机试、软件工程

头文件

文件名	内容	解释
limits.h	#define INT_MAX 2147483647 #define INT_MIN (-INT_MAX - 1)	最大最小值
math.h	pow	
iomanip	setprecision(2)	小数点后位数
algorithm	sort	`Sort(start,end,cmp)` cmp用于规定排序的方法，可不填，默认升序。
string	stod	`stod(num)` 字符串转数字

内存

数组初始化

```
#include <cstring>
char b[6][6];
memset(b,'O',sizeof(b));
```

输入、输出和文件操作

1、运算符重载

返回值和参数都是ostream的引用，ostream等价于cout

```
friend ostream &operator<<(ostream &c, const vector<int> v)
{
    for (auto i : v)
    {
        c << i << " ";
    }
    c << endl;
    return c;
}
```

2、输出

```
float n = 1.32;
float n1 = 1.3275;
float n2 = 123.3;
//输出十六进制
cout << hex << n;
//设置宽度 默认右对齐，填充字符为空格
cout.fill('*');
cout.width(6); //仅生效一次
cout << n;//输出**1.32
//设置精度（多少位有效数字）
cout.precision(3);
cout << n1; //输出1.33
cout << n2; //输出123
//保留小数点后多少位
#include<iomanip>
cout << fixed << setprecision(2) << n1; //输出1.32
```

3、输入

输入字符： a a b c

```
int ch;
```

```
while(ch = cin.get() != EOF){
    //处理字符ch
}
```

记得吃回车啊！否则后面的**字符串输入**会出问题。

```
int n,m;
cin >> n>>m;
cin.get();
while(n--){
    string s;
    getline(cin,s);
}
```

输入字符串：

```
string a;//以字符-分隔开的字符串，这个分隔字符可以是空格
getline(cin,a,'-');
```

输入数字串，不给数字个数或结束条件，例如输入

```
2
0 2 1 3
2 3 1 1 4
```

```
int main(){
    int n;
    cin>>n;
    while(n--){
        vector<int> v;
        int temp;
        int a;
         while (1)
                {
                    cin>>temp;
                    v.push_back(temp);
                    if (getchar() == '\n') break;
                }
```

```
        for(auto i:v){
            cout<<i<<" ";
        }
        cout<<endl;
    }
    return 0;
}
```

4、文件操作

学习下fsream头文件的函数！

```
#include <iostream>
#include <cstdlib>
#include <fstream>
int main() {
    using namespace std;
    ifstream in_file;//要读取的文件
    ofstream out_file;//要写入的文件
    out_file.open(R"(C:\Users\Trafalgar\Desktop\a.txt)",ios::in);//文件不存在则新建，存在则截断（丢弃原内容）
    //文件打开方式选项：
    //  ios::in  = 0x01,  //供读，文件不存在则创建(ifstream默认的打开方式)
    //  ios::out  = 0x02,  //供写，文件不存在则创建，若文件已存在则清空原内容(ofstream默认的打开方式)
    //  ios::ate  = 0x04,  //文件打开时，指针在文件最后。可改变指针的位置，常和in、out联合使用
    //  ios::app  = 0x08,  //供写，文件不存在则创建，若文件已存在则在原文件内容后写入新的内容，指针位置总在最后
    //  ios::trunc  = 0x10,  //在读写前先将文件长度截断为0（默认）
    //  ios::nocreate  = 0x20,  //文件不存在时产生错误，常和in或app联合使用
    //  ios::noreplace = 0x40,  //文件存在时产生错误，常和out联合使用
    //  ios::binary    = 0x80   //二进制格式文件
    double value = 3.2;
    int cnt = 10;
```

```
    while (cnt--)
    {//写
        out_file << value++<<" ";
    }
    out_file << "\nthis is a test\n";
    out_file.close();//记得关闭文件
    in_file.open(R"(C:\Users\Trafalgar\Desktop\a.txt)");
    if (!in_file.is_open()) {
            cout << "could not open file\n";
            exit(EXIT_FAILURE);
    }
    double sum = 0.0;
    in_file >> noskipws;//保留空格和换行符
    in_file >> value;//读
    while (in_file.good()) {
            sum += value;
            in_file >> value;
    }
    if (in_file.eof()) {
            cout << "EOF\n";
    }
    else if (in_file.fail()) {
            cout << "由于数据不匹配无法读取\n";
    }
    else cout << "未知错误\n";
    cout << "sum=" << sum;
    in_file.close();
    return 0;
}
```

5、字符串操作

数字转字符串

l to array

`itoa(i ,num ,10 );`

- i ---- 需要转换成字符串的数字
- num ---- 转换后保存字符串的变量
- 10 ---- 转换数字的基数（即进制，10就是说按10进制转换数字）
- 返回值：指向num这个字符串的指针

或者 `sprintf(x,"%d",n);`

或者`ifstream`？？

```
char x[6];
itoa(n,x,10);
#include <stdio.h>
sprintf(x,"%d",n);
```

逆转字符串：

```
#include<algorithm>
reverse(s.begin(),s.end());
```

字符串排序

自定义降序函数

```
#include <algorithm>
bool com(char a,char b){
    return a>b;
}
string s;
getline(cin,s);
sort(s.begin(),s.begin()+s.size()/2,com);
```

字符串查找、替换

将a中第一次出现的b替换为c

```
#include <string>
string a, b, c;
getline(cin, a);
getline(cin, b);
```

```
getline(cin, c);
auto pos = a.find(b);
if (pos != string::npos)
{
    a.replace(pos,b.size(),c);
    cout<<a;
}
```

数据结构

自定义结构体

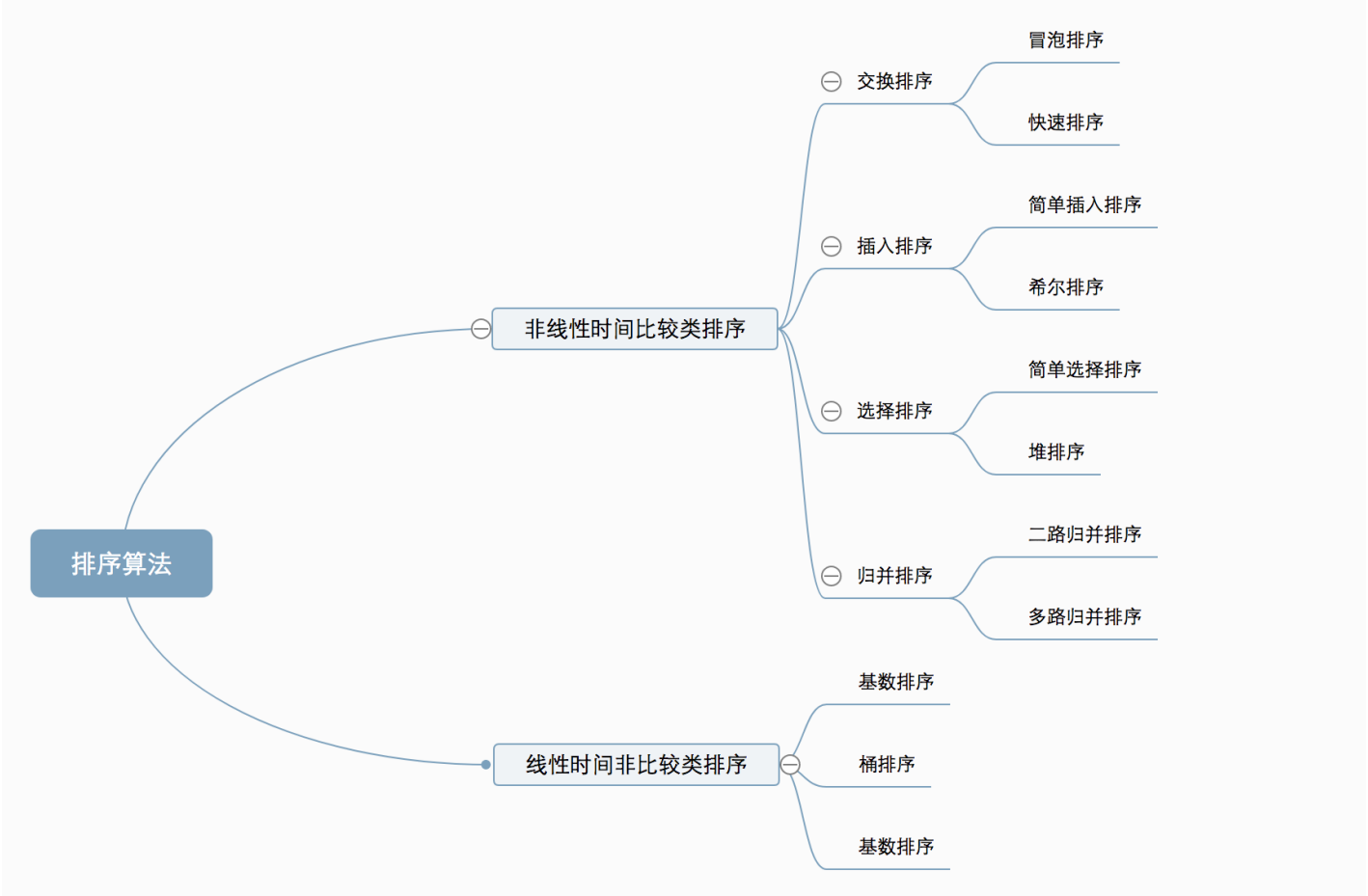
线索二叉树(引线二叉树) 的定义如下:一个二叉树通过如下的方法“穿起来”：所有原本为空的右(孩子)指针改为指向该节点在中序序列中的后继，所有原本为空的左(孩子)指针改为指向该节点的中序序列的前驱

```
//线索二叉树
//二叉树添加了直接指向节点的前驱和后继的指针的二叉树称为线索二叉树。
typedef struct ThreadedBinaryNode{
    int val;
    int rtag,ltag;
    struct ThreadedBinaryNode* lchild,rchild;
} ThreadedBinaryNode,*ThreadedBinaryTree
```

算法

1、排序

```
sort(v.begin(),v.end(),greater<int>());
```



记住：
堆快归，
时间nlogn
空间依次增（1、n、nlogn）
堆快选，不稳定

各种排序法的性能比较				
排序方法	最坏时间复杂度	平均时间复杂度	辅助空间	稳定性
直接插入排序	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
二分插入排序	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
Shell排序	$O(n^{1.3})$	$O(n^{1.3})$	$O(1)$	不稳定
直接选择排序	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	不稳定
冒泡排序	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
快速排序	$O(n^2)$	$O(n\log_2 n)$	$O(\log_2 n)$	不稳定
归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	稳定
基数排序	$O(d*(n+rd))$	$O(d*(n+rd))$	$O(rd+n)$	稳定

插入排序

你手上已经有一堆有序的牌，每次抽牌视为一轮。如果抽到的牌比最右边大则跳过去，小则寻找自己的位置。

```
1  void insert_sort(vector<int> &a, int n){
2      int key, k;
3      for (int i = 1; i < n; i++)
4      {                  //抽n张牌 循环
5          key = a[i]; //现在你抽到一张牌了，大小是key
6          k = i - 1;  //你把牌拿在手牌最右边比较（假设手牌从左往右递增）
7          while (key < a[k] && k >= 0)
8          {                       //你发现每张牌都比key大
9              a[k + 1] = a[k]; //那就把这些牌后移，给key腾出位置
10             k--;                //继续往前找
```

```
        }                       //终于找到了一个比key小的牌了，循环结束
        a[k + 1] = key;         //位置也腾出来了，直接插在它右边
    }
}
```

简单选择排序

你手上已经有一堆有序的牌，每次抽牌视为一轮。每轮你不是抽牌，而是在牌堆中寻找最小的牌。寻找过程中，min_index指针始终指向牌堆中最小的牌。最后交换min_index指向的牌和i指向的牌（手上最右边的牌）这一轮就结束了。

```
void select_sort(vector<int> &v){
    for (int i = 0; i < v.size() - 1; i++)
    {
        int min_index = i;
        for (int j = i + 1; j < v.size(); j++)
        { //从牌堆进行n-1次选择，每次都选出最小的
            if (v[j] < v[min_index])
            {//min_index指针始终指向最小数
                min_index = j;
            }
        }
        swap(v[min_index], v[i]);
    }
}
```

冒泡排序

每轮从左至右，两两比较，将大的往右冒泡到最右边。这样最右边分别是第一大的、第二大的……

```
void bubble_sort(vector<int> &v){
    for (int i = 0; i < v.size(); i++)
    {//i是步数，控制j的变化范围
        for (int j = 0; j < v.size() - i - 1; j++)
        {//注意j的变化范围
            //每次把最大的冒泡到最右边即可
```

```
            if (v[j + 1] < v[j])
                swap(v[j + 1], v[j]);
        }
    }
}
```

快速排序

分治法的典型应用：快速排序的本质就是把基准数大的都放在基准数的右边，把比基准数小的放在基准数的左边，这样就找到了该数据在数组中的正确位置。接下来递归处理左半边和右半边即可。

```
void quick_sort(vector<int> &v, int left, int right){
    //只有一个数时，不需要比较，结束递归
    if (left >= right)
        return;
    //i是low，j是high，tmp初始值为v[i]，所以v[i]最后还原成tmp
    int i = left, j = right, tmp = v[i];
    while (i < j)
    {
        while (j > i and v[j] > tmp)
        { //找到右边第一个小于tmp的
            j--;
        }
        //v[i]已经被记录，可以用于记录v[j]
        //实质就是把比tmp小的挪到左边
        v[i] = v[j];
        while (j > i and v[i] < tmp)
        {
            i++;
        }
        v[j] = v[i];
    }

    v[i] = tmp;
```

```
    //递归处理其他部分
    quick_sort(v, left, i - 1);
    quick_sort(v, i + 1, right);
}
```

归并排序

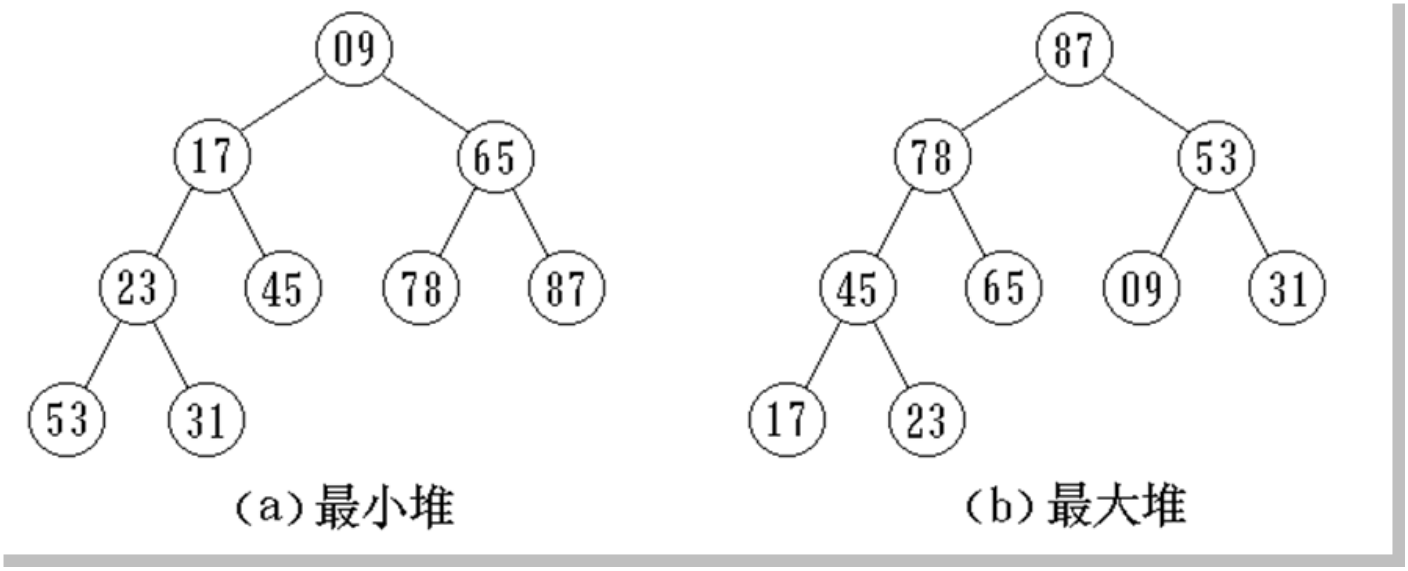
2路归并排序：把长度为n的序列分为两个n/2的子序列，对这两个序列进行归并排序，再合并这两个子序列。有点像快排，但是他们区别在于归并排序分割毫不费力，合并要两两比较着合并；快排分割的时候要用力找出一个分界点，而合并的时候毫不费力。时间复杂度O（nlog2n），空间复杂度O（n）。

```
int* merge(int *a, int a_len, int *b, int b_len) {
        vector<int> res;
        int *t1 = a, *t2 = b;
        while (a_len > 0 && b_len > 0) {
                if (*t1 < *t2) {
                        res.push_back(*t1);
                        t1++;
                        a_len--;
                }
                else {
                        res.push_back(*t2);
                        t2++;
                        b_len--;
                }
        }
        while (a_len)
        {
                res.push_back(*t1);
                t1++;
                a_len--;
        }
        while (b_len) {
```

```
23                  res.push_back(*t2);
24                  t2++;
25                  b_len--;
26          }
27          int *re = new int[res.size()];
28          if (!res.empty()) {
29                  memcpy(re, &res[0], res.size() * sizeof(int));
30          }
31          return re;
32  }
33  int* merge_sort(int *a, int n) {
34          if (n < 2) return a;
35          int mid = n / 2;
36          int *t = a;
37          return merge(merge_sort(a, mid), mid, merge_sort(t + mid, n - mid), n -
    mid);
38  }
```

堆排序

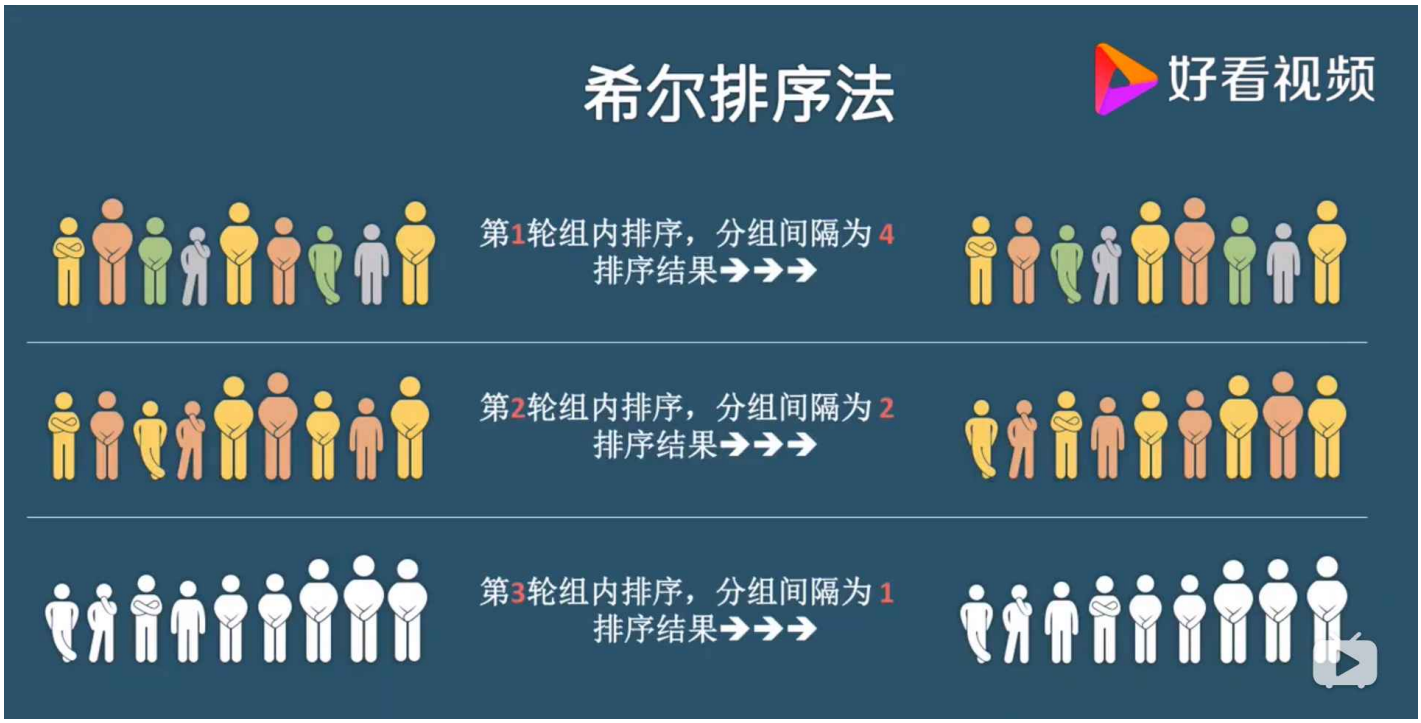
l选择排序的主要操作是比较，要提高它的速度必须减少比较的次数。而实际上如果能**利用以前的比较结果**则可以提高排序速度。



如何由一个无序序列建成一个堆

希尔排序

每轮使用插入排序，效率比一开始就用插入排序要高



2.BFS

```
void bfs(queue<int> qu){
    int size = qu.size();//要记录size
    for(int i=0;i<size;i++){
        int num = qu.front();
        if(num==123456) return;
        qu.pop();
        qu.push(alpha(num));
        qu.push(beta(num));
    }
    cnt++;
    bfs(qu);
}
```

软件工程

类之间的关系有哪些?

继承：类继承另一个类的功能
实现：类实现接口的功能
依赖：A类的某个方法使用到了B类
关联：强依赖关系，B类作为一个属性出现在了A类
聚合：一种特别的关联，公司与个人的关系
组合：强聚合关系，整体与部分的联系更紧密，如汽车与轮胎

软件工程标准步骤?

问题定义
可行性研究
需求分析
总体设计
详细设计
编码和单元测试
综合测试
软件维护

有哪些软件测试分类?

黑盒测试：不考虑软件内部原理，以用户角度测试软件输入输出
白盒测试：知道软件内部工作过程，确定每个分支都能按照预定正常工作
灰盒测试：集合白盒黑盒
冒烟测试：测试软件基本功能，快速
系统测试：验证系统是否满足需求规格的黑盒类测试
性能测试：负载测试和压力测试
安全测试：假扮黑客侵入系统
兼容性测试：不同平台不同环境下的测试

自顶向下和自底向上测试方法的区别?

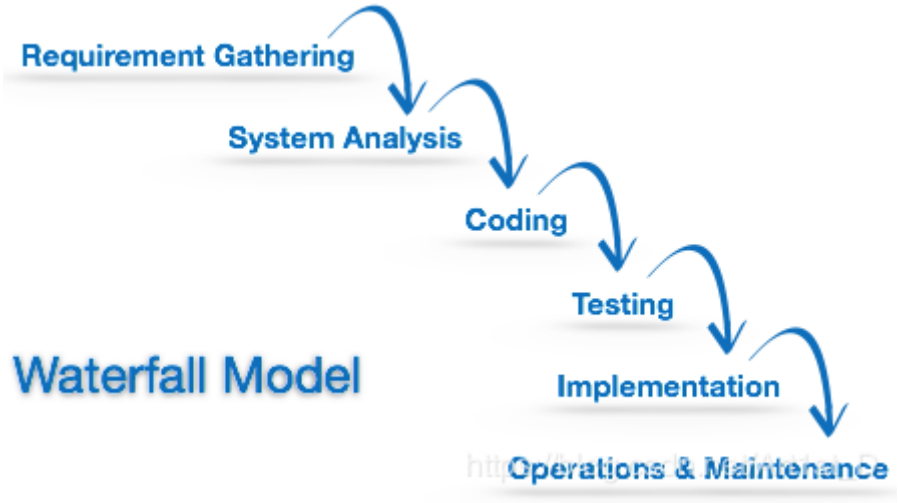
自顶向下：从程序入口主控模块开始，按照系统程序结构，沿着控制层次从上而下测试各模块。方便把握整体结构，早期可发现顶层错误。
自底向上：从最底层模块，即叶子结点开始，按照调用从下而上的测试各模块。最后一个模块提交后才能完整系统测试，某些模块可以提前测试。

软件工程的三要素?

方法、工具、过程。

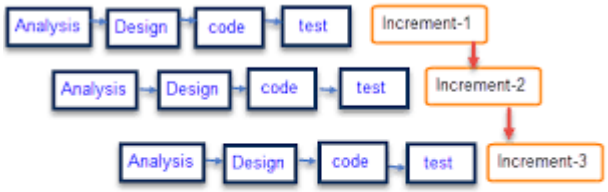
软件工程的主要模型?

1、瀑布模型：前一阶段工作结束才可以进行下一阶段工作，编码时间靠后。基于文档，易于维护，但加大了工作量，由于几乎完全依赖书面的规格说明，容易导致最终产品不能真正满足用户需要。



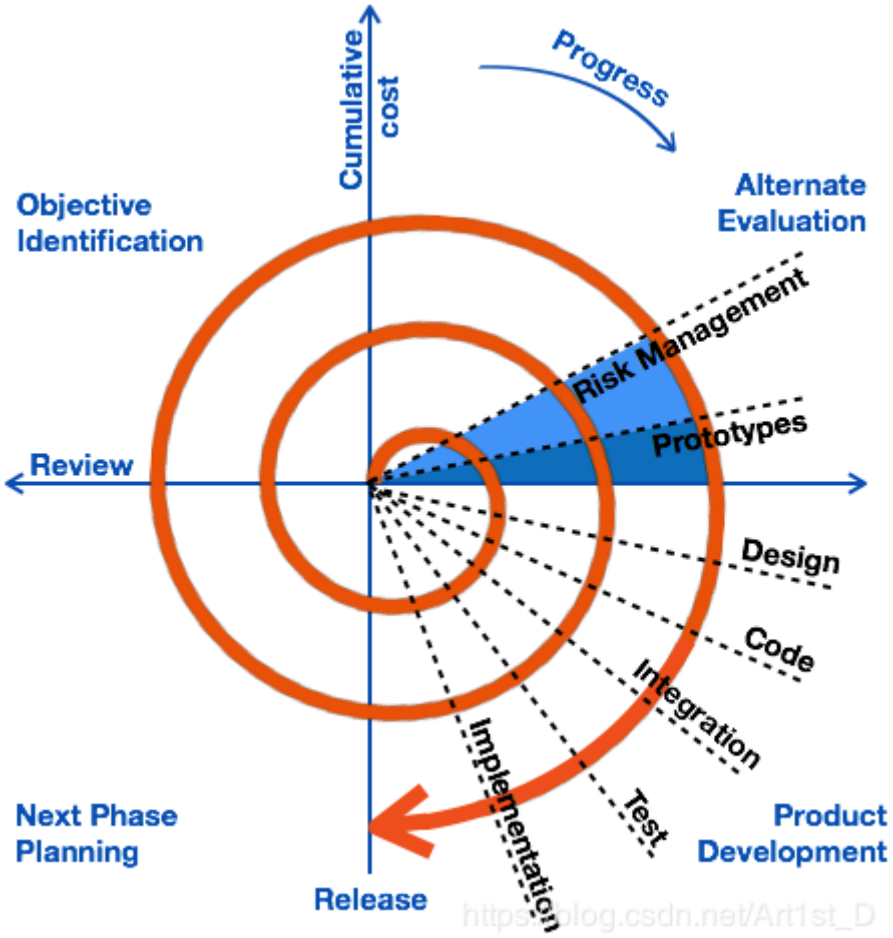
2、快速原型：快速建立可以运行的程序，就是所谓的“原型系统”。完成的功能是最终软件的一个子集。不带反馈环，满足用户真实需求，但会导致系统设计差，难以维护。

3、增量模型：每个阶段不交付完整产品，软件由一系列增量构件组成。降低开发风险，易于维护，但不容易控制整体过程

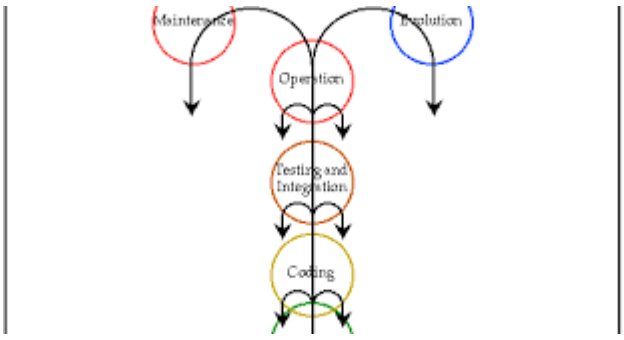


Incremental Model

4、螺旋模型：结合快速原型和瀑布模型，有利于软件重用，减少风险，风险人员需要一定经验。

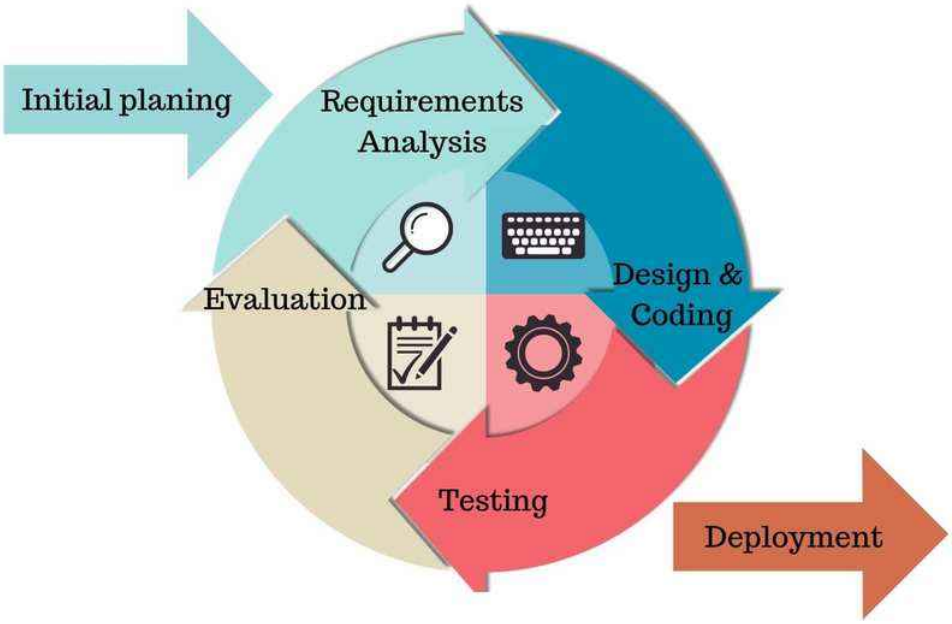


5、喷泉模型：迭代，无缝，节省开发时间。



6、敏捷

敏捷开发是一种以人为核心、迭代、循序渐进的迭代开发方法。 在敏捷开发中，软件项目的构建被切分成多个子项目，各个子项目的成果都经过测试，具备集成和可运行的特征。 换言之，就是把一个大项目分为多个相互联系，但也可独立运行的小项目，并分别完成，在此过程中软件一直处于可使用状态。
迭代开发将一个大任务，分解成多次连续的开发，本质就是逐步改进。开发者先快速发布一个有效但不完美的最简版本，然后不断迭代。每一次迭代都包含规划、设计、编码、测试、评估五个步骤，不断改进产品，添加新功能。通过频繁的发布，以及跟踪对前一次迭代的反馈，最终接近较完善的产品形态。



什么是死代码?

永远不会被执行到的代码。

内聚和耦合

内聚：指一个好的内聚模块内应当尽量只做一件事，描述的是模块内的功能联系。
耦合：各模块之间相互连接的一种度量，耦合强弱取决于模块间接口的复杂程度。
（1）内聚类型低→高：功能内聚、信息内聚、通信内聚、过程内聚、时间内聚、逻辑内聚、偶然内聚
（2）耦合类型高→低：内容耦合、公共耦合、外部耦合、控制耦合、标记耦合、数据耦合、非直接耦合

前沿名词

云计算

云计算是一种按使用量付费的模式，这种模式提供可用的、便捷的、按需的网络访问， 进入可配置的计算资源共享池，这些资源能够被快速提供，只需投入很少的管理工作，或与服务供应商进行很少的交互。

调制

载波：载波是指被来调制以传输信号的波形，一般为正弦波。一般要求正弦载波的频率远远高于调制信号的带宽，否则会发生混叠，使传输信号失真。调制信号就是把低频信号叠加高频信号的栽坡上,这种调制是通过修改原始正弦坡的某种参数来实现的自,修改原始正弦坡参数的依据是调制坡的值.最通有的调制方式是:调幅(AM),调频(FM),调相((FB).信号源百发出的基带信号通过调制（频谱搬移以及幅值变换）后的信号叫做调制信号。

复习打卡

- [x] ~~我复习并且记忆了一遍~~
- [x] ~~我复习并且记忆了一遍~~
- [x] ~~我复习并且记忆了一遍~~
- [x] ~~我复习并且记忆了一遍~~
- [x] ~~我复习并且记忆了一遍~~
- [x] ~~我复习并且记忆了一遍~~
- [x] ~~我复习并且记忆了一遍~~
- [x] ~~我复习并且记忆了一遍~~
- [x] ~~我复习并且记忆了一遍~~
- [x] ~~我复习并且记忆了一遍~~
- [ ] 我复习并且记忆了一遍