УЧЕБНО-НАУЧНО-ПРОИЗВОДСТВЕННЫЙ КОМПЛЕКС
«МЕЖДУНАРОДНЫЙ УНИВЕРСИТЕТ КЫРГЫЗСТАНА»

«УТВЕРЖДЕНО»
Ректор НОУ УНИК «МУК»
к.т.н., доцент Савченков Е.Ю.
_____
«16» 10 2018 г.

МАГИСТРАТУРА
Кафедра «**Компьютерных информационных систем и управления**»
Учебно-методический комплекс дисциплины
Технология разработки программного обеспечения

Направление: **710100 «Информатика и вычислительная техника»**
Профиль: **Компьютерные информационные системы**
Академическая степень - **магистр**

Форма обучения – **очная**

График проведения модулей 1, 2 - семестры

неделя	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8 Мод.**	**9**	**10**	**11**	**12**	**13**	**14**	**15 Мод.**	**16**
лекц. зан.	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
лаб. зан.	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

«РАССМОТРЕНО»
Протокол заседания кафедры
«КИСиУ»
№ 2 от 16.10.2018
Зав. кафедрой д.т.н., проф. Миркин Е.Л.
_____

«СОГЛАСОВАНО»
Проректор по академ. вопросам
проф. Мадалиев М.М.
_____

Составитель _____ к.т.н., и. о. доцент
Нежинских С.С.

Директор Научной библиотеки _____ Асанова Ж.Ш.

БИШКЕК 2018

ОГЛАВЛЕНИЕ

АННОТАЦИЯ

Цель дисциплины (модулей)заключаетсяформированиеу выпускникасистемы понятий, знаний, умений и навыков в области современного программирования на основе использования структурной и объектно-ориентированной технологии программирования.Задачамидисциплины(модулей)являются:обучение основным подходам к проектированию, разработке и использованию программ; дать обучающимся знание технологий разработки программных средств с использованием универсальных языков программирования; рассмотреть использование объектно-ориентированного подхода в проектировании программ; получение практических навыков использования технологии обобщенного программирования, использования стандартных библиотек классов и шаблонов.. На изучение дисциплины отводится90часов.Рубежный контрольуспеваемостипроводится на 5,10, 15неделях.Формы текущего контроля:опрос, проверка задания, посещаемость.Форма рубежного контроля— модульная работа.Форма итоговогоконтроля— экзамен.

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ДИСЦИПЛИНЫ (МОДУЛЕЙ)

1. Пояснительная записка

1.1. Миссия и стратегия

Миссией является подготовка профессионалов в своей будущей деятельности путем создания новых знаний и умений, способствование сохранению, приумножению научных, культурных и нравственных ценностей общества. Активизация разработки и внедрения новых организационных форм и методов обучения, максимально мотивирующих активную творческую работу, как обучающихся, так и преподавателей.

Стратегии развития - модернизация образовательной деятельности университета – совершенствование образовательного процесса в соответствии с требованиями Болонского процесса.

1.2. Цель и задачи дисциплины (модулей)

Цель дисциплины: формирование у выпускника системы понятий, знаний, умений и навыков в области современного программирования на основе использования структурной и объектно- ориентированной технологии программирования.

Задачи дисциплины:

- обучение основным подходам к проектированию, разработке и использованию программ;
- дать обучающимся знание технологий разработки программных средств с использованием универсальных языков программирования;
- рассмотреть использование объектно-ориентированного подхода в проектировании программ;
- получение практических навыков использования технологии обобщенного программирования, использования стандартных библиотек классов и шаблонов.

1.3. Формируемые компетенции, а также перечень планируемых результатов обучения по дисциплине (модулю) (знания, умения владения), сформулированные в компетентностном формате.

Дисциплина (модуль) направлена на формирование следующих компетенций:

- общенаучными (ОК):
  - способностью к самоорганизации и к самообразованию (ОК-7);
- профессиональными (ПК):
  - способностью к организации учебной деятельности в конкретной предметной области (математика, физика, информатика) (ПК-9).

В результате освоения дисциплины обучающийся должен демонстрировать следующие результаты образования:

1. Знать:
   - основные базовые элементы языка программирования: простые и структурированные типы данных, переменные, выражения, функции;
   - основные управляющие структуры языка программирования;
   - стили и методы программирования;
   - возможности инструментальных средств программирования в различных средах;
   - современные средства разработки и анализа программных средств;
2. Уметь:
   - работать в среде программирования;
   - реализовывать алгоритм решения задачи на языке программирования;
   - использовать среду программирования в своей будущей профессиональной деятельности;

- оценивать качественные и количественные характеристики программного продукта;

3. Владеть:
   - методами и технологиями программирования в программных средах;
   - опытом работы в различных средах программирования;
   - навыками отладки, тестирования и оптимизации программ.

1.4. Место дисциплины (модулей) в структуре ООП ВПО

Дисциплина (модуль) «Современные технологии создания WEB приложений» является частью цикла (блока) дисциплин учебного плана по направлению подготовки 710100 «Информатика и вычислительная техника»,специальностиКомпьютерные информационные системы. Для освоения дисциплины (модулей) необходимы компетенции, сформированные в ходе изучения следующих дисциплин и прохождения практик:Основы программирования,Математическая логика и теория алгоритмов.

2. Структура дисциплины (модулей)

Структура дисциплины (модулей) для очной формы обучения

Общая трудоемкость дисциплины составляет3кредита,90ч., в том числе аудиторная работа обучающихся с преподавателем48ч., самостоятельная работа обучающихся42ч.

№ п/п	*Раздел, Темы Дисциплины*	*Семестр*	*Неделя семестра*	*Виды учебной работы, включая самостоятельную работу студентов и трудоемкость (в часах)*				*Формы текущего контроля успеваемости (по неделям семестра) Форма промежуточной аттестации (по семестрам)*
				лекции	*Сем. Заня/лаб. заня*	*СРС*	*СРСиП*	
	Модуль 1							
1.1	Стратегии разработки программных средств и систем и реализующие их модели жизненного цикла	1	1-2	2	4	2	2	опрос,проверка задания,посещаемость
1.2	Выбор модели жизненного цикла для конкретного проекта	1	3-5	3	6	3	3	опрос, проверка задания, посещаемость

2.1	Классические методологии разработки программных средств	1	6-7	2	4	4	4	опрос, проверка задания, посещаемость
2.2	Case-технологии структурного анализа и проектирования программных средств	1	8-10	3	6	4	4	опрос, проверка задания, посещаемость
	Модуль 2							
3.1	Методология объектно-технологии ориентированного анализа и проектирования сложных систем	1	11-13	3	2	4	4	опрос, проверка задания, посещаемость
3.2	Инструментальные средства разработки программного обеспечения	1	14-15	2	4	4	4	опрос, проверка задания, посещаемость
4	Консультация	1	16	1	2	0	0	

3. Содержание дисциплины (модулей)

Модуль 1

Тема 1.1. Стратегии разработки программных средств и систем и реализующие их модели жизненного цикла.

Тема1.2. Выбор модели жизненного цикла для конкретного проекта.

Модуль 2

Тема2.1. Классические методологии разработки программных средств.

Тема2.2. Case-технологии структурного анализа и проектирования программных средств.

Модуль 3

Тема3.1. Методология объектно-технологии ориентированного анализа и проектирования сложных систем.

Тема 3.2. Инструментальные средства разработки программного обеспечения.

4. Конспект лекций

Модуль 1

Тема 1.1.

Стратегии разработки программных средств и систем. Модели жизненного цикла, реализующие каскадную стратегию разработки программных средств и систем. Модели быстрой разработки приложений. Модели жизненного цикла, реализующие инкрементную стратегию разработки программных средств и систем. Модели жизненного цикла, реализующие эволюционную стратегию разработки программных средств и систем.

Тема 1.2.

Классификация проектов по разработке программных средств и систем. Процедура выбора модели жизненного цикла программных средств и систем. Адаптация модели жизненного цикла разработки программных средств и систем к условиям конкретного проекта.

Модуль 2

Тема2.1.

Структурное программирование. Модульное проектирование программных средств. Методы нисходящего проектирования. Методы восходящего проектирования. Оценка структурного разбиения программы на модули.

Тема 2.2.

Общие сведения о CASE-технологии технологиях. Методология функционального моделирования IDEF0. Методология структурного анализа потоков данных DFD. Методология информационного моделирования IDEF1X.

Модуль 3

Тема 3.1.

Основы объектно-технологии ориентированного анализа и проектирования. Математические основы объектно-технологии ориентированного анализа и проектирования.

Тема 3.2.

История развития CASE-технологии средств. Базовые принципы построения CASE-технологии средств. Основные функциональные возможности CASE-технологии средств. Классификация CASE-технологии средств.

5. Информационные и образовательные технологии

Информационные и образовательные технологии

№ п/п	Наименование раздела	Виды учебной работы	Формируемые компетенции (указывается код компетенции)	Информационные и образовательные технологии
1	Модуль №1.	Лекция	ОК-7,ПК-9	Лекция-визуализация с применением слайд-проектора, Дискуссия, Лекция с разбором конкретных ситуаций
		Лабораторная работа	ОК-7,ПК-9	Дискуссия,Консультирование с разбором абстрактных ситуаций
		Самостоятельная работа	ОК-7,ПК-9	Использование электронного курса лекций, Консультирование и проверка заданий посредством электронной почты
2	Модуль№2.	Лекция	ОК-7,ПК-9	Лекция-визуализация с применением слайд-проектора, Дискуссия, Лекция с

				разбором конкретных ситуаций
		Лабораторная работа	ОК-7,ПК-9	Дискуссия,Консультирование с разбором абстрактных ситуаций
		Самостоятельная работа	ОК-7,ПК-9	Использование электронного курса лекций, Консультирование и проверка заданий посредством электронной почты

6. Фонд оценочных средств для текущего, рубежного и итогового контролей по итогам освоению дисциплины (модулей)

6.1. Перечень компетенций с указанием этапов их формирования в процессе освоения дисциплины

№ п/п	Контролируемые разделы дисциплины (модулей)	Код контролируемой компетенции (компетенций)	Наименование оценочного средства
1	Модуль №1,№2,№3	ОК-7	опрос,выполнение лабораторных работ
2	Модуль №1,№2,№3	ПК-9	опрос,выполнение лабораторных работ

6.2. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности

Форма контроля	Срок отчетности	Макс. количество баллов	
		За одну работу	Всего
Текущий контроль: - опрос	1, 2, 3, 4, 5, 6, 7, 8 недели	8баллов	До 40 баллов
-выполнение лабораторных работ	1, 2, 3, 4, 5, 6, 7, 8 недели	6баллов	До30 баллов
- посещаемость	1, 2, 3, 4, 5, 6, 7, 8 недели	0,2	10 баллов
Рубежный контроль: (сдача модуля)	8 неделя	100%×0,2=20 баллов	
Итого за I модуль			До 100 баллов

Форма контроля	Срок отчетности	Макс. количество баллов

		За одну работу	Всего
Текущий контроль: - опрос -выполнение лабораторных работ - посещаемость	 9, 10, 11, 12, 13, 14, 15 недели 9, 10, 11, 12, 13, 14, 15 недели 9, 10, 11, 12, 13, 14, 15 недели	 8баллов 6баллов 0,2	 До 40 баллов До 30 баллов 10 баллов
Рубежный контроль: (сдача модуля)	15 неделя	100%×0,2=20 баллов	
Итого за IIмодуль			До 100 баллов
Итоговый контроль(экзамен)	Сессия	ИК = Бср × 0,8 + Бэкз × 0,2	

Полученный совокупный результат (максимум 100 баллов) конвертируется в традиционную шкалу:

Рейтинговая оценка (баллов)	**Оценка экзамена**
От 0 до 54	неудовлетворительно
от 55 до 69 включительно	удовлетворительно
от 70 до 84 включительно	хорошо
от 85 до 100	отлично

6.3.Описание показателей и критериев оценивания компетенций на различных этапах их формирования, описание шкал оценивания

Текущий контроль(0 - 80 баллов)

При оценивании посещаемости, опроса и выполнениилабораторных работ учитываются:

- посещаемость (10баллов)
- степень раскрытия содержания материала (25баллов);
- изложение материала (грамотность речи, точность использования терминологии и символики, логическая последовательность изложения материала (20баллов);
- знание теории изученных вопросов, сформированность и устойчивость используемых при ответе умений и навыков (25баллов).

Рубежный контроль(0 – 20 баллов)

При оценивании контрольной работы учитывается:

- полнота выполненной работы (задание выполнено не полностью и/или допущены две и более ошибки или три и более неточности) –10баллов;
- обоснованность содержания и выводов работы (задание выполнено полностью, но обоснование содержания и выводов недостаточны, но рассуждения верны) – 5баллов;
- работа выполнена полностью, в рассуждениях и обосновании нет пробелов или ошибок, возможна одна неточность –5баллов.

Итоговый контроль (экзаменационная сессия) – ИК = Бср × 0,8 + Бэкз × 0,2

При проведении итогового контроля обучающийся должен ответить на 3 вопроса (два вопроса теоретического характера и один вопрос практического характера).

При оценивании ответа на вопрос теоретического характера учитывается:

- теоретическое содержание не освоено, знание материала носит фрагментарный характер, наличие грубых ошибок в ответе (0баллов);
- теоретическое содержание освоено частично, допущено не более двух-трех недочетов (10баллов);
- теоретическое содержание освоено почти полностью, допущено не более одного-двух недочетов, но обучающийся смог бы их исправить самостоятельно (20баллов);
- теоретическое содержание освоено полностью, ответ построен по собственному плану (30 баллов).

При оценивании ответа на вопрос практического характера учитывается:

- ответ содержит менее 20% правильного решения (0-9баллов);
- ответ содержит 21-89 % правильного решения (10-39баллов);
- ответ содержит 90% и более правильного решения (40баллов).

6.4. Типовые контрольные задания или иные материалы, необходимые для оценки знаний, умений, навыков и (или) опыта деятельности..

Переченьвопросов:

- Что подразумевается под технологией разработки ПО?
- Что является целью структурных методов проектирования ПС?
- Дайте определение программного продукта.
- Дайте определение системы.
- Назовите базовый стандарт в области ЖЦ ПС и систем.
- Определите понятие ЖЦ программного средства или системы.
- Определите понятие модели ЖЦ программного средства или системы.
- Определите иерархическую структуру ЖЦ ПС, регламентированную стандартом СТБ
- ИСО/МЭК 12207–2003.
- На какие группы делятся процессы ЖЦ – В соответствии с положениями стандарта
- СТБ ИСО/МЭК 12207–2003?
- Назовите основные стороны, участвующие в ЖЦ ПС и систем.
- Перечислите и определите назначение процессов ЖЦ в каждой группе, регламентированной стандартом СТБ ИСО/МЭК 12207–2003.
- Перечислите работы процесса разработки, регламентированные стандартом СТБ ИСО/МЭК 12207–2003, и опишите их содержание.
- Назовите системные и программные работы процесса регламентированного стандартом СТБ ИСО/МЭК 12207–2003.
- Назовите базовые стратегии разработки ПС и систем.
- Охарактеризуйте сущность каскадной стратегии разработки ПС и систем, перечислите достоинства, недостатки и области применения данной стратегии.
- Охарактеризуйте сущность инкрементной стратегии разработки ПС и систем, перечислите достоинства, недостатки и области применения данной стратегии.
- Охарактеризуйте сущность эволюционной стратегии разработки ПС и систем, перечислите достоинства, недостатки и области применения данной стратегии.
- Дайте сравнительную характеристику каскадной, инкрементной и эволюционной стратегий разработки ПС и систем.
- Назовите общие черты каскадных моделей жизненного цикла. В чем заключаются ее преимущества и недостатки по сравнению с классической каскадной моделью?

- Назовите основные черты RAD-моделей ЖЦ. Перечислите основные достоинства, недостатки и области использования RAD-моделей.
- Назовите общие черты инкрементных моделей ЖЦ.
- Назовите общие черты эволюционных моделей ЖЦ.
- В чем заключаются ее особенности, достоинства и недостатки по сравнению с базовой спиральной моделью ЖЦ Боэма?
- Охарактеризуйте схему классификации проектов по разработке ПС и систем предложенную Институтом качества программного обеспечения SQI для выбора модели ЖЦ.
- Перечислите категории критериев, положенных в основу схемы классификации проектов Института SQI.
- Перечислите шаги процедуры выбора модели ЖЦ ПС и систем, предложенной Институтом SQI.
- Назовите критерии категории характеристик требований к проекту.
- Назовите критерии категории характеристик команды разработчиков.
- Назовите критерии категории характеристик пользователей (заказчиков).
- Назовите критерии категории характеристик типов проектов и рисков.

7. Учебно-методическое и информационное обеспечение дисциплины

7.1.Список источников и литературы

- Литература:
  - Основная:
    1. Москвитин А.А. Решение задач на компьютерах: учебное пособие. - М.; Берлин: Директ-Медиа, 2015 - Ч. II. Разработка программных средств. - 427 с. Режим доступа: http://biblioclub.ru/index.php?page=book&id=273667
    2. Влацкая И.В. Проектирование и реализация прикладного программного обеспечения: учебное пособие / И.В. Влацкая, Н.А. Заельская, Н.С. Надточий. - Оренбург: ОГУ, 2015 — 119 с. Режим доступа: http://biblioclub.ru/index.php?page=book&id=439107
  - Дополнительная:
    3. Гультяев А.К. Macromedia Authorware 6.0. Разработка мультимедийных учебныхкурсов / А. К. Гультяев. - СПб.: КОРОНА принт, 2007 - 400 с.
    4. Котляров В.П. Основы тестирования программного обеспечения: учеб.пособ.- М.: ИНТУИТ, 2006
    5. Скопин И.Н. Основы менеджмента программных проектов. Курс лекций: учеб. пособ.- М.: ИНТУИТ, 2004

7.2. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимый для освоения дисциплины (модулей)

1. http://study.utmn.ru– Портал доступа к электронным образовательным ресурсам ТюмГУ;
2. http://biblioclub.ruбиблиотека онлайн.
3. Электронные ресурсы Kyrlibnet http://arch.kyrlibnet.kg

8. Перечень учебно-методического обеспечения для самостоятельной работы обучающихся.

8.1. Планы практических (семинарских) и лабораторных занятий. Методические указания по организации и проведению

- Тема 1. Этапы разработки программного обеспечения при структурном подходе к программированию. Стадия «Техническое задание».
  - Цель занятия: изучение этапов разработки программного обеспечения при структурном подходе к программированию. Стадия «Техническое задание».
  - Форма проведения –разработка программного обеспечения
  - Содержание занятия:
    - Этапы разработки программного обеспечения при структурном подходе к программированию. Стадия «Техническое задание».
  - Литература:[1, 2]
  - Материально-техническое обеспечение занятия: ЭВМ
- Тема 2. Структурный подход к программированию. Стадия «Эскизный проект».
  - Цель занятия: изучение структурного подхода к программированию.
  - Форма проведения –разработка программного обеспечения
  - Содержание занятия:
    - Структурный подход к программированию.
  - Литература:[1, 2]
  - Материально-техническое обеспечение занятия: ЭВМ
- Тема3. Структурный подход к программированию. Стадия «Технический проект».
  - Цель занятия: изучение структурного подхода к программированию.
  - Форма проведения –разработка программного обеспечения
  - Содержание занятия:
    - Структурный подход к программированию
  - Литература:[1, 2]
  - Материально-техническое обеспечение занятия: ЭВМ
- Тема4. Этапы разработки программного обеспечения. Стадия «Реализация».
  - Цель занятия: изучение этапов разработки программного обеспечения на стадии Реализации.
  - Форма проведения –разработка программного обеспечения
  - Содержание занятия:
    - Этапы разработки программного обеспечения.
  - Литература:[1, 2, 3]
  - Материально-техническое обеспечение занятия: ЭВМ
- Тема5. Тестирование программ методами «белого ящика».
  - Цель занятия: изучение тестирования программ методами «белого ящика».
  - Форма проведения –разработка программного обеспечения
  - Содержание занятия:
    - Тестирование программ методами «белого ящика».
  - Литература:[1, 3, 4]
  - Материально-техническое обеспечение занятия: ЭВМ
- Тема6. Использование технологий OLE, COM и ActiveX.
  - Цель занятия: изучение технологий OLE, COM и ActiveX.
  - Форма проведения –разработка программного обеспечения
  - Содержание занятия:
    - Использование технологий OLE, COM и ActiveX.
  - Литература:[1, 3, 4]
  - Материально-техническое обеспечение занятия: ЭВМ
- Тема7. Создание сетевых приложений на C++ с использованием Windows Sockets API.
  - Цель занятия: изучение сетевых приложений на C++ с использованием Windows Sockets API.

- Форма проведения –разработка программного обеспечения
- Содержание занятия:
  - Создание сетевых приложений на C++ с использованием Windows Sockets API.
- Литература:[1, 3, 4]
- Материально-техническое обеспечение занятия: ЭВМ

- Тема8. Проектирование программной системы при объектном подходе к программированию.
  - Цель занятия: изучение программной системы при объектном подходе к программированию.
  - Форма проведения – разработка программного обеспечения
  - Содержание занятия:
    - Проектирование программной системы при объектном подходе к программированию.
  - Литература:[1, 2, 4, 5]
  - Материально-техническое обеспечение занятия: ЭВМ
- Тема9. Динамические структуры данных.
  - Цель занятия: изучение динамических структур данных.
  - Форма проведения –разработка программного обеспечения
  - Содержание занятия:
    - Динамические структуры данных.
  - Литература:[1, 2, 4, 5]
  - Материально-техническое обеспечение занятия: ЭВМ
- Тема10. Объектно-ориентированное программирование (ООП).
  - Цель занятия: изучение объектно-ориентированного программирования.
  - Форма проведения –разработка программного обеспечения
  - Содержание занятия:
    - Объектно-ориентированное программирование (ООП).
  - Литература:[2, 4, 5]
  - Материально-техническое обеспечение занятия: ЭВМ

8.2. Методические указания для обучающихся по освоению дисциплины (модулей)

Вид работы	Содержание (перечень вопросов)	Трудоемкость самостоятельной работы (в часах)	Рекомендации
Модуль№1.			
Конспектирование материала на лекционных занятиях Выполнение заданий лабораторных работ Выполнение тестовых и контрольных работ. Работа с учебной литературой. Написание программы	Стратегии разработки программных средств и систем и реализующие их модели жизненного цикла	5	[1, 2]
Конспектирование материала на	Выбор модели жизненного цикла для	5	[1, 2]

лекционных занятиях Выполнение заданий лабораторных работ Выполнение тестовых и контрольных работ. Работа с учебной литературой. Написание программы	конкретного проекта		
Итого		10	
Модуль№2.			
Конспектирование материала на лекционных занятиях Выполнение заданий лабораторных работ Выполнение тестовых и контрольных работ. Работа с учебной литературой. Написание программы	Классические методологии разработки программных средств	8	[1, 2, 3]
Конспектирование материала на лекционных занятиях Выполнение заданий лабораторных работ Выполнение тестовых и контрольных работ. Работа с учебной литературой. Написание программы	Case-технологии технологии структурного анализа и проектирования программных средств	8	[1, 2, 4]
Итого		16	
Модуль №3.			
Конспектирование материала на лекционных занятиях Выполнение заданий лабораторных работ Выполнение тестовых и контрольных работ. Работа с учебной литературой. Написание программы	Методология объектно-технологии ориентированного анализа и проектирования сложных систем	8	[1, 2, 5]
Конспектирование материала на	Инструментальные средства разработки	8	[1, 2, 5]

лекционных занятиях Выполнение заданий лабораторных работ Выполнение тестовых и контрольных работ. Работа с учебной литературой. Написание программы	программного обеспечения		
Итого		16	
Итого по дисциплине		42	

8.3. Методические рекомендации по подготовке письменных работ

Разработанное программное обеспечение должно быть предоставлено в скомпилированном виде, а так же в виде текстового файла, содержащего исходный код программы.

8.4. Иные материалы

Не предусмотрено.

9. Материально-техническое обеспечение дисциплины

Для изучения дисциплины, необходимо следующее оборудование: ЭВМ, проектор.

Требования к аудитории: компьютерный класс, имеющий ЭВМ в количестве идентичном количеству обучающихся, ЭВМ для преподавателя с подключенным проектором, наличие доски средств для отображения/удаления информации на доске (мел/ветошь, маркер/губка).

10. Глоссарий

Не предусмотрено.

11. Приложения

Приложение 1

Тема №1: Введение. Требования к современным технологиям.

Программное средство (ПС) – это программа или логически связанная совокупность программ на носителях данных, снабженная программной документацией.

Программная документация позволяет понять, какие функции выполняет та или иная программа, как подготовить исходные данные и запустить требуемую программу в процесс ее выполнения, а также: что означают получаемые результаты (или каков эффект выполнения этой программы). Будем считать, что в ПС имеется ошибка, если оно не выполняет функции, описанные в документации по ее применению.

Надежность ПС - это его способность безотказно выполнять определенные функции при заданных условиях в течение заданного периода времени с достаточно большой вероятностью.

Технология программирования – это система научно-обоснованных принципов, методов и средств, обеспечивающих создание и развитие ПС, в течении всего жизненного периода (жизненного цикла) программного средства.

Состав технологии программирования:

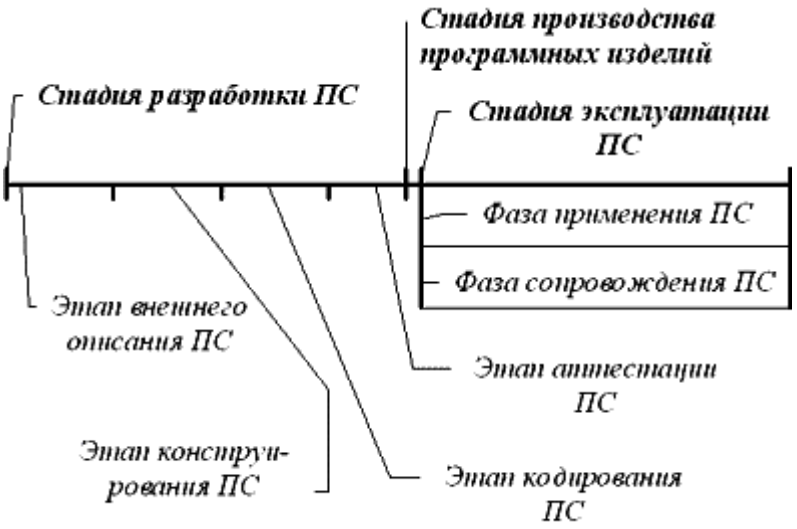
1. принципы и методы организации труда при разработке, эксплуатации ПС;
2. принципы и методы разработки самих программ;

3. средства инструментальной поддержки и автоматизации, позволяющие унифицировать разработку ПС.

Тема №2: Общие принципы разработки программных средств.

Жизненный цикл ПС

Жизненный цикл ПС - период его разработки и эксплуатации (использования), начиная от момента возникновения замысла ПС и заканчивая прекращением всех видов его использования.



Внешнее описание ПС является описанием его поведения с точки зрения внешнего по отношению к нему наблюдателю с фиксацией требований относительно его качества. Внешнее описание ПС начинается с определения требований к ПС со стороны пользователей (заказчика).
Конструирование ПС охватывает процессы: разработку архитектуры ПС, разработку структур программ ПС и их детальную спецификацию.
Кодирование: создание текстов программ на языке программирования, их отладка и тестирование ПС.
На этапе *аттестации* ПС производится оценка качества ПС, после успешного завершения которого, разработка ПС считается законченной.
Программное изделие (ПИ) - экземпляр или копия, снятая с разработанного ПС. *Изготовление* ПИ - это процесс генерации и/или воспроизведения (снятия копии) программ и программных документов ПС с целью их поставки пользователю для применения по назначению.
Производство ПИ - это совокупность работ по обеспечению изготовления требуемого количества ПИ в установленные сроки. *Стадия производства* ПС в жизненном цикле ПС является, по-существу, вырожденной (не существенной), так как представляет рутинную работу, которая может быть выполнена автоматически и без ошибок.
Стадия *эксплуатации* ПС охватывает процессы хранения, внедрения и сопровождения ПС, а также транспортировки и применения ПИ по своему назначению. Она состоит из двух параллельно проходящих фаз: фазы применения ПС и фазы сопровождения ПС.
Применение ПС - это использование ПС для решения практических задач на компьютере путем выполнения ее программ.
Сопровождение ПС - это процесс сбора информации о его качестве в эксплуатации, устранения обнаруженных в нем ошибок, его доработки и модификации, а также извещения пользователей о внесенных в него изменениях.

Понятие качества ПС.

Качество ПС - это совокупность его черт и характеристик, которые влияют на его способность удовлетворять заданные потребности пользователей.

Качество ПС является удовлетворительным, когда оно обладает указанными свойствами в такой степени, чтобы гарантировать успешное его использование.
Критерии качества ПС:

- функциональность,
- надежность,
- легкость применения,
- эффективность,
- сопровождаемость,
- мобильность.

Функциональность - это способность ПС выполнять набор функций, удовлетворяющих заданным или подразумеваемым потребностям пользователей. Набор указанных функций определяется во внешнем описании ПС.
Легкость применения - это характеристики ПС, которые позволяют минимизировать усилия пользователя по подготовке исходных данных, применению ПС и оценке полученных результатов, а также вызывать положительные эмоции определенного или подразумеваемого пользователя.
Эффективность - это отношение уровня услуг, предоставляемых ПС пользователю при заданных условиях, к объему используемых ресурсов.
Сопровождаемость - это характеристики ПС, которые позволяют минимизировать усилия по внесению изменений для устранения в нем ошибок и по его модификации в соответствии с изменяющимися потребностями пользователей.
Мобильность - это способность ПС быть перенесенным из одной среды (окружения) в другую, в частности, с одной ЭВМ на другую.
Обязательные критерии качества: функциональность и надежность.

Тема №3: Разработка структуры программы и модульное программирование

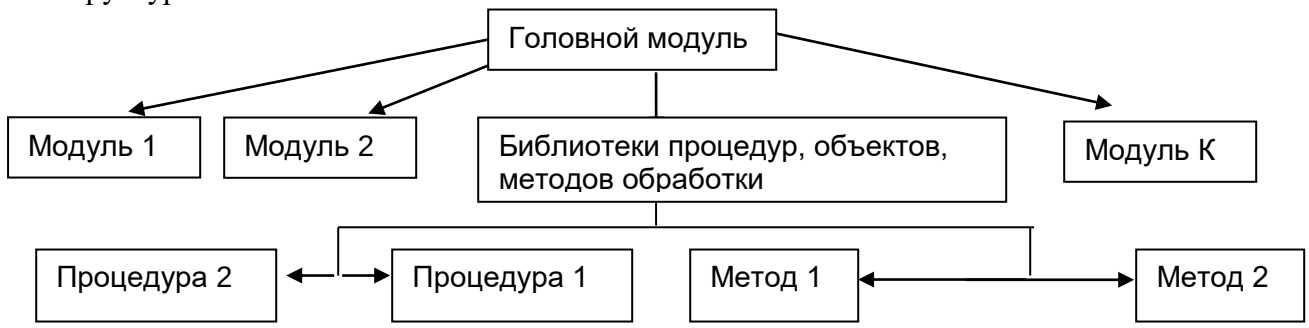
Приступая к разработке каждой программы ПС, следует иметь ввиду, что она является большой системой, поэтому нужно принимать меры для ее упрощения. Для этого программу разрабатывают по частям, которые называются программными модулями. Такой метод программирования называют модульным программированием.
Модуль – это самостоятельная часть программы, имеющая определенное значение и обеспечивающая заданные функции обработки автономно от других программных модулей. Каждый программный модуль программируется, компилируется и отлаживается отдельно от других модулей программы, и тем самым физически разделен с другими программными модулями.
Каждый программный модуль может включаться в состав разных программ, если выполнены условия его использования, описанные в документации по ее использованию. Т.е. хорошо продуманный модуль позволяет избежать дублирования в программировании.
Любое ПС имеет свою структуру, которая разрабатывается для удобства:

1. разработки
2. программирования
3. отладки
4. внесения изменений

Также структуризация ПС позволяет:

1. распределить работы по исполнителям, обеспечив их загрузку и требуемые сроки разработки
2. построить календарные графики проектных работ и осуществлять их координацию в процессе создания ПИ.
3. контролировать трудозатраты и стоимость проектных работ.

При создании ПС выделяются многократно используемые модули, их типизируют и унифицируют, за счет чего сокращаются сроки и трудозатраты на разработку ПС в целом. Некоторые ПС используют готовые модули из библиотек стандартных процедур, функций, объектов и методов обработки данных.
Типовая структура ПС:



Типы модулей:

Наименование модуля	**Описание**
Головной модуль	Управляет запуском ПС (он в единственном числе)
Управляющий модуль	Обеспечивает вызов других модулей, задает последовательность вызова на выполнение очередного модуля
Рабочие модули	Выполняют функции обработки
Сервисные модули	Осуществляют обслуживающие функции

Свойства модуля:

1. один вход и один выход – на входе программный модуль получает определенный набор исходных данных, выполняет обработку данных и возвращает результат
2. функциональная завершенность – модуль выполняет перечень операций для реализации каждой отдельной функции в полном составе.
3. логическая независимость – результат работы модуля зависит только от исходных данных, и не зависит от работы других модулей.
4. слабые информационные связи с другими программными модулями – обмен информации между модулями должен быть по возможности минимизирован.
5. обозримый по размеру и сложности.

Каждый модуль состоит из спецификации и тела модуля.
Спецификация – правила использования модуля.
Тело – способ реализации процесса обработки.
Принцип модульного программирования ПС:

1. Определение состава и подчиненность функций
2. определение набора программных модулей, реализующих эти функции

При составлении алгоритма необходимо учитывать:

1. каждый модуль вызывается на выполнение вышестоящим модулем и закончив работу, возвращает управление вызвавшему модулю.

Методы разработки структуры программы.

В качестве модульной структуры программы принято использовать древовидную структуру, включая деревья со сросшимися ветвями. В узлах дерева размещаются модули, а стрелки

показывают подчиненность модулей (т.е. в тексте модуля, из которого она исходит, имеется ссылка на модуль, в который она входит).
Модульная структура программы должна включать и совокупность спецификаций модулей, образующих эту программу. *Спецификация* программного модуля содержит:

1. синтаксическую спецификацию его входов, позволяющую построить обращение к нему на используемом языке программирования
2. функциональную спецификацию модуля (описание всех функций, выполняемых этим модулем).

Существуют разные методы разработки структуры программы. Обычно используют 2 метода:

1. Метод восходящей разработки
2. метод нисходящей разработки

Метод восходящей разработки. Сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с модулей самого нижнего уровня, в таком порядке, чтобы для каждого программируемого модуля были уже запрограммированы все модули, к которым он может обращаться. После того, как все модули программы запрограммированы, производится их поочередное тестирование и отладка в принципе в таком же (восходящем) порядке, в каком велось их программирование.
На первый взгляд такой порядок разработки программы кажется вполне естественным: каждый модуль при программировании выражается через уже запрограммированные непосредственно подчиненные модули, а при тестировании использует уже отлаженные модули. Не рекомендуется, т.к.:

1. для программирования какого-либо модуля совсем не требуется текстов используемых им модулей - для этого достаточно, чтобы каждый используемый модуль был лишь специфицирован (в объеме, позволяющем построить правильное обращение к нему), а для тестирования его возможно используемые модули заменять их имитаторами (заглушками).
2. каждая программа в какой-то степени подчиняется некоторым внутренним для нее, но глобальным для ее модулей соображениям (принципам реализации, предположениям, структурам данных и т.п.), что определяет ее концептуальную целостность и формируется в процессе ее разработки. При восходящей разработке эта глобальная информация для модулей нижних уровней еще не ясна в полном объеме, поэтому очень часто приходится их перепрограммировать.
3. при восходящем тестировании для каждого модуля (кроме головного) приходится создавать ведущую программу (модуль), которая должна подготовить для тестируемого модуля необходимое состояние информационной среды и произвести требуемое обращение к нему. Это приводит к большому объему "отладочного" программирования и в то же время не дает никакой гарантии, что тестирование модулей производилось именно в тех условиях, в которых они будут выполняться в рабочей программе.

Метод *нисходящей разработки*.
Сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с модуля самого верхнего уровня (головного), переходя к программированию какого-либо другого модуля только в том случае, если уже запрограммирован модуль, который к нему обращается. После того, как все модули программы запрограммированы, производится их поочередное тестирование и отладка в таком же (нисходящем) порядке.
Положительные стороны

1. При таком порядке разработки программы вся необходимая глобальная информация формируется своевременно, т.е. ликвидируется весьма неприятный источник просчетов при программировании модулей.

2. Существенно облегчается и тестирование модулей, производимое при нисходящем тестировании программы. Первым тестируется головной модуль программы, который представляет всю тестируемую программу и поэтому тестируется при "естественном" состоянии информационной среды, при котором начинает выполняться эта программа. При этом все модули, к которым может обращаться головной, заменяются на их имитаторы. И*митатор модуля* - простой программный фрагмент, сигнализирующий, о самом факте обращения к имитируемому модулю с необходимой для правильной работы программы обработкой значений его входных параметров и с выдачей, если это необходимо, заранее запасенного подходящего результата. После завершения тестирования и отладки головного и любого последующего модуля производится переход к тестированию одного из модулей, которые в данный момент представлены имитаторами, если таковые имеются. Для этого имитатор выбранного для тестирования модуля заменяется на сам этот модуль и добавляются имитаторы тех модулей, к которым может обращаться выбранный для тестирования модуль. При этом каждый такой модуль будет тестироваться при "естественных" состояниях информационной среды, возникающих к моменту обращения к этому модулю при выполнении тестируемой программы. Таким образом большой объем "отладочного" программирования заменяется программированием достаточно простых имитаторов используемых в программе модулей. Кроме того, имитаторы удобно использовать для подыгрывания процессу подбора тестов путем задания нужных результатов, выдаваемых имитаторами.

Рассмотренные методы называются *классическими.* В них модульная древовидная структура программы должна разрабатываться до начала программирования модулей. Однако такой подход вызывает ряд возражений: маловероятно, чтобы до программирования модулей можно было разработать структуру программы достаточно точно и содержательно.

При конструктивном и архитектурном подходах к разработке программ модульная структура формируется в процессе программирования модулей.

Конструктивный подход (Модификация нисходящей разработки**)**

Модульная древовидная структура программы формируется в процессе программирования модуля. Сначала программируется головной модуль, исходя из спецификации программы в целом, причем спецификация программы является одновременно и спецификацией ее головного модуля, так как последний полностью берет на себя ответственность за выполнение функций программы. В процессе программирования головного модуля, в случае, если эта программа достаточно большая, выделяются внутренние функции, в терминах которых программируется головной модуль. Это означает, что для каждой выделяемой функции создается спецификация реализующего ее фрагмента программы, который в дальнейшем может быть представлен некоторым поддеревом модулей. Важно заметить, что здесь также ответственность за выполнение выделенной функции берет головной (может быть, и единственный) модуль этого поддерева, так что спецификация выделенной функции является одновременно и спецификацией головного модуля этого поддерева. В головном модуле программы для обращения к выделенной функции строится обращение к головному модулю указанного поддерева в соответствии с созданной его спецификацией. Таким образом, на первом шаге разработки программы (при программировании ее головного модуля) формируется верхняя начальная часть дерева, например, такая, которая показана на рис. 7.1.

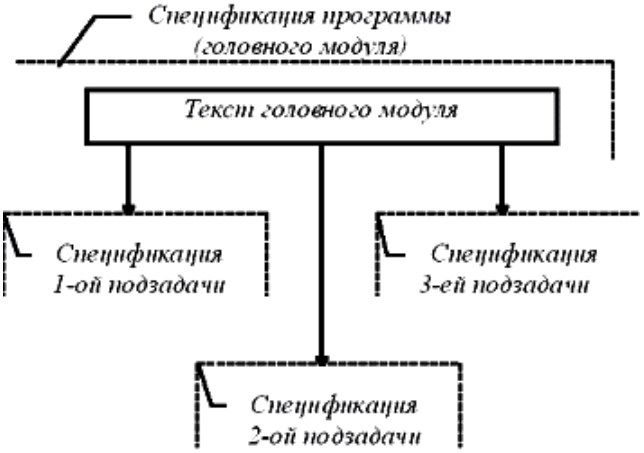


Рис. 7.1. Первый шаг формирования модульной структуры программы при конструктивном подходе.

Аналогичные действия производятся при программировании любого другого модуля, который выбирается из текущего состояния дерева программы из числа специфицированных, но пока еще не запрограммированных модулей. В результате этого производится очередное доформирование дерева программы, например, такое, которое показано на рис. 7.2.

Архитектурный подход (модификация восходящей разработки)

Модульная структура программы формируется в процессе программирования модуля. Но при этом ставится другая цель разработки: повышение уровня используемого языка программирования, а не разработка конкретной программы. Это означает, что для заданной предметной области выделяются типичные функции, каждая из которых может использоваться при решении разных задач в этой области, и специфицируются, а затем и программируются отдельные программные модули, выполняющие эти функции. Так как процесс выделения таких функций связан с накоплением и обобщением опыта решения задач в заданной предметной области, то обычно сначала выделяются и реализуются отдельными модулями более простые функции, а затем постепенно появляются модули, использующие ранее выделенные функции. Такой набор модулей создается в расчете на то, что при разработке той или иной программы заданной предметной области в рамках конструктивного подхода могут оказаться приемлемыми некоторые из этих модулей.

Это позволяет сократить трудозатраты на разработку конкретной программы путем подключения к ней заранее заготовленных и проверенных на практике модульных структур нижнего уровня. Так как такие структуры могут многократно использоваться в разных конкретных программах, то архитектурный подход может рассматриваться как путь борьбы с дублированием в программировании. В связи с этим программные модули, создаваемые в рамках архитектурного подхода, обычно параметризуются для того, чтобы усилить применимость таких модулей путем настройки их на параметры.

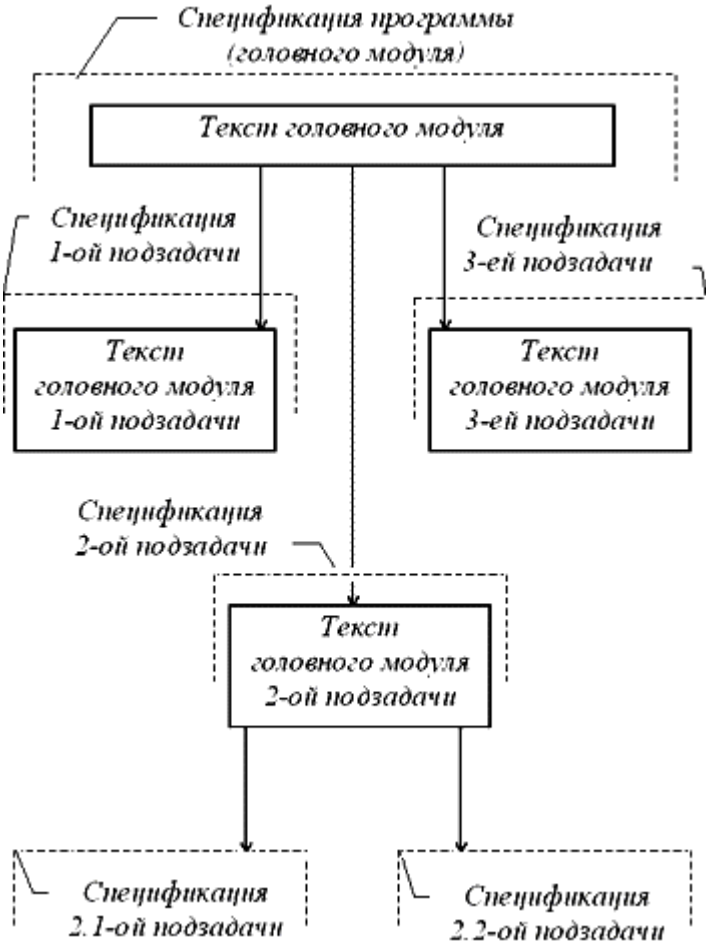


Рис. 7.2. Второй шаг формирования модульной структуры программы при конструктивном подходе.

Метод нисходящей реализации. Каждый запрограммированный модуль начинают сразу же тестировать до перехода к программированию другого модуля.

Тема №4: Разработка программного модуля. Структурное программирование.

Порядок разработки программного модуля.

1. **изучение и проверка спецификации модуля, выбор языка программирования;**

(т.е. разработчик изучая спецификацию выясняет понятна она ему или нет, достаточно ли полно она описывает модуль; затем он выбирает язык программирования, на котором будет написан модуль, хотя язык программирования может быть единым для всего ПС)

2. **выбор алгоритма и структуры данных** *(здесь выясняется не известны ли какие-либо алгоритмы для решения поставленной задачи и если есть, то воспользоваться им)*
3. **программирование модуля** *(написание кода программы)*
4. **шлифовка текста модуля** *(редактирование имеющихся комментариев, добавление дополнительных комментариев, для того чтобы обеспечить требуемое качество)*
5. **проверка модуля** *(проверяется логика работы модуля, отлаживается его работа)*

Применяются следующие методы контроля программного модуля:

- статическая проверка текста модуля *(текст прочитывается с начала до конца с целью найти ошибки в модуле. Обычно для такой проверки привлекают, кроме разработчика модуля, еще одного или даже нескольких программистов. Рекомендуется ошибки, обнаруживаемые при такой проверке исправлять не сразу, а по завершению чтения текста модуля)*

- сквозное прослеживание *(вручную прокручивается выполнение модуля (оператор за оператором в той последовательности, какая вытекает из логики работы модуля) на некотором наборе тестов)*

6. **компиляция модуля.**

Структурное программирование.

На сегодняшний день самой популярной методикой программирования является структурное программирование «сверху-вниз».

Структурное программирование – это процесс пошагового разбиения алгоритма на все более мелкие части, с целью получить такие элементы, для которых можно легко написать конкретные предписания.

Два принципа структурного программирования:

1. последовательная детализация «сверху – вниз»
2. ограниченность базового набора структур для построения алгоритмов любой степени сложности

Требования структурного программирования:

1. программа должна составлятся мелкими шагами, таким образом сложная задача разбивается на достаточно простые, легко воспринимаемые части
2. логика программы должна опираться на минимальное число достаточно базовых управляющих структур (линейные, разветвляющиеся и циклические структуры)

Основные свойства и достоинства структурного программирования:

1. уменьшение сложности программ
2. возможность демонстрации правильности программ на различных этапах решения задачи
3. наглядность программ
4. простота модификации (внесения изменения) программ.

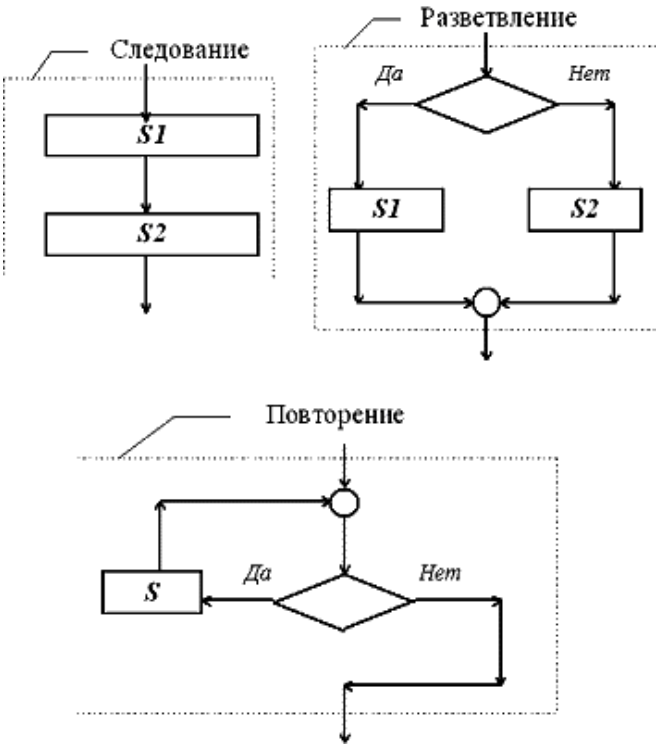
Современные средства программирования должны обеспечивать максимальную защиту от возможных ошибок разработчика.

Тут можно провести аналогию с развитием методов управления автотранспортом. Сначала безопасность обеспечивалась за счет разработки правил движения. Затем появилась система разметки дорог и регулирования перекрестков. И, наконец, стали строиться транспортные развязки, которые в принципе предотвращают пересечение потоков машин и пешеходов. Впрочем, используемые средства должны определяться характером решаемой задачи: для проселочной дороги вполне достаточно соблюдения простого правила - "смотри под ноги и по сторонам".

Основная идея структурного программирования: программа должна представлять собой множество блоков, объединенных в виде иерархической древовидной структуры, каждый из которых имеет один вход и один выход.

Любую программу можно построить, используя лишь три основных типа блоков:

1. функциональный блок - отдельный линейный оператор или их последовательность;
2. **блок разветвления** - If <условие> Then...Else.
3. обобщенный цикл - конструкция типа While <условие> Do <операторы> (проверка в начале цикла!);



Существенно, что каждая из этих конструкций имеет по управлению только один вход и один выход. Тем самым, и обобщенный оператор имеет только один вход и один выход.

Структурное программирование иногда называют еще "программированием без GO TO". Однако дело здесь не в операторе GO TO, а в его беспорядочном использовании. Очень часто при воплощении структурного программирования на некоторых языках программирования оператор перехода (GO TO) используется для реализации структурных конструкций, не снижая основных достоинств структурного программирования. Запутывают программу как раз "неструктурные" операторы перехода, особенно переход на оператор, расположенный в тексте модуля выше (раньше) выполняемого оператора перехода. Тем не менее, попытка избежать оператора перехода в некоторых простых случаях может привести к слишком громоздким структурированным программам, что не улучшает их ясность и содержит опасность появления в тексте модуля дополнительных ошибок. Поэтому можно рекомендовать избегать употребления оператора перехода всюду, где это возможно, но не ценой ясности программы.
К полезным случаям использования оператора перехода можно отнести выход из цикла или процедуры по особому условию, "досрочно" прекращающего работу данного цикла или данной процедуры, т.е. завершающего работу некоторой структурной единицы (обобщенного оператора) и тем самым лишь локально нарушающего структурированность программы. Большие трудности (и усложнение структуры) вызывает структурная реализация реакции на возникающие исключительные (часто ошибочные) ситуации, так как при этом требуется не только осуществить досрочный выход из структурной единицы, но и произвести необходимую обработку этой ситуации (например, выдачу подходящей диагностической информации). Обработчик исключительной ситуации может находиться на любом уровне структуры программы, а обращение к нему может производиться с разных нижних уровней. Вполне приемлемой с технологической точки зрения является следующая "неструктурная" реализация реакции на исключительные ситуации. Обработчики исключительных ситуаций помещаются в конце той или иной структурной единицы и каждый такой обработчик программируется таким

образом, что после окончания своей работы производит выход из той структурной единицы, в конце которой он помещен. Обращение к такому обработчику производится оператором перехода из данной структурной единицы (включая любую вложенную в нее структурную единицу).

Вообще говоря, главное в структурном программировании - грамотное составление правильной логической схемы программы, реализация которой языковыми средствами - дело вторичное.

Тема №5: Стиль программирования. Выбор языка программирования.

Программы должны составляться так, чтобы их могли прочитать в первую очередь люди, а не машины. Помните: программы читаются людьми, т.е. удобочитаемость программ - существенна, поэтому имеет смысл говорить о стандарте стиля программ.

Легко читаемая программа создает впечатление, что ее автор хорошо знал что делал.

Программа должна передавать логику и структуру алгоритма настолько, насколько это возможно.

Один из аргументов против стандартизации стиля программирования звучит так: стиль программирования – это дело личного мнения и вкуса, поэтому не следует вводить на него каких – либо ограничений – этот аргумент, в сущности, утверждает, что хаос лучше порядка.

Правило стандартизации стиля: если существует более одного способа сделать что – либо и выбор произвольный, остановись на одном способе и всегда его придерживайся.

Легче избежать путаницы, придерживаясь стандартных приемов, хороший набор стандартных приемов позволяет сконцентрировать внимание на главной задаче.

Невозможно выработать единого промышленного стандарта для всех программ, но в пределах единого проекта, задачи, разработки – это нормальное явление.

Комментарии:

Желательность комментариев очевидна. Программы с пояснительными комментариями значительно легче отслеживать, читая чужую программу, программисты не мало времени отслеживают логику чужой программы, т.е. не комментируемая программа – свидетельство делитанского похода.

Когда следует писать комментарий? В процессе написания программы. Бессмысленно вставлять комментарии, после того, как программа завершена.

Делайте комментариев больше, чем это кажется необходимым. Т.к. цель комментариев – облегчить понимание программы. Если вы затрудняетесь в написание комментария – значит “Вы не ведаете, что творите! ”

Три типа комментариев:

1. вводные
2. оглавление
3. пояснительные.

Вводные комментарии:

Каждая программа, подпрограмма или процедура, должна начинаться с комментариев, поясняющих, что она делает. Минимальная информация, содержащаяся в вводных комментариях должна включать следующие пункты:

- ✓ Назначение программы.
- ✓ Указания по вызову программы и ее использованию.
- ✓ Список и назначение основных переменных или массивов.
- ✓ Указание по вводу – выводу. Список всех файлов.
- ✓ Список используемых подпрограмм.
- ✓ Назначение применяемых математических методов.
- ✓ (иногда) Сведения о времени выполнения программы.
- ✓ Требуемый объем памяти.
- ✓ Специальные указания оператору.

- ✓ Сведения об авторе.
- ✓ Дату написания программы.

Пояснительный комментарий:

Пояснениями нужно сопровождать те части программы, которые трудно понять без комментариев. Сопровождайте комментариями те действия, которые, с вашей точки зрения, могут быть не совсем понятны другому. Комментарии не должны объяснять синтаксис языка программирования, а должны раскрывать логику программы или цель конкретного действия. Если логика программы понятна только на основании прочтения комментариев, это качественные комментарии.

Расположение комментариев.

Комментарии легче читаются, если они отделяются пустыми строками (до и после комментариев). Дополнительный метод выделения комментариев – заключение их в прямоугольник из специальных символов (звездочки, слеши, скобки и т.д.). Располагаются комментарии т.о., чтобы это не делало программу менее наглядной. Неправильные комментарии хуже, чем их отсутствие. Делайте пробелы для улучшения читаемости программ.

Выбор имен переменных, файлов.

Имена переменных должны наилучшим образом определять те величины, которые они представляют. Если ограничения на длину имени отсутствуют, используйте имена постольку длинные, поскольку это нужно, но не длиннее, чем необходимо.

- ✓ Имена переменных не должны совпадать со служебными словами.
- ✓ Избегайте схожих по виду имен и подобных по написанию символов.
- ✓ Различие имен должно быть всегда явно ощутимым.
- ✓ Когда имена содержат лишнюю информацию это тоже плохо, т.к. программа расширяется, становится громоздкой.
- ✓ При выборе имен переменных старайтесь установить, что обозначает эта переменная на естественном языке, и выбирайте наиболее подходящее слово.

Имена файлов:

- ✓ Использование одних и тех же имен для одинаковых файлов в различных программах, обеспечивает быструю идентификацию их (узнаваемость).
- ✓ Иногда использование префикса, помогает определять, какие поля связаны логически.
- ✓ При выборе имен записей используйте имена, ориентированные на содержание записи, а не на конкретное задание.

Отступы:

При записи операторов для указания связи между ними делают одинаковый отступ от начала строки. И хотя отступы не оказывают влияния на логику программы, существенно улучшают ее читаемость. Циклы – типичный случай для использования отступов.

- ✓ Для выявления структуры программы – используйте отступы.
- ✓ Если операторы занимают несколько строк, то строки, начиная со второй, должны иметь отступ чтобы находиться справа от знака равенства.

Выбор и обоснование языка программирования.

Программа должна быть существенно *коммуникативна*. Успешная коммуникация достигается благодаря *языку*, понятному программистам и процессору. Команды, выполняемые центральным процессором компьютерной системы, должны быть выражены в машинном коде. Каждая команда при этом выступает в виде последовательности нулей и единиц. Программист, конечно, может затрачивая огромные усилия понимать и составлять программы на машинном языке. Но для создания больших и надежных программ машинный код совершенно неприемлем. *Трансляция* - один из способов преодолеть “языковый” барьер. Если воспользоваться языком, понятным программисту, а потом обеспечить перевод с него в машинный код, проблема будет решена. Может показаться, что для этой цели лучше выбрать естественный язык, например,

английский, но на самом деле удобнее сконструировать особый язык. На сегодняшний день создано много таких языков, которые названы - *языки программирования высокого уровня*. Хорошо сконструированные языки высокого уровня обладают целым рядом достоинств, основанных на следующих фактах:

а) *Средства, предоставляемые языком, позволяют удовлетворить потребности конкретной прикладной области*. Например, один язык может быть разработан для научных, преимущественно численных расчетов, другой для коммерческих приложений, обрабатывающих много нечисловой информации, третий будет применяться в других прикладных областях.

б) *В визуальном отношении программа должна быть такой, чтобы ее легко было читать, и чтобы была ясна ее структура*. Проектирование и написание больших программ - сложная интеллектуальная задача, и для ее успешного решения программист должен предельно сосредоточиться на том, что он делает и ясно представлять себе задачу. Это очень полезно не только тому, кто пишет программу, но и особенно программисту, которому поручено ее усовершенствовать.

в) *В язык могут быть (и даже должны быть) встроены средства, помогающие выявлять и предупреждать ошибки*. Учитывая важность того, чтобы законченная программа была правильной, и принимая во внимание естественную подверженность программиста ошибкам, такие средства следует признать главным достоинством языков высокого уровня.

Отношение и эффективность.

Существует три типа программ и отношение к эффективности этих программ должно быть различным.

Первый тип – это часто используемые программы: операционные системы, компиляторы, трансляторы, прикладные программы. Для таких программ эффективность является первостепенной задачей, вследствие их частого использования и специфического выполнения.

Второй тип – производственные программы, написанные профессиональными программистами, эксплуатируемые долгое время. Хотя эффективность таких программ существенна, обычно большее внимание уделяется их эксплуатационным характеристикам.

Третий тип – программы, написанные не программистами, а научными сотрудниками это программы должны уместиться в заданном объёме памяти и выполняется за конкретное время. Время для них важней всего.

Следовательно: ещё до написания программы необходимо установить, на сколько эффективной она должна быть.

Очевидно, что следует модифицировать только те программы, которые многократно повторяются (используются). «Не стоит экономить на спичках». Многие методы, делающие программу эффективной могут существенно влиять на ее удобочитаемость, а удобочитаемость программы более существенно, чем ее эффективность. В каком случае эффективности отдают предпочтение в ущерб удобочитаемости: если программа должна быть выточена в библиотеку часто используемых программ функции, программа не помещается в памяти, она важна, но слишком долго выполняется.

Оптимизация программ.

Если программу следует оптимизировать, необходимо тщательно проверить алгоритм. Программу, подлежащую оптимизации следует разбить на подпрограммы (в соответствии с принципом структурного программирования). Если не возможно учесть время выполнения каждой подпрограммы подсчитайте количество операторов в подпрограмме, особенно выполненных в тело циклов. Ищите операторы, которые можно модифицировать – (do While и CASE) особенно это касается операторов цикла и ввода – вывода. Для каждой подпрограммы можно вычислить коэффициент: процент времени * процент улучшения. Программу с высоким коэффициентом следует оптимизировать в первую очередь. Здесь можно использовать два

подхода: «чистка» (использование очевидных неточностей в исходной программе) и перепрограммирование (если подвергается подпрограмма существенным изменениям).

Эффективность выполнения программ.

Эффективность программы во время выполнения определяйте использованием двух ресурсов. Первый – необходимое для работы время, второй - понять, какая, требуется программа. Хорошей считается программа, которая выполняется при минимальном расходе минимального времени. Вопрос о распределении памяти не представляет интереса до тех пор, пока ее достаточно. Но когда памяти явно не хватает, вопрос об экономии становится очевидным. С широким внедрением ПК в быт, с использованием мультимедийных возможностей появилась еще больше необходимости деления памяти на участки различного размера.

Тема №6: Виды ошибок. Основные принципы отладки ПС.

Виды ошибок:

1. Ошибки в описании задачи

Отсутствие взаимопонимания между программистом и заказчиком и качественное определение требований, приводит к получению нежелательных результатов, такие ошибки являются разрушительными и ведут к полному перепрограммированию.

2. Ошибки в выборе алгоритма

Неэффективный алгоритм может привести весь процесс программирования также к нежелательному результату, т.е. не эффективный метод может привести к перепрограммированию.

3. Ошибки анализа

Эти ошибки связаны с неполным учетом возникающих ситуаций (например, пренебрежение или не знание области допустимых значений переменных могут привести к нереальным результатам). Мелкие или крупные логические ошибки из которых можно выделить :

   a. отсутствие заданий начальных значений
   b. не верное условие окончания цикла
   c. не верная индексация цикла
   d. отсутствие задания обнуления циклов.

4. Ошибки общего характера - ошибки из-за не достаточного знания тонкостей языка или самой системы или машины.
5. Синтаксические ошибки - ошибки, вызванные не правильным написанием операторов.
6. Семантические ошибки – неправильное использование написанных операторов.
7. Ошибки в данных

Виды контроля ПС:

- Визуальный,
- Статический
- динамический

Визуальный контроль - это проверка программ “ за столом “, без использования компьютера.

1. сначала осуществляется чтение программы
2. затем осуществляется сквозной контроль программы (ее ручная прокрутка на нескольких заранее подобранных простых тестах).

Статический контроль - это проверка программы по ее тексту (без выполнения) с помощью инструментальных средств. *Формы статического контроля:*

1. синтаксический контроль программы с помощью компилятора, при котором проверяется соответствие текста программы синтаксическим правилам языка программирования.
2. контроль правдоподобия программы, то есть выявление в ее тексте конструкций, которые хотя и синтаксически корректны, но скорее всего содержат ошибку или свидетельствуют о ней. *Основные неправдоподобные ситуации* :

- использование в программе неинициализированных переменных (то есть переменных, не получивших начального значения) ;
- наличие в программе описаний переменных, процедур, меток, файлов, в дальнейшем не используемых в ее тексте;
- наличие в тексте программы фрагментов, никогда не выполняющихся;
- наличие в тексте программы переменных, ни разу не используемых для чтения после присваивая им значений;
- наличие в тексте программы заведомо бесконечных циклов ;

Даже если присутствие в тексте программы неправдоподобных конструкций не приводит к ее неправильной работе, исправление этого фрагмента повысит ясность и эффективность программы, т. е. благотворно скажется на ее качестве.

Основные цели и принципы отладки

***Отладка* ПС** - это деятельность, направленная на обнаружение и исправление ошибок в ПС с использованием процессов выполнения его программ.

***Тестирование* ПС** - это процесс выполнения его программ на некотором наборе данных, для которого заранее известен результат применения или известны правила поведения этих программ. Указанный набор данных называется тестовым или просто *тестом*.

Таким образом, отладку можно представить в виде многократного повторения трех процессов: тестирования, в результате которого может быть констатировано наличие в ПС ошибки, поиска места ошибки в программах и документации ПС и редактирования программ и документации с целью устранения обнаруженной ошибки.

Отладка = Тестирование + Поиск ошибок + Редактирование.

Каждому программисту известно, сколько времени и сил уходит на отладку программ. На этот этап приходится около 50% общей стоимости разработки программного обеспечения.

Тестирование - это процесс выполнения программы с целью обнаружения в ней ошибок. Но Нельзя гарантировать, что тестированием ПС можно установить наличие каждой имеющейся в ПС ошибки. Поэтому возникает две задачи:

1. подготовить такой набор тестов и применить к ПС, чтобы обнаружить в нем по возможности большее число ошибок. Однако чем дольше продолжается процесс тестирования (и отладки в целом), тем большей становится стоимость ПС.
2. определить момент окончания отладки ПС (или отдельной его компоненты). Отладка заканчивается, когда тестами охвачено множество различных ситуаций, возникающих при выполнении программ ПС, и относительно редко появляются ошибки в ПС на последнем отрезке процесса тестирования.

Заповеди отладки.

Заповедь 1. Считайте тестирование ключевой задачей разработки ПС, поручайте его самым квалифицированным и одаренным программистам; нежелательно тестировать свою собственную программу.

Заповедь 2. Хорош тот тест, для которого высока вероятность обнаружить ошибку, а не тот, который демонстрирует правильную работу программы.

Заповедь 3. Готовьте тесты как для правильных, так и для неправильных данных.

Заповедь 4. Избегайте невоспроизводимых тестов, документируйте их пропуск через компьютер; детально изучайте результаты каждого теста.

Заповедь 5. Каждый модуль подключайте к программе только один раз; никогда не изменяйте программу, чтобы облегчить ее тестирование.

Заповедь 6. Пропускайте заново все тесты, связанные с проверкой работы какой-либо программы ПС или ее взаимодействия с другими программами, если в нее были внесены изменения (например, в результате устранения ошибки).

Советы по организации тестирования

- необходимой частью каждого теста должно являться описание ожидаемых результатов работы программы, чтобы можно было быстро выяснить наличие или отсутствие ошибки в ней;
- должны являться правилом доскональное изучение результатов каждого теста, чтобы не пропустить малозаметную на поверхностный взгляд ошибку в программе;
- тестирования не должно планироваться исходя из предположения, что в программе не будут обнаружены ошибки;
- следует всегда помнить, что тестирование - творческий процесс, а не относиться к нему как к рутинному занятию.

Тема №7: Основные принципы организации тестирования ПС

Существует 4 этапа тестирования многомодульных ПС:

1. тестирование отдельных модулей;
2. совместное тестирование модулей;
3. тестирование функций программного комплекса;
4. тестирование всего комплекса в целом.

На первых двух этапах используются прежде всего методы структурного программирования. При структурном тестировании программа рассматривается как “белый ящик” (т.е. ее текст открыт для пользования). Происходит проверка логики программы.

Совместное тестирование модулей

Два подхода к совместному тестированию модулей:

1. *Пошаговое*. Каждый модуль для своего тестирования подключается к набору уже проверенных модулей. Здесь модули проверяются не изолированно друг от друга, поэтому требуются либо только ведущие модули, либо только заглушки.
2. *Монолитное*. Сначала по отдельности тестируются все модули программного комплекса, а затем все они объединяются в рабочую программу для комплексного тестирования. Для каждого модуля требуется модуль, имитирующий вызов тестируемого модуля, и один или несколько модулей, имитирующих работу модулей, вызываемых из тестируемого. При пошаговом тестировании

Преимущества пошагового подхода:	**Преимущества монолитного тестирования:**
1. меньшая трудоемкость; 2. более раннее обнаружение ошибок в интерфейсах между модулями (их сборка начинается раньше, чем при монолитном тестировании); 3. легче отладка, то есть локализация ошибок (они в основном связаны с последним из подключенных модулей); 4. более совершенны результаты тестирования (более тщательная проверка совместного использования модулей).	1. меньше расход машинного времени (хотя из-за большей сложности отладки может потребоваться дополнительная проверка программы, и это преимущество будет сведено на нет); 2. предоставляется больше возможностей для организации параллельной работы на начальном этапе тестирования.

В целом более целесообразным является выбор пошагового тестирования. При его использовании возможны две стратегии подключения модулей: нисходящая и восходящая.

Нисходящее тестирование	**Восходящее тестирование**

начинается с головного модуля программы, а выбор следующего подключаемого модуля происходит из числа модулей, вызываемых из уже протестированных.	начинается с терминальных модулей (т.е. тех, которые не вызывают никаких других модулей программы).
Проблемы и недостатки	
1. Создание заглушек. Как правило, недостаточно, чтобы в заглушке выполнялся вывод соответствующего информационного сообщения и возврат всегда одних и тех же значений выходных данных. 2. появляется соблазн совмещения нисходящего проектирования с тестированием; 3. может возникнуть желание перейти к тестированию модуля следующего уровня до завершения тестирования предыдущего по объективным причинам (необходимости создания нескольких версий заглушек, использования модулями верхнего уровня ресурсов модулей нижних уровней).	1. Проверка всей структуры разрабатываемого программного комплекса возможна только на завершающей стадии тестирования. 2. тестовые данные готовятся, как правило, не в той форме, которая рассчитана на пользователя (кроме случая, когда отлаживается последний, головной, модуль отлаживаемой программ); 3. большой объем отладочного программирования (при отладке одного модуля часто приходится составлять для разных тестов много ведущих отладочных модулей); 4. необходимость специального тестирования сопряжения модулей.
Достоинства	
1. уже на ранней стадии тестирования есть возможность получить работающий вариант разрабатываемой программы; 2. быстро могут быть выявлены ошибки, связанные с организацией взаимодействия с пользователем; 3. большинство тестов готовится в форме, рассчитанной на пользователя; 4. во многих случаях относительно небольшой объем отладочного программирования (имитаторы модулей, как правило, весьма просты и каждый пригоден для большого числа, нередко - для всех, тестов); 5. отпадает необходимость тестирования сопряжения модулей.	1. Нет проблемы выбора следующего подключаемого модуля - учитывается лишь то, чтобы он вызывал только уже протестированные модули. 2. В отличие от заглушек ведущие модули не должны иметь несколько версий, поэтому их разработка в большинстве случаев проще. 3. Поскольку нет промежуточных модулей (тестируемый модуль является для рабочего варианта программы модулем самого верхнего уровня), нет проблем, связанных с возможностью или трудностью задания тестов; 4. нет возможности совмещения проектирования с

	тестированием; 5. нет трудностей, вызывающих желание перейти к тестированию следующего модуля, не завершив проверки предыдущего. 6. простота подготовки тестов; 7. возможность полной реализации плана тестирования модуля.

Правила выбора тестируемых модулей: модули, содержащие операции ввода-вывода, должны подключаться к тестированию как можно раньше;
Наиболее важные для программы в целом модули должны тестироваться в первую очередь.
Хотя однозначного вывода о преимуществах той или иной стратегии пошагового тестирования сделать нельзя (нужно учитывать конкретные характеристики тестируемой программы), в большинстве случаев более предпочтительным является восходящее тестирование. Однако чаще применяют некоторые модификации нисходящего тестирования, либо некоторую комбинацию нисходящего и восходящего тестирования.

Тестирование функций ПС

Используются методы функционального тестирования. При функциональном тестировании программа рассматривается как “черный ящик” (то есть ее текст не используется). Происходит проверка соответствия поведения программы ее внешней спецификации.

Тестирование всего программного комплекса

При комплексной отладке тестируется ПС в целом, причем тесты готовятся по каждому из документов ПС. Тестирование этих документов производится в порядке, обратном их разработке. Тестирование при комплексной отладке представляет собой применение ПС к конкретным данным, которые в принципе могут возникнуть у пользователя, но, возможно, в моделируемой (а не в реальной) среде. Например, некоторые недоступные при комплексной отладке устройства ввода и вывода могут быть заменены их программными имитаторами.
Тестирование качества ПС. Целью тестирования является поиск нарушений требований качества, сформулированных в спецификации качества ПС. Это наиболее трудный и наименее изученный вид тестирования. Ясно лишь, что далеко не каждый примитив качества ПС может быть испытан тестированием. Точность, устойчивость, защищенность, временная эффективность, в какой-то мере - эффективность по памяти, эффективность по устройствам, расширяемость и, частично, независимость от устройств могут тестироваться. Легкость применения ПС оценивается при тестировании документации по применению ПС.
Тестирование документации по применению ПС. Целью тестирования является поиск несогласованности документации по применению и совокупностью программ ПС, а также неудобств применения ПС. Этот этап непосредственно предшествует подключению пользователя к завершению разработки ПС (тестированию требований к ПС и аттестации ПС), поэтому весьма важно разработчикам сначала самим воспользоваться ПС так, как это будет делать пользователь. Все тесты на этом этапе готовятся исключительно на основании только документации по применению ПС. Должны быть протестированы все неясные места в документации, а также все примеры, использованные в документации.
Тестирование определения требований к ПС. Целью тестирования является выяснение, в какой мере ПС не соответствует предъявленному определению требований к нему. Особенность этого вида тестирования заключается в том, что его осуществляет организация-покупатель или организация-пользователь ПС как один из путей преодоления барьера между разработчиком и

пользователем. Обычно это тестирование производится с помощью контрольных задач - типовых задач, для которых известен результат решения. В тех случаях, когда разрабатываемое ПС должно прийти на смену другому варианту ПС, который решает хотя бы часть задач разрабатываемого ПС, тестирование производится путем решения общих задач с помощью, как старого, так и нового ПС с последующим сопоставлением полученных результатов. Иногда в качестве формы такого тестирования используют *опытную* эксплуатацию ПС - ограниченное применение нового ПС с анализом использования результатов в практической деятельности. По-существу, этот вид тестирования во многом перекликается с испытанием ПС при его аттестации, но выполняется до аттестации, а иногда и вместо аттестации.

Тема №8: Виды программных документов.

Документация, создаваемая в процессе разработки программных средств.

При разработке ПС создается большой объем разнообразной документации. Она необходима как средство передачи информации между разработчиками ПС, как средство управления разработкой ПС и как средство передачи пользователям информации, необходимой для применения и сопровождения ПС. На создание этой документации приходится большая доля стоимости ПС.

Эту документацию можно разбить на две группы:

- Документы управления разработкой ПС.
- Документы, входящие в состав ПС.

Документы управления разработкой ПС, протоколируют процессы разработки и сопровождения ПС, обеспечивая связи внутри коллектива разработчиков и между коллективом разработчиков и *менеджерами* - лицами, управляющими разработкой. Эти документы могут быть следующих типов:

- *Планы, оценки, расписания*. Эти документы создаются менеджерами для прогнозирования и управления процессами разработки и сопровождения.
- *Отчеты об использовании ресурсов в процессе разработки*. Создаются менеджерами.
- *Стандарты*. Эти документы предписывают разработчикам, каким принципам, правилам, соглашениям они должны следовать в процессе разработки ПС. Эти стандарты могут быть как международными или национальными, так и специально созданными для организации, в которой ведется разработка данного ПС.
- *Рабочие документы*. Это основные технические документы, обеспечивающие связь между разработчиками. Они содержат фиксацию идей и проблем, возникающих в процессе разработки, описание используемых стратегий и подходов, а также рабочие (временные) версии документов, которые должны войти в ПС.
- *Заметки и переписка*. Эти документы фиксируют различные детали взаимодействия между менеджерами и разработчиками.

Документы, входящие в состав ПС, описывают программы ПС как с точки зрения их применения пользователями, так и с точки зрения их разработчиков и сопроводителей (в соответствии с назначением ПС). Здесь следует отметить, что эти документы будут использоваться не только на стадии эксплуатации ПС (в ее фазах применения и сопровождения), но и на стадии разработки для управления процессом разработки (вместе с рабочими документами) - во всяком случае они должны быть проверены (протестированы) на соответствие программам ПС. Эти документы образуют два комплекта с разным назначением:

- Пользовательская документация ПС (П-документация).
- Документация по сопровождению ПС (С-документация).

Пользовательская документация программных средств.

Пользовательская документация ПС объясняет пользователям, как они должны действовать, чтобы применить данное ПС. К такой документации относятся документы, которыми руководствуется пользователь при *инсталяции* ПС (при установке ПС с соответствующей

настройкой на среду применения ПС), при применении ПС для решения своих задач и при управлении ПС (например, когда данное ПС взаимодействует с другими системами).
Следует различать две категории пользователей ПС:

- ординарных пользователей ПС
- администраторов ПС.

Ординарный пользователь ПС использует ПС для решения своих задач (в своей предметной области). Это может быть инженер, проектирующий техническое устройство, или кассир, продающий железнодорожные билеты с помощью ПС. Он может и не знать многих деталей работы компьютера или принципов программирования.
Администратор ПС управляет использованием ПС ординарными пользователями и осуществляет сопровождение ПС, не связанное с модификацией программ. Например, он может регулировать права доступа к ПС между ординарными пользователями, поддерживать связь с поставщиками ПС или выполнять определенные действия, чтобы поддерживать ПС в рабочем состоянии, если оно включено как часть в другую систему.
Состав пользовательской документации для достаточно больших ПС:

- *Общее функциональное описание ПС*. Дает краткую характеристику функциональных возможностей ПС. Предназначено для пользователей, которые должны решить, насколько необходимо им данное ПС.
- *Руководство по инсталяции ПС*. Предназначено для системных администраторов. Он должен детально предписывать, как устанавливать системы в конкретной среде. Он должен содержать описание машинно-считываемого носителя, на котором поставляется ПС, файлы, представляющие ПС, и требования к минимальной конфигурации аппаратуры.
- *Инструкция по применению ПС*. Предназначена для ординарных пользователей. Содержит необходимую информацию по применению ПС, организованную в форме удобной для ее изучения.
- *Справочник по применению ПС*. Предназначен для ординарных пользователей. Содержит необходимую информацию по применению ПС, организованную в форме удобной для избирательного поиска отдельных деталей.
- *Руководство по управлению ПС*. Предназначено для системных администраторов. Оно должно описывать сообщения, генерируемые, когда ПС взаимодействует с другими системами, и как реагировать на эти сообщения. Кроме того, если ПС использует системную аппаратуру, этот документ может объяснять, как сопровождать эту аппаратуру.

Разработка пользовательской документации начинается сразу после создания внешнего описания. Качество этой документации может существенно определять успех ПС. Она должна быть достаточно проста и удобна для пользователя. Поэтому, хотя черновые варианты (наброски) пользовательских документов создаются основными разработчиками ПС, к созданию их окончательных вариантов часто привлекаются профессиональные технические писатели. Кроме того, для обеспечения качества пользовательской документации разработан ряд стандартов, в которых предписывается порядок разработки этой документации, формулируются требования к каждому виду пользовательских документов и определяются их структура и содержание .

Документация по сопровождению программных средств.

Документация по сопровождению ПС описывает ПС с точки зрения ее разработки. Эта документация необходима, если ПС предполагает изучение того, как оно устроена, и модернизацию его программ. Сопровождение - это продолжающаяся разработка. Поэтому в случае необходимости модернизации ПС к этой работе привлекается специальная команда разработчиков-сопроводителей. Этой команде придется иметь дело с такой же документацией,

которая определяла деятельность команды первоначальных (основных) разработчиков ПС, - с той лишь разницей, что эта документация для команды разработчиков-сопроводителей будет, как правило, чужой (она создавалась другой командой). Команда разработчиков-сопроводителей должна будет изучать эту документацию, чтобы понять строение и процесс разработки модернизируемого ПС, и внести в эту документацию необходимые изменения, повторяя в значительной степени технологические процессы, с помощью которых создавалось первоначальное ПС.

Документация по сопровождению ПС можно разбить на две группы:

(1) документация, определяющая строение программ и структур данных ПС и технологию их разработки;

(2) документацию, помогающую вносить изменения в ПС.

Документация первой группы содержит итоговые документы каждого технологического этапа разработки ПС. Она включает следующие документы:

- Внешнее описание ПС.
- Описание архитектуры ПС, включая внешнюю спецификацию каждой ее программы.
- Для каждой программы ПС - описание ее модульной структуры, включая внешнюю спецификацию каждого включенного в нее модуля.
- Для каждого модуля - его спецификация и описание его строения.
- Тексты модулей на выбранном языке программирования.
- Документы установления достоверности ПС, описывающие, как устанавливалась достоверность каждой программы ПС и как информация об установлении достоверности связывалась с требованиями к ПС.

Документация второй группы содержит

- Руководство по сопровождению ПС, которое описывает известные проблемы вместе с ПС, описывает, какие части системы являются аппаратно- и программно-зависимыми, и как развитие ПС принято в расчет в его строении (конструкции).

Общая проблема сопровождения ПС - обеспечить, чтобы все его представления шли в ногу, когда ПС изменяется. Чтобы этому помочь, связи и зависимости между документами и их частями должны быть зафиксированы в базе данных управления конфигурацией.

Тема №9: Обеспечение функциональности, надежности и качества ПС. Технологии оценки качества ПС.

Обсудим обеспечение примитивов качества ПС, выражающих критерии функциональности и надежности ПС.

Обеспечение завершенности программного средства.

Завершенность ПС является общим примитивом качества ПС для выражения и функциональности и надежности ПС.

Функциональность ПС определяется его функциональной спецификацией. Завершенность ПС как примитив его качества является мерой того, насколько эта спецификация реализована в данном ПС. Обеспечение этого примитива в полном объеме означает реализацию каждой из функций, определенной в функциональной спецификации, со всеми указанными там деталями и особенностями.

Однако в спецификации качества ПС могут быть определены несколько уровней реализации функциональности ПС: может быть определена некоторая упрощенная (начальная или стартовая) версия, которая должна быть реализована в первую очередь, могут быть также определены и несколько промежуточных версий. В этом случае возникает дополнительная технологическая задача: организация наращивания функциональности ПС. Упрощенная версия требуемого ПС должна быть рассчитана на практически полезное применение любыми пользователями, для которых оно предназначено. Поэтому главный принцип обеспечении функциональности такого ПС заключается в том, чтобы с самого начала разрабатывать ПС

таким образом, как будто требуется ПС в полном объеме, до тех пор, пока разработчики не будут иметь дело с теми частями или деталями ПС, реализацию которых можно отложить в соответствии со спецификацией его качества. Тем самым, и внешнее описание и описание архитектуры ПС должно быть разработано в полном объеме. Можно отложить лишь реализацию тех программных подсистем, определенных в архитектуре разрабатываемого ПС, функционирования которых не требуется в начальной версии этого ПС.

Обеспечение точности программного средства.

- Обеспечение этого примитива связано с действиями над значениями вещественных типов (точнее говоря, со значениями, представляемыми с некоторой погрешностью). Обеспечить требуемую точность при вычислении значения той или иной функции - значит получить это значение с погрешностью, не выходящей за рамки заданных границ.

Обеспечение автономности программного средства.

Этот примитив качества обеспечивается на этапе спецификации качества принятием решения об использовании в разрабатываемом ПС какого-либо подходящего базового программного обеспечения или не использовании в нем никакого базового программного обеспечения. При повышенных требованиях к надежности разрабатываемого ПС надежность базового программного обеспечения, имеющегося в распоряжении разработчиков, может оказаться неудовлетворительной, поэтому от его использования приходиться отказываться, а реализацию его функций в требуемом объеме приходится включать в состав ПС. Аналогичные решения приходится принимать при жестких ограничениях на используемые ресурсы.

Обеспечение устойчивости программного средства.

Этот примитив качества обеспечивается с помощью так называемого защитного программирования. Вообще говоря, защитное программирование применяется для повышения надежности ПС при программировании модуля в более широком смысле. Защитное программирование основано на важной предпосылке: худшее, что может сделать модуль, - это принять неправильные входные данные и затем вернуть неверный, но правдоподобный результат. Для того, чтобы этого избежать, в текст модуля включают проверки его входных и выходных данных на их корректность в соответствии со спецификацией этого модуля, в частности, должны быть проверены выполнение ограничений на входные и выходные данные и соотношений между ними, указанные в спецификации модуля. В случае отрицательного результата проверки возбуждается соответствующая исключительная ситуация. В связи с этим в конец этого модуля включаются фрагменты второго рода - обработчики соответствующих исключительных ситуаций, которые помимо выдачи необходимой диагностической информации, могут принять меры либо по исключению ошибки в данных (например, потребовать их повторного ввода), либо по ослаблению влияния ошибки (например, мягкую остановку управляемых ПС устройств во избежание их поломки при аварийном прекращении выполнения программы).

Применение защитного программирования модулей приводит к снижению эффективности ПС как по времени, так и по памяти. Поэтому необходимо разумно регулировать степень применения защитного программирования в зависимости от требований к надежности и эффективности ПС,. Входные данные разрабатываемого модуля могут поступать как непосредственно от пользователя, так и от других модулей. Наиболее употребительным случаем применения защитного программирования является применение его для первой группы данных, что и означает реализацию устойчивости ПС.

Обеспечение защищенности программных средств.

Различают следующие виды защиты ПС от искажения информации:

- Защита от сбоев аппаратуры.
- Защита от влияния "чужой" программы.
- защита от отказов "своей" программы;

- защита от ошибок оператора (пользователя);
- защита от несанкционированного доступа;
- защита от защиты.
- Защита от влияния "чужой" программы

Защита от сбоев аппаратуры в настоящее время является не очень злободневной задачей (с учетом уровня достигнутой надежности компьютеров).
Защита от влияния «чужой программы» относится прежде всего к операционным системам или к программам, выполняющим частично их функции. Различают две разновидности этой защиты:

- защита от отказов,
- защита от злонамеренного влияния "чужой" программы.

При появлении мультипрограммного режима работы компьютера в его памяти может одновременно находится в стадии выполнения несколько программ, попеременно получающих управление в результате возникающих прерываний. Одна из таких программ (обычно: операционная система) занимается обработкой прерываний и управлением мультипрограммным режимом. В каждой из таких программ могут возникать отказы, которые могут повлиять на выполнение функций другими программами. Поэтому управляющая программа должна обеспечить защиту себя и других программ от такого влияния. Для этого аппаратура компьютера должна реализовывать следующие возможности:

- защиту памяти,
- два режима функционирования компьютера: привилегированный и рабочий (пользовательский),
- два вида операций: привилегированные и ординарные,
- корректную реализацию прерываний и начального включения компьютера,
- временное прерывание.

Защита памяти означает возможность программным путем задавать для каждой программы недоступные для нее участки памяти. *В привилегированном режиме могут выполняться любые операции (как ординарные, так и привилегированные), а в рабочем режиме - только ординарные. Попытка выполнить привилегированную операцию, а также обратиться к защищенной памяти в рабочем режиме вызывает соответствующее прерывание. Причем к привилегированным операциям относятся операции изменения защиты памяти и режима функционирования, а также доступа к внешней информационной среде. Начальное включение компьютера и любое прерывание должно автоматически включать привилегированный режим и отмену защиты памяти. В этом случае управляющая программа (операционная система) может полностью защитить себя от влияния других программ, если все точки передачи управления при начальном включении и прерываниях будут принадлежать этой программе, если она никакой другой программе не позволит работать в привилегированном режиме (при передаче управления любой другой программе будет включаться только рабочий режим) и если она полностью защитит свою память (содержащую, в частности, всю ее управляющую информацию, включая так называемые вектора прерываний) от других программ. Тогда никто не помешает ей выполнять любые реализованные в ней функции защиты других программ (в том числе и доступа к внешней информационной среде). Для облегчения решения этой задачи часть такой программы помещается в постоянную память, т.е. неотделима от самого компьютера. Наличие временного прерывания позволяет управляющей программе защититься от зацикливания в других программах (без такого прерывания она могла бы просто лишиться возможности управлять).*
Защита от отказов "своей" программы обеспечивается надежностью этой программы.

Защита от ошибок пользователя обеспечивается выдачей предупредительных сообщений о попытках изменить состояние внешней информационной среды с требованием подтверждения этих действий, а также возможностью восстановления состояния отдельных компонент внешней информационной среды.
Защита от несанкционированного доступа обеспечивается использованием паролей. В этом случае каждому пользователю предоставляются определенные информационные и процедурные ресурсы, для использования которых требуется предъявление некоторого пароля, ранее зарегистрированного в ПС этим пользователем. Однако, в отдельных случаях могут быть предприняты настойчивые попытки взломать такую защиту, если защищаемые ресурсы представляют для кого-то чрезвычайную ценность. Для такого случая приходится предпринимать дополнительные меры для защиты от взлома защиты.
Защита от взлома защиты связана с использованием в ПС специальных программистских приемов, затрудняющих преодоление защиты от несанкционированного доступа.
Использование обычных паролей оказывается недостаточной, когда речь идет о чрезвычайно настойчивом стремлении добиться доступа к ценной информации. Во-первых, потому, что информацию о паролях, которую использует ПС для защиты от несанкционированного доступа, "взломщик" этой защиты относительно легко может достать, если он имеет доступ к самому этому ПС. Во-вторых, используя компьютер, можно осуществлять достаточно большой перебор возможных паролей с целью найти подходящий для доступа к интересующей информации.
Защититься от такого взлома можно следующим образом. Пароль или просто секретное целое число X знает только владелец защищаемой информации, а для проверки прав доступа в компьютере хранится другое число Y=F(X), однозначно вычисляемое ПС при каждой попытке доступа к этой информации при предъявлении секретного слова. При этом функция F может быть хорошо известной всем пользователям ПС, однако она обладает таким свойством, что восстановление слова X по Y практически невозможно: при достаточно большой длине слова X (например, в несколько сотен знаков) для этого требуется астрономическое время. Такое число Y будем называть электронной (компьютерной) подписью владельца секретного слова X (а значит, и защищаемой информации).
Другая разновидность такой защиты связана с защитой сообщений, пересылаемых по компьютерным сетям, преднамеренных искажений. Такое сообщения может перехватываться на "перевалочных" пунктах компьютерной сети и подменяться другим сообщением от автора перехваченного сообщения. Такая ситуация возникает прежде всего при осуществлении банковских операций с использованием компьютерной сети. Путем подмены такого сообщения, являющего распоряжением владельца банковского счета о выполнении некоторой банковской операции деньги с его счета могут быть переведены на счет "взломщика" защиты. Защиту от такого взлома защиты можно осуществить следующим образом. Наряду с функцией F, определяющей компьютерную подпись владельца секретного слова X, которую знает адресат защищаемого сообщения (если только ее владелец является клиентом этого адресата), в ПС определена другая функция Stamp, по которой отправитель сообщения должен вычислить число S=Stamp(X,R), используя секретное слово X и текст передаваемого сообщения R. Функция Stamp также считается хорошо известной всем пользователям ПС и обладает таким свойством, что по S практически невозможно ни восстановить число X, ни подобрать другое сообщение R с соответствующей компьютерной подписью. Само передаваемое сообщение (вместе со своей защитой) должно иметь вид: R Y S , причем Y (компьютерная подпись) позволяет адресату установить истинность клиента, а S как бы скрепляет защищаемое сообщение Rc компьютерной подписью Y. В связи с этим будем называть число S электронной печатью. В ПС определена еще одна функция Notary, по которой получатель защищаемого сообщения проверяет истинность передаваемого сообщения: Notary(R,Y,S). Она позволяет однозначно установить, что сообщение R принадлежит владельцу секретного слова X.

Защита от защиты необходима в том случае, когда пользователь забыл (или потерял) свой пароль. Для такого случая должна быть предусмотрена возможность для особого пользователя (администратора ПС), отвечающего за функционирования системы защиты, производить временное снятие защиты от несанкционированного доступа для хозяина забытого пароля с целью дать ему возможность зафиксировать новый пароль.

ОБЕСПЕЧЕНИЕ КАЧЕСТВА ПРОГРАММНОГО СРЕДСТВА

Общая характеристика процесса обеспечения качества программного средства.

Спецификация качества определяет основные ориентиры, которые на всех этапах разработки ПС так или иначе влияют при принятии различных решений на выбор подходящего варианта. Обеспечение качества осуществляется в каждом технологическом процессе: принятые в нем решения в той или иной степени оказывают влияние на качество ПС в целом. В частности и потому, что значительная часть примитивов качества связана не столько со свойствами программ, входящих в ПС, сколько со свойствами документации. В силу противоречивости примитивов качества весьма важно придерживаться выбранных приоритетов в их обеспечении. Во всяком случае полезно придерживаться двух общих принципов:

- сначала необходимо обеспечить требуемую функциональность и надежность ПС, а затем уже доводить остальные критерии качества до приемлемого уровня их присутствия в ПС;
- нет никакой необходимости и может быть даже вредно добиваться более высокого уровня присутствия в ПС какого-либо примитива качества, чем тот, который определен в спецификации качества ПС.

Обеспечение легкости применения программного средства.

П-документированость ПС определяет состав пользовательской документации П-документированность и информативность определяют состав и качество пользовательской документации. Коммуникабельность обеспечивается созданием подходящего пользовательского интерфейса и соответствующей реализации исключительных ситуаций.

Обеспечение эффективности программного средства.

Эффективность ПС обеспечивается принятием подходящих решений на разных этапах его разработки. Особенно сильно на эффективность ПС влияет выбор структуры и представления данных. Но и выбор алгоритмов, используемых в тех или иных программных модулях, а также особенности их реализации (включая выбор языка программирования) может существенно повлиять на эффективность ПС. При этом постоянно приходится разрешать противоречие между временной эффективностью и эффективностью по памяти. Поэтому весьма важно, чтобы в спецификации качества было явно указано количественное соотношение между показателями этих примитивов качества или хотя бы заданы количественные границы для одного из этих показателей.

С учетом сказанного, рекомендуется придерживаться следующих принципов для обеспечения эффективности ПС:

- сначала нужно разработать надежное ПС, а уж потом добиваться требуемой его эффективности в соответствии со спецификацией качества этого ПС;
- для повышения эффективности ПС используйте прежде всего оптимизирующий компилятор - это может обеспечить требуемую эффективность;
- если достигнутая эффективность ПС не удовлетворяет спецификации его качества, то найдите самые критические модули с точки зрения требуемой эффективности ПС; эти модули и попытайтесь оптимизировать в первую очередь путем их ручной переделки;
- не занимайтесь оптимизацией модуля, если этого не требуется для достижения требуемой эффективности ПС.

Обеспечение сопровождаемости.

С-документированность, информативность и понятность определяют состав и качество документации по сопровождению. Кроме того, относительно текстов программ (модулей) можно сделать следующие рекомендации.

- используйте в тексте модуля комментарии, проясняющие и объясняющие особенности принимаемых решений; По-возможности, включайте комментарии на самой ранней стадии разработки текста модуля;
- используйте осмысленные и устойчиво различимые имена, не используйте сходные имена и ключевые слова;
- соблюдайте осторожность в использовании констант (уникальная константа должна иметь единственное вхождение в текст модуля: при ее объявлении или, в крайнем случае, при инициализации переменной в качестве константы);
- не бойтесь использовать не обязательные скобки;
- размещайте не больше одного оператора в строке; для прояснения структуры модуля используйте дополнительные пробелы в начале каждой строки;
- избегайте трюков, т.е. таких приемов программирования, когда создаются фрагменты модуля, основной эффект которых не очевиден или скрыт, например, побочные эффекты функций.

Расширяемость обеспечивается созданием подходящего инсталлятора.
Структурированность и модульность упрощают как понимание текстов программ, так и их модификацию.

Тема №10: Аттестация программного средства

Назначение аттестации программного средства.

Аттестация ПС - это авторитетное подтверждение качества ПС. Обычно для аттестации ПС создается аттестационная комиссия из экспертов, представителей заказчика и представителей разработчика. Эта комиссия проводит *испытания* ПС с целью получения необходимой информации для оценки его качества. Под испытанием ПС мы будем понимать процесс проведения комплекса мероприятий, исследующих пригодность ПС для успешной его эксплуатации (применения и сопровождения) в соответствии с требованиями заказчика. Этот комплекс включает проверку полноты и точности программной документации, изучение и обсуждение других ее свойств, а также необходимое тестирование программ, входящих в состав ПС, и, в частности, соответствия этих программ имеющейся документации.
На основе информации, полученной во время испытаний ПС, прежде всего должно быть установлено, что ПС выполняет декларированные функции, а также должно быть установлено, в какой степени ПС обладает декларированными примитивами и критериями качества. Таким образом, оценка качества ПС является основным содержанием процесса аттестации.
Произведенная оценка качества ПС фиксируется в соответствующем решении аттестационной комиссии.

Виды испытаний программного средства.

Известны следующие виды испытаний ПС, проводимых с целью аттестации ПС:

- испытания компонент ПС;
- системные испытания;
- приемо-сдаточные испытания;
- полевые испытания;
- промышленные испытания.

Испытания компонент ПС - это проверка (тестирование) работоспособности отдельных подсистем ПС. Проводятся только в исключительных случаях по специальному решению аттестационной комиссии.
Системные испытания ПС - это проверка (тестирование) работоспособности ПС в целом. Может включать те же виды тестирования, что и при комплексной отладке ПС (см. лекцию 10).

Проводится по решению аттестационной комиссии, если возникают сомнения в качестве проведения отладки разработчиками ПС.
Приемо-сдаточные испытания являются основным видом испытаний при аттестации ПС. Именно с этих испытаний начинает работу аттестационная комиссия. Эти испытания начинаются с изучения представленной документации, в том числе, и документации по тестированию и отладке ПС. Если в документации отсутствуют достаточно полные результаты тестирования ПС, аттестационная комиссия может принять решение о проведении системных испытаний ПС или о прекращении процесса аттестации с рекомендацией разработчику провести дополнительное (более полное) тестирование ПС. Кроме того, во время этих испытаний могут выборочно пропускаться тесты разработчиков, а также контрольные задачи пользователей и дополнительные тесты, подготовленные комиссией для оценки качества аттестуемого ПС.
Полевые испытания ПС - это демонстрация ПС вместе с технической системой, которой управляет эта ПС, узкому кругу заказчиков в реальных условиях и осуществляется тщательное наблюдение за поведением ПС. Заказчикам должна быть предоставлена возможность задания собственных контрольных примеров, в частности, с выходов в критические режимы работы технической системы, а также с вызовом в ней аварийных ситуаций. Это дополнительные испытания, проводимые по решению аттестационной комиссии только для некоторых ПС, управляющих определенными техническими системами.
Промышленные испытания ПС - это процесс передачи ПС в постоянную эксплуатацию пользователям. Представляет собой период опытной эксплуатации ПС пользователями со сбором информации об особенностях поведения ПС и ее эксплуатационных характеристиках. Это завершающие испытания ПС, которые проводятся по решению аттестационной комиссии, если на предшествующих испытаниях получена недостаточно полная или надежная информация для оценки качества аттестуемого ПС.

Методы оценки качества программного средства.

Оценка качества ПС по каждому из критериев сводится к оценке каждого из примитивов, связанных с этим критерием качества ПС, в соответствии с их конкретизацией, произведенной в спецификации качества этого ПС. Методы оценки примитивов качества ПС можно разделить на четыре группы:

- непосредственное измерение показателей примитива качества;
- обработка программ и документации ПС специальными программными инструментами (процессорами);
- тестирование программ ПС;
- экспертная оценка на основании изучения программ и документации ПС.

Непосредственное измерение показателей примитива качества производится путем подсчета числа вхождений в тот или иной программный документ характерных единиц, объектов, конструкций и т.п., а также путем измерения времени работы различных устройств и объема занятой памяти ЭВМ при выполнении контрольных примеров. Например, некоторым показателем эффективности по памяти может быть число строк программы на языке программирования, а некоторым показателем эффективности по времени может быть время ответа на запрос. Использование каких-либо показателей для примитивов качества может определяться в спецификации качества ПС. Метод непосредственного измерения показателей примитива качества может сочетаться с использованием тестирования программ.
Для установления наличия у ПС некоторых примитивов качества могут использоваться определенные программные инструментальные средства. Такие программные инструменты обрабатывают тексты программ или программной документации с целью контроля каких-либо примитивов качества или получения некоторых показателей этих примитивов качества. Для оценки структурированности программ ПС, если они программировались на подходящем структурном диалекте базового языка программирования, достаточно было бы их пропустить

через конвертер структурированных программ, осуществляющий синтаксический и некоторый семантический контроль этого диалекта и переводящий тексты этих программ на входной язык базового транслятора. Однако таким путем в настоящее время удается контролировать лишь небольшое число примитивов качества, да и то в редких случаях. В ряде случаев вместо программных инструментов, контролирующих качество ПС, полезнее применять инструменты, осуществляющие преобразование представления программ или программной документации. Таким, например, является форматер программ, приводящий тексты программ к удобочитаемому виду, - обработка текстов программ ПС таким инструментом может автоматически обеспечить наличие соответствующего примитива качества у ПС.

Для оценки некоторых примитивов качества ПС используется тестирование. К таким примитивам относится прежде всего завершенность ПС, а также его точность, устойчивость, защищенность и другие примитивы качества. В ряде случаев для оценки отдельных примитивов качества ПС тестирование применяется в сочетании с другими методами. Так для оценки качества документации по применению ПС тестирование применяется в сочетании с экспертной оценкой этой документации. Если при комплексной отладке ПС было проведено достаточно полное тестирование, то эти же тесты могут быть использованы и при аттестации ПС. В этом случае аттестационная комиссия может воспользоваться протоколами тестирования, проведенного при комплексной отладки. Однако и в этом случае необходимо выполнить какие-либо новые тесты или хотя бы повторно некоторые старые. Если же тестирование при комплексной отладке будет признано недостаточно полным, то необходимо провести более полное тестирование. В этом случае может быть принято решение о проведении испытаний компонент или системных испытаний ПС, а также о возврате ПС разработчикам на доработку.

Весьма важно, чтобы для оценки ПС по критерию легкости применения было проведено (во время отладки и аттестации ПС) полное тестирование по тестам, подготовленным на основании документации по применению, а по критерию сопровождаемости - по тестам, подготовленным по каждому из документов, предлагаемых для сопровождения ПС.

Для оценки большинства примитивов качества ПС в настоящее время можно применять только метод экспертных оценок. Этот метод заключается в следующем: назначается группа экспертов, каждый из этих экспертов в результате изучения представленной документации составляет свое мнение об обладании ПС требуемым примитивом качества, а затем голосованием членов этой группы устанавливается оценка требуемого примитива качества ПС. Эта оценка может производиться как по двухбалльной системе ("обладает" - "не обладает"), так и учитывать степень обладания ПС этим примитивом качества (например, производиться по пятибалльной системе). При этом группа экспертов должна руководствоваться конкретизацией этого примитива и указанием о способе его оценки, сформулированными в спецификации качества аттестуемого ПС.