**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**
Государственное образовательное учреждение
высшего профессионального образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

УТВЕРЖДАЮ
Заведующий кафедрой ЭМИС
_____ И. Г. Боровской
«___» _____ 2012 г.

А.Н. Стась

# Технология разработки программного обеспечения

***Методические указания по выполнению лабораторных работ и самостоятельной работе для магистрантов направления 230100 «Информатика и вычислительная техника»***

2012

Содержание

*Лабораторная работа 1.* Методы разработки эффективных алгоритмов. .....3
*Лабораторная работа 2.* Разработка описания и анализ информационной системы.....14
Самостоятельная работа 1. Методология объектно-ориентированного моделирования....16
*Самостоятельная работа 2.* Методология управление проектами.....16

***Лабораторная работа 1.*** **Методы разработки эффективных алгоритмов.**

**Цель работы:**

Сравнение эффективных и неэффективных алгоритмов сортировки массивов.

## Теоретический материал

Введем некоторые понятия и обозначения. Если у нас есть элементы

$$a_1, a_2, \ldots\ldots, a_n$$

то сортировка есть перестановка этих элементов в массив

$$a_{k1}, a_{k2}, \ldots\ldots, a_{kn}$$

где

$$a_{k1} <= a_{k2} <= \ldots.. <= a_{kn}.$$

Считаем, что тип элемента определен как INTEGER .

```
Const n=???; //здесь указывается нужная длина массива
Var A: array[1..n] of integer;
```

Выбор INTEGER до некоторой степени произволен. Можно было взять и другой тип, на котором определяется общее отношение порядка.

Мы будем сначала классифицировать методы по их экономичности, т. е. по времени их работы. Хорошей мерой эффективности может быть $C$ – число необходимых сравнений ключей и $M$ – число пересылок (перестановок) элементов. Эти числа суть функции от $n$ – числа сортируемых элементов. Хотя хорошие алгоритмы сортировки требуют порядка $n*\log n$ сравнений, мы сначала разберем несколько простых и очевидных методов, их называют ***прямыми***, где требуется порядка $n^2$ сравнений ключей.

**Сортировка с помощью прямого включения**.

Такой метод широко используется при игре в карты. Элементы мысленно делятся на уже “готовую” последовательность $a_1, \ldots, a_{i-1}$ и исходную последовательность. При каждом шаге, начиная с $I = 2$ и увеличивая $i$ каждый раз на единицу, из исходной последовательности извлекается $i$- й элементы и перекладывается в готовую последовательность, при этом он вставляется на нужное место.

**Таблица.Пример сортировки с помощью прямого включения**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Начальные ключи | 44 | 55 | 12 | 42 | 94 | 18 | 06 | 67 |
| I = 2 | 44 | 55 | 12 | 42 | 94 | 18 | 06 | 67 |
| I = 3 | 12 | 44 | 55 | 42 | 94 | 18 | 06 | 67 |
| I = 4 | 12 | 42 | 44 | 55 | 94 | 18 | 06 | 67 |
| I = 5 | 12 | 42 | 44 | 55 | 94 | 18 | 06 | 67 |
| I = 6 | 12 | 18 | 42 | 44 | 55 | 94 | 06 | 67 |
| I = 7 | 06 | 12 | 18 | 42 | 44 | 55 | 94 | 67 |
| I = 8 | 06 | 12 | 18 | 42 | 44 | 55 | 67 | 94 |

ПРОГРАММА СОРТИРОВКА С ПОМОЩЬЮ ПРЯМОГО ВКЛЮЧЕНИЯ.

```
PROGRAM SI;
VAR
 I,J,N,X:INTEGER;
 A:ARRAY[0..50] OF INTEGER;
BEGIN
 WRITELN('Введите длину массива');
 READ(N);
 WRITELN('Введите массив');
 FOR I:=1 TO N DO READ(A[I]);
 FOR I:=2 TO N DO BEGIN
  X:=A[I];
  A[0]:=X;
  J:=I;
  WHILE X<A[J-1] DO BEGIN
   A[J]:=A[J-1];
   DEC(J)
  END;
  A[J]:=X
 END;
 WRITELN('Результат:');
```

```
 FOR I:=1 TO N DO WRITE(A[I],' ')
END.
```

**Сортировка с помощью прямого выбора**

Этот прием основан на следующих принципах:

1. Выбирается элемент с наименьшим ключом.
2. Он меняется местами с первым элементом $a_1$.
3. Затем этот процесс повторяется с оставшимися *n-1* элементами, *n-2* элементами и т.д. до тех пор, пока не останется один, самый большой элемент.

**Пример сортировки с помощью прямого выбора**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Начальные ключи | 44 | 55 | 12 | 42 | 94 | 18 | 06 | 67 |
| I = 2 | 06 | 55 | 12 | 42 | 94 | 18 | 44 | 67 |
| I = 3 | 06 | 12 | 55 | 42 | 94 | 18 | 44 | 67 |
| I = 4 | 06 | 12 | 18 | 42 | 94 | 55 | 44 | 67 |
| I = 5 | 06 | 12 | 18 | 42 | 94 | 55 | 44 | 67 |
| I = 6 | 06 | 12 | 18 | 42 | 44 | 55 | 94 | 67 |
| I = 7 | 06 | 12 | 18 | 42 | 44 | 55 | 94 | 67 |
| I = 8 | 06 | 12 | 18 | 42 | 44 | 55 | 94 | 67 |

Такой метод сортировки – его называют прямым выбором – в некотором смысле противоположен прямому включению. Полностью алгоритм прямого выбора приводится в программе 2.3.

ПРОГРАММА. СОРТИРОВКА С ПОМОЩЬЮ ПРЯМОГО ВЫБОРА.

```
PROGRAM SS;
VAR I,J,R,X,N:INTEGER;
   A:ARRAY[0..50] OF INTEGER;
BEGIN
 WRITELN('Введи длину массива');
```

```
READ(N);
WRITELN('Введи  массив');
FOR I:=1 TO N DO READ(A[I]);
FOR I:=1 TO N-1 DO BEGIN
 R:=I;
 X:=A[I];
 FOR J:=I+1 TO N DO IF A[J]<X THEN BEGIN
  R:=J;
  X:=A[R]
 END;
 A[R]:=A[I];
 A[I]:=X
END;
WRITELN('Результат:');
FOR I:=1 TO N DO WRITE(A[I],' ')
END.
```

**Сортировка с помощью прямого обмена**.

Как и в методе прямого выбора, мы повторяем проходы по массиву, сдвигая каждый раз наименьший элемент оставшейся последовательности к левому концу массива. Если мы будем рассматривать как вертикальные, а не горизонтальные построения, то элементы можно интерпретировать как пузырьки в чане с водой, причем вес каждого соответствует его ключу (см. таб. 2.3.).

Пример пузырьковой сортировки.

| I = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 44 | 06 | 06 | 06 | 06 | 06 | 06 | 06 |
| 55 | 44 | 12 | 12 | 12 | 12 | 12 | 12 |
| 12 | 55 | 44 | 18 | 18 | 18 | 18 | 18 |
| 42 | 12 | 55 | 44 | 42 | 42 | 42 | 42 |
| 94 | 42 | 18 | 55 | 44 | 44 | 44 | 44 |
| 18 | 94 | 42 | 42 | 55 | 55 | 55 | 55 |
| 06 | 18 | 94 | 67 | 67 | 67 | 67 | 67 |
| 67 | 67 | 67 | 94 | 94 | 94 | 94 | 94 |

Такой метод сортировки известен под именем ***“пузырьковая сортировка”***. Он представлен в программе 2.4.

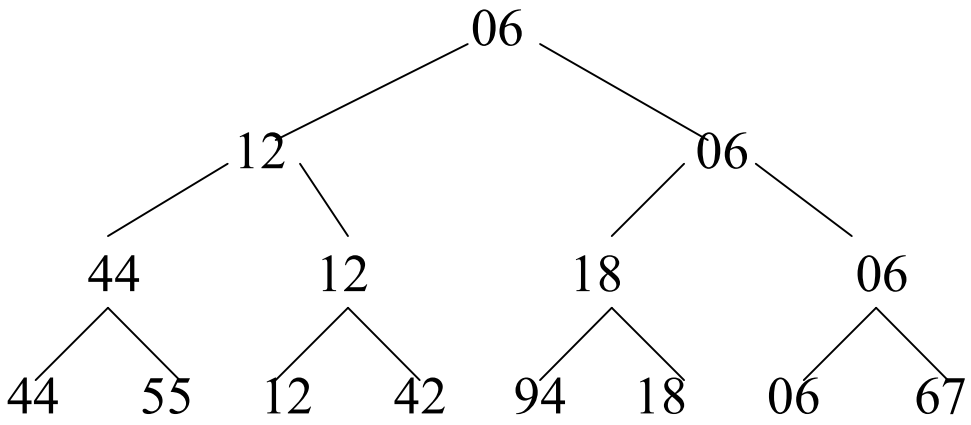
ПУЗЫРЬКОВАЯ СОРТИРОВКА.

```
PROGRAM BS;
VAR I,J,X,N:INTEGER;
   A:ARRAY[0..50] OF INTEGER;
BEGIN
 WRITELN('Введи длину массива');
 READ(N);
 WRITELN('Введи массив');
 FOR I:=1 TO N DO READ(A[I]);
 FOR I:=2 TO N DO  FOR J:=N DOWNTO I DO IF A[J-1]>A[J] THEN BEGIN
  X:=A[J-1];
  A[J-1]:=A[J];
  A[J]:=X
 END;
 WRITELN('Результат:');
 FOR I:=1 TO N DO WRITE(A[I],' ')
END.
```

**Сортировка с помощью дерева**

Метод сортировки с помощью прямого выбора основан на повторяющихся поисках наименьшего ключа среди *n* элементов, среди *n-1* оставшихся элементов и т. д. Как же усовершенствовать упомянутый метод сортировки? Этого можно добиться, действуя согласно следующим этапам сортировки:

1.оставлять после каждого прохода больше информации, чем просто идентификация единственного минимального элемента. Проделав *n-1* сравнений, мы можем построить дерево выбора вроде представленного на рисунке 2.1.



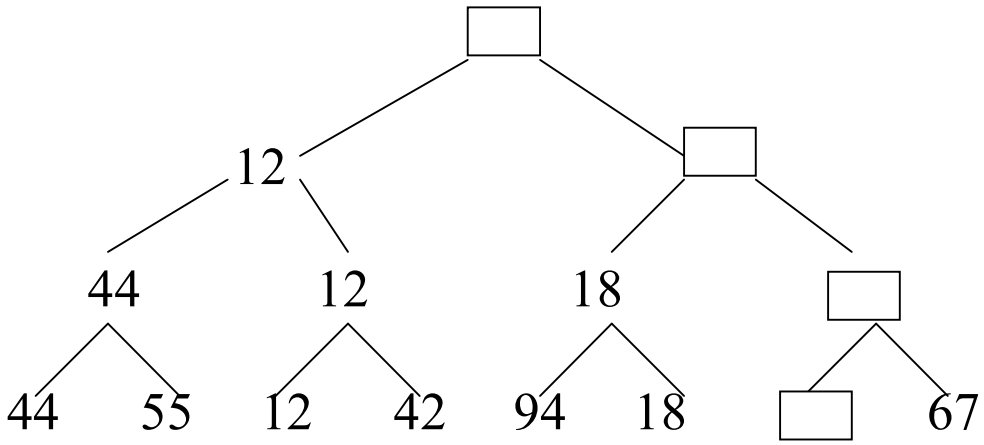

ПОВТОРЯЮЩИЙСЯ ВЫБОР СРЕДИ ДВУХ КЛЮЧЕЙ.

2. спуск вдоль пути, отмеченного наименьшим элементом, и исключение его из дерева путем замены либо на пустой элемент в самом низу, либо на элемент из соседний ветви в промежуточных вершинах (см. рисунки 2.2 и 2.3.).

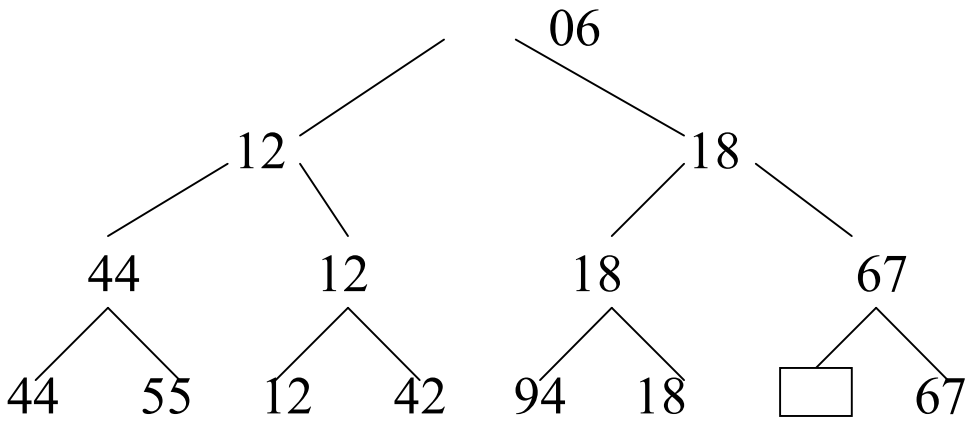
Д. Уилльямсом был изобретен метод ***Heapsort***, в котором было получено существенное улучшение традиционных сортировок с помощью деревьев. Пирамида определяется как последовательность ключей *a[L], a[L+1], ... , a[R],* такая, что

*a[i] <= a[2i] и a[i] <= a[2i+1] для i = L…R/2 .*

Р. Флойдом был предложен некий “лаконичный” способ построения пирамиды на ”*том же месте*”. Здесь *a[1] ... a[n]* – некий массив, причем *a[m]…a[n] (m = [n DIV 2] + 1 )* уже образуют пирамиду, поскольку индексов i и j, удовлетворяющих соотношению *j = 2i (или j = 2i + 1)*, просто не существует.

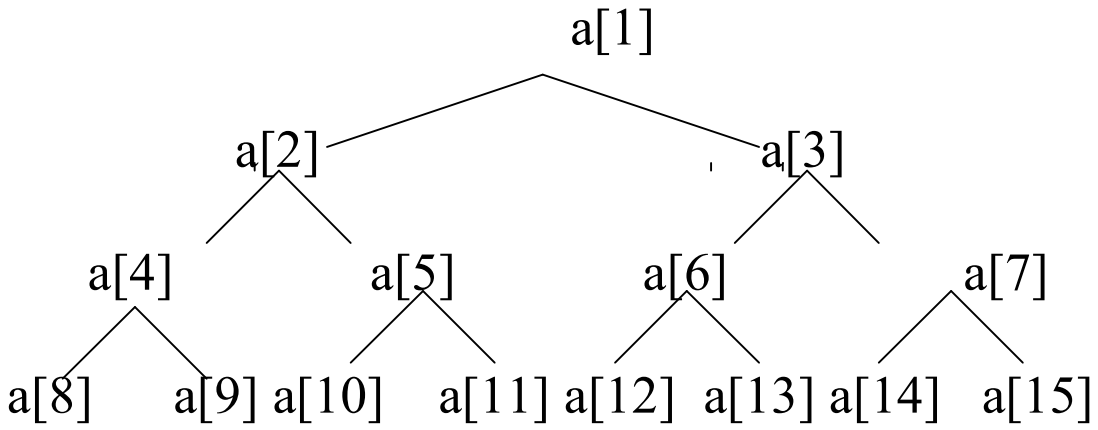



.ВЫБОР НАИМЕНЬШЕГО КЛЮЧА.




ЗАПОЛНЕНИЕ ДЫРОК.

Эти элементы образуют как бы нижний слой соответствующего двоичного дерева (см. рисунок 2.4.), для них никакой упорядоченности не требуется. Теперь пирамида расширяется влево; каждый раз добавляется и сдвигами ставится в надлежащую позицию новый элемент. Таблица 2.6. иллюстрирует этот процесс, а получающаяся пирамида показана на рисунке 2.4.




.МАССИВ, ПРЕДСТАВЛЕННЫЙ В ВИДЕ ДВОИЧНОГО ДЕРЕВА.

Таблица. Построение пирамиды.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 44 | 55 | 12 | 42) | 94 | 18 | 06 | 67 |
| 44 | 55 | 12) | 42 | 94 | 18 | 06 | 67 |
| 44 | 55) | 06 | 42 | 94 | 18 | 12 | 67 |
| 44) | 42 | 06 | 55 | 94 | 18 | 12 | 67 |
| 06 | 42 | 12 | 55 | 94 | 18 | 44 | 67 |

Каждый раз будем брать последнюю компоненту пирамиды (скажем *x*), прятать верхний элемент пирамиды в освободившемся теперь месте, а *x* сдвигать в нужное место. В таблице 2.7. приведены необходимые в этом случае *n-1* шагов.

Таблица 2.7. Примеры процесса сортировки с помощью *Heapsort*.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 06 | 42 | 12 | 55 | 94 | 18 | 44 | 67 |
| 12 | 42 | 18 | 55 | 94 | 67 | 44 | 06 |
| 18 | 42 | 44 | 55 | 94 | 67 | 12 | 06 |
| 42 | 55 | 44 | 67 | 94 | 18 | 12 | 06 |
| 44 | 55 | 94 | 67 | 42 | 18 | 12 | 06 |
| 55 | 67 | 94 | 44 | 42 | 18 | 12 | 06 |
| 67 | 94 | 55 | 44 | 42 | 18 | 12 | 06 |
| 94 | 67 | 55 | 44 | 42 | 18 | 12 | 06 |

Пример из таблицы 2.7. показывает, что получающийся порядок фактически является обратным. Однако это можно легко исправить, и мы получаем программу 2.7.

ПРОГРАММА 2.7. HEARSORT.

```
PROGRAM HS;
VAR I,X,L,N,R:INTEGER;
   A:ARRAY[0..50] OF INTEGER;

PROCEDURE SIFT(L,R: INTEGER);
```

```
VAR
 I,J,X: INTEGER;
BEGIN
 I:=L;
 J:=2*L;
 X:=A[L];
 IF (J<R)AND(A[J]<A[J+1]) THEN INC(J);
 WHILE (J<=R)AND(X<A[J]) DO BEGIN
  A[I]:=A[J];
  A[J]:=X;
  I:=J;
  J:=2*J;
  IF (J<R)AND(A[J]<A[J+1]) THEN INC(J);
 END
END;

BEGIN
 WRITELN('Введи длину  массива');
 READ(N);
 WRITELN('Введи  массив');
 FOR I:=1 TO N DO READ(A[I]);
 L:=(N DIV 2)+1;
 R:=N;
 WHILE L>1 DO BEGIN
  DEC(L);
  SIFT(L,N)
 END;
 WHILE R>1 DO BEGIN
  X:=A[1];
  A[1]:=A[R];
  A[R]:=X;
  DEC(R);
  SIFT(1,R)
 END;
 WRITELN(‘Результат:');
```

```
 FOR I:=1 TO N DO WRITE(A[I],' ')
END.
```

**Сортировка с помощью разделения.**

Этот улучшенный метод сортировки основан на обмене. Это самый лучший из всех известных на данный момент методов сортировки массивов. Его производительность столь впечатляюща, что изобретатель Ч. Хоар назвал этот метод ***быстрой сортировкой*** (***Quicksort***) . В *Quicksort* исходят из того соображения, что для достижения наилучшей эффективности сначала лучше производить перестановки на большие расстояния. Предположим, что у нас есть *n* элементов, расположенных по ключам в обратном порядке. Их можно отсортировать за *n/2* обменов, сначала поменять местами самый левый с самым правым, а затем последовательно сдвигаться с двух сторон. Это возможно в том случае, когда мы знаем, что порядок действительно обратный. Однако полученный при этом алгоритм может оказаться и не удачным, что, например, происходит в случае *n* идентичных ключей: для разделения нужно *n/2* обменов. Этих необязательных обменов можно избежать, если операторы просмотра заменить на такие:

*WHILE a[i] <= x DO i := i + 1 END;*

*WHILE x <= a[i] DO j := j – 1 END*

В этом случае *x* не работает как барьер для двух просмотров. В результате просмотры массива со всеми идентичными ключами приведут к переходу через границы массива.

Наша цель – не только провести разделение на части исходного массива элементов, но и отсортировать его. Будем применять процесс разделения к получившимся двум частям, до тех пор, пока каждая из частей не будет состоять из одного-единственного элемента. Эти действия описываются в программе 2.8.

ПРОГРАММА 2.8. QUICKSORT.

```
PROGRAM QS;
VAR N,I:INTEGER;
   A:ARRAY[0..50] OF INTEGER;

PROCEDURE SORT(L,R: INTEGER);
VAR
```

```
 I,J,X,W: INTEGER;
BEGIN
 I:=L;
 J:=R;
 X:=A[(L+R) DIV 2];
 REPEAT
  WHILE A[I]<X DO INC(I);
  WHILE X<A[J] DO DEC(J);
  IF I<=J THEN BEGIN
   W:=A[I];
   A[I]:=A[J];
   A[J]:=W;
   INC(I);
   DEC(J)
  END
 UNTIL I>J;
 IF L<J THEN SORT(L,J);
 IF I<R THEN SORT(I,R);
END;


BEGIN
 WRITELN('Введи  длину  массива');
 READ(N);
 WRITELN('Введи  массив');
 FOR I:=1 TO N DO READ(A[I]);
 SORT(1, N);
 WRITELN('Результат:');
 FOR I:=1 TO N DO WRITE(A[I],' ')
END.
```

Требования к результатам выполнения работы:

- следует сравнить приведенные выше методы с точки зрения теории эффективности;
- реализовать приведенные выше методы на любом языки программирования.

- провести серию экспериментов по измерению времени, затрачиваемому на сортировку с использованием того или иного метода.
- сравнить полученные результаты с теоретическими выкладками.

***Лабораторная работа 2.*** **Разработка описания и анализ информационной системы**

**Цель работы:**

Описать и проанализировать информационную систему, распределить роли в группе разработчиков.

Составить и проанализировать требования к информационной системе, оформить техническое задание на разработку программного обеспечения.

3) Изучить методологии функционального моделирования IDEF0 и IDEF3.

Лабораторная работа направлена на ознакомление с процессом разработки требований к информационной системе и составления технического задания на разработку программного обеспечения, получение навыков по использованию основных методов формирования и анализа требований; ознакомление с процессом описания информационной системы и получение навыков по использованию основных методов анализа ИС; ознакомление с методологиями функционального моделирования IDEF0 и IDEF3, получение навыков по применению данных методологий для построения функциональных моделей на основании требований к информационной системе.

**Требования к результатам выполнения работы:**

– наличие описания информационной системы;

– наличие заключения о возможности реализации проекта, содержащего рекомендации относительно разработки системы, базовые предложения по объёму требуемого бюджета, числу разработчиков, времени и требуемому программному обеспечению.

– наличие диаграммы идентификации точек зрения и диаграммы иерархии точек зрения;

– наличие сценариев событий (последовательности действий);

– наличие пользовательских требований, четко описывающих будущий функционал системы;

– наличие системных требований, включающих требования к структуре, программному интерфейсу, технологиям разработки, общие требования к системе (надежность, масштабируемость, распределённость, модульность, безопасность, открытость, удобство пользования и т.д.);

– наличие составленного технического задания.

модель должна отражать весь указанный в описании функционал, а также чётко отражать существующие потоки данных и описывать правила их движения;

наличие в модели не менее трёх уровней;

не менее двух уровней декомпозиции в стандарте IDEF0 (контекстная диаграмма + диаграммы A0);

на диаграмме 1-го уровня (A0) не менее 4-х функциональных блоков;

на диаграмме 2-го и далее уровнях должна быть декомпозиция в стандарте IDEF3, на каждой диаграмме не менее 2-х функциональных блоков.

На выполнение работы предусмотрено 6 часов.

Самостоятельная работа 1. Методология объектно-ориентированного моделирования

**Цель работы:** Ознакомление с основными элементами определения, представления, проектирования и моделирования программных систем с помощью языка UML.
Практическая работа направлена на ознакомление с основными элементами определения, представления, проектирования и моделирования программных систем с помощью языка UML, получение навыков по применению данных элементов для построения объектно-ориентированных моделей ИС на основании требований.
Требования к результатам выполнения практической работы:

– модель системы должна содержать: диаграмму вариантов использования; диаграммы взаимодействия для каждого варианта использования; диаграмму классов, позволяющая реализовать весь описанный функционал ИС; объединенную диаграмму компонентов и размещения

– для классов указать стереотипы;

– в зависимости от варианта задания диаграмма размещения должна показывать расположение компонентов в распределенном приложении или связи между встроенным процессором и устройствами.

***Самостоятельная работа 2.*** **Методология управление проектами**

**Цель работы:** Изучение методологии управления проектами. Получение навыков по применению данных методологий для планирования проекта.
Работа направлена на ознакомление с основными понятиями методологии управления проектами, получение навыков по применению данных понятий при построении плана проекта, построения графика работ, распределения исполнителей, управления рисками.
Требования к результатам:

- Построить модель управления проектом. Модель включает:
    - определение всех этапов проекта, зависимых этапов, определение длительности этапов;
    - построение на основе полученных данных сетевой и временной диаграмм;
    - построение диаграммы распределения работников по этапам;

- при определении этапа указывается его название – отражающее суть этапа (например, определение пользовательских требований, проектирование интерфейса и т.д.);
- этапов должно быть не менее 7, срок реализации проекта – пол года с 1.06.2007 по 31.12.2007;
- в проекте задействовано 6 человек персонала (фамилии необходимо придумать), некоторые из них участвуют на нескольких этапах проекта.