговая.

Стандарт документирования  
ГОСТ при разработке программных продуктов дает возможность:  
• унифицировать программные изделия для взаимного обмена и применения ранее разработанных программ в новых разработках;  
• снизить трудоемкость и повысить эффективность разработки, сопровождения, изготовления и эксплуатации программных продуктов;  
• автоматизировать изготовление и хранение программной документации.  
• основу отечественной нормативной базы в области документирования ПО составляет комплекс стандартов Единой системы программной  
документации (ЕСПД).  
• стандарты ЕСПД в основном охватывают ту часть документации, которая создается в процессе разработки ПО, и связаны, по большей части, с документированием функциональных характеристик ПО.

Значение ЕСПД  
• стандарты ЕСПД вносят элемент упорядочения в процесс документирования ПС;  
• предусмотренный стандартами ЕСПД состав программных документов не является «жестким»: стандарты позволяют вносить в комплект документации на ПС дополнительные виды;  
• стандарты ЕСПД позволяют мобильно изменять структуры и содержание установленных видов программной документации исходя из требований заказчика и пользователя.

# 39.Управление командой проекта.

Управление проектами – это управление и организация процесса разработки ПО для достижения поставленной цели в установленное время и в рамках бюджета.

Управление командой проекта – это деятельность, связанная с принятием и реализацией решений, регулирующих работу связанных с проектом людей.

Команда проекта, возглавляемая руководителем, создается для управления проектом на срок его реализации. Ее форма сложит отражением существующей организационной управленческой структуры, структуры разделения функций, ответственности и обязонностей.  
Модели организации команд

* Административная модель (теория X)  
  Характерные черты модели:  
  − властная пирамида – решения принимаются сверху-вниз;  
  − четкое распределение ролей и обязанностей;  
  − четкое распределение ответственности;  
  − следование инструкциям, процедурам, технологиям;  
  − роль менеджера: планирование, контроль, принятие основных решений
* Модель хаоса (теория Y)  
  Характерные черты модели:  
  − отсутствие явно выраженных признаков власти;  
  − роль менеджера – поставить задачу, обеспечить ресурсами, не мешать и следить, чтобы не мешали другие;  
  − отсутствие инструкций и регламентированных процедур;  
  − индивидуальная инициатива – решения по проблеме принимается там, где проблема обнаружена;  
  − процесс творческий на основе дружеской соревновательности
* Открытая архитектура (теория Z)  
  Характерные черты модели:  
  − адаптация к условиям работы – если нужна помощь, то обсуждаем идеи совместно;  
  − коллективное обсуждение проблем –принятое решение, обязательно для всех;  
  − распределенная ответственность;  
  − динамическое изменение состава рабочих групп в зависимости от текущих задач;  
  − отсутствие специализации;  
  − задача менеджера – активное участие в процессе, контроль обсуждений, обеспечение возможности активного участия всех
* Модель проектной группы MSF for Agile Software Development Построение команды в MSF соответствует ряду концепций (key concepts), первые три кажутся самоочевидными, остальные являются «ноу-хау»:  
  − концентрация на нуждах заказчика (customer-focused mindset);  
  − нацеленность на конечный результат (product mindset);  
  − установка на отсутствие дефектов (zero-defect mindset);  
  − «Проектная группа – команда равных» (teem of peers);  
  − стремление к самосовершенствованию (willingness to learn).