信息与软件工程学院
School of Information and Software Engineering

高级软件工程

第三章软件工程师的综合素养

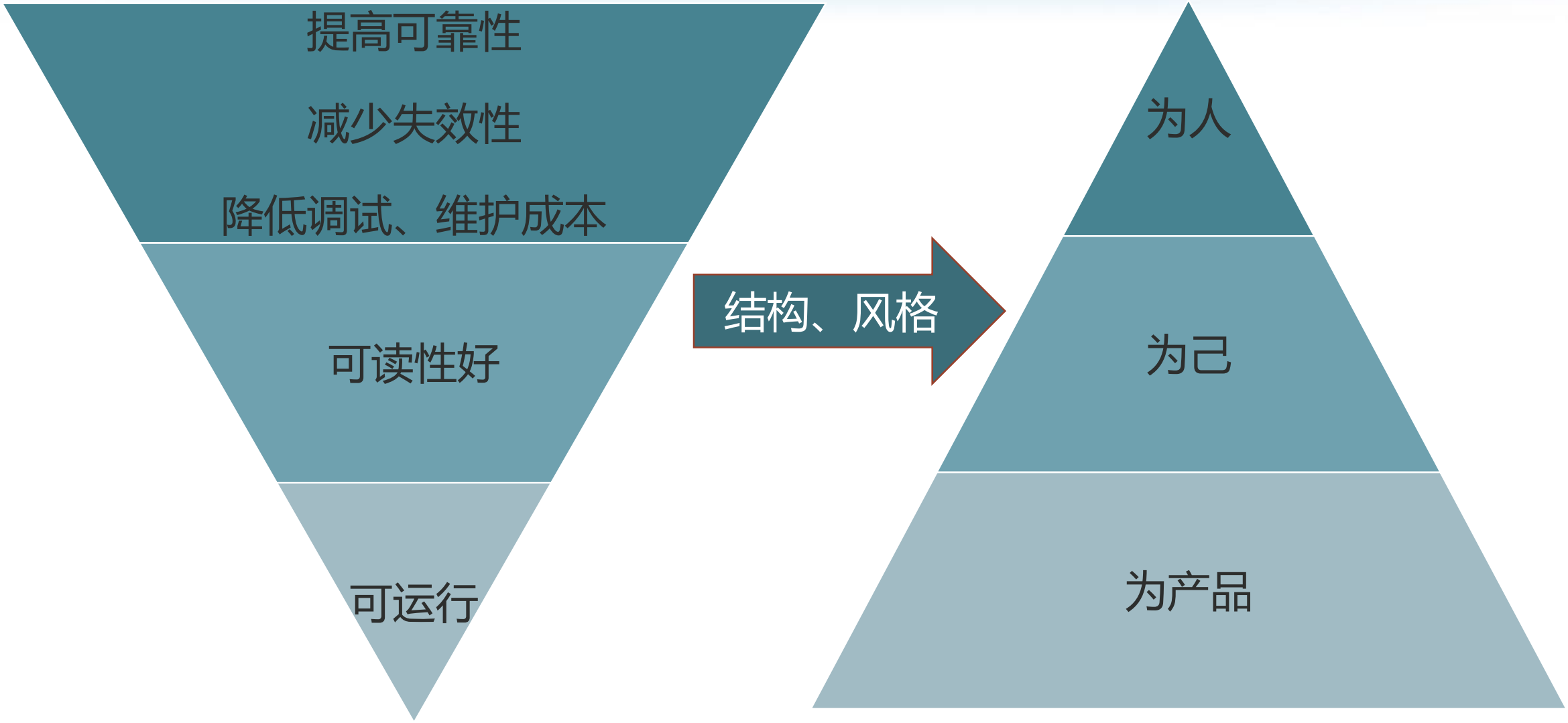
姓名 | 许毅

许毅0421@uestc.edu.cn

2024/1/2

第三章目录

编程规范——重要性



程序设计风格

程序设计风格

1）基本要求

2）可读性要求

3）正确性与容错性要求

4）可移植性要求

5）输入和输出要求

6）重用性要求

程序设计风格—基本要求

- 程序结构清晰且简单易懂，单个函数的行数一般不要超过100行。
- 算法设计应该简单，代码要精简，避免出现垃圾程序。
- 尽量使用标准库函数（类方法）和公共函数（类方法）。
- 最好使用括号以避免二义性。

程序设计风格—可读性要求

注　释

- 程序头,函数头说明；接口说明；子程序清单,有关数据的说明；模块位置；开发历史等
- 主要变量(结构、联合、类或对象):含义的注释。
- 应保持注释与代码完全一致。
- 处理过程的每个阶段和典型算法前都有相关注释说明，但是不要对每条语句注释。

程序设计风格—可读性要求

格 式

- **程序格式清晰：**
- **一行只写一条语句，不要密密麻麻，分不出层次**
- **显示程序的逻辑结构，**利用空格、空行和缩进进行，**缩进量一般为4个字符。**

```
if (A<-17)AND NOT(B<=49)OR Cthen A=A-
1 if A>100 thenA=A*2 endif elseA=A+1
endif
```

写成

```
if  (A < -17)  AND  NOT (B <= 49) OR C
     then A=A-1
          if A>100
             then A=A*2
          endif
     else A=A+1
end if
```

程序设计风格—可读性要求

程序本身

语句力求简单、清晰，不要片面追求效率,程序编写得过于紧凑,使语句复杂化。

例如:

V是一个N×N单位矩阵,

当I≠J时,V(I,J)=0;

当I=J时,V(I,J)=1。

```
for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
            V[i][j] = (i/j) * (j/i)
            else
                V[i][j] = 0;
```

写成

```
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        if (i==j)
            V[i][j] = 1;
```

…

程序设计风格—可读性要求

程序本身-续

- 简单变量的运算速度比下标（数组）变量的运算要快,程序员可把语句：

  X=A[I]+1/A[I]

  写成

  AI= A[I]; X=AI+1/AI

编程时尽可能使用已有的库函数。

尽量用公共过程或子程序代替具有独立功能的重复代码段 。

使用括号清晰地表达算术表达式和逻辑表达式的运算顺序。

尽量使用三种基本控制结构编写程序,使用IF THEN ELSE结构实现分支;使用DO UNTIL或DO WHILE来实现循环。

避免采用过于复杂的条件测试，少用含有“否定”运算符的条件语句，例如:

IF NOT ((CHAR< =’0’) OR (CHAR> =’9’)) THEN ……

改成

IF (CHAR>’0’) AND (CHAR<’9’) THEN ……

程序设计风格—可读性要求

程序本身-续

- 避免使用空的ELSE语句和IF THEN IF语句

```
IF(CHAR>=’A’) THEN
IF(CHAR<=’Z’) THEN
PRINT “This is a letter。”
ELSE//这个语句的配套IF逻辑上不明确
PRINT "This is not a letter。”
```

程序设计风格—可读性要求

程序本身-续

- 避免使用ELSE GOTO和ELSE RETURN结构。
- 避免过多的循环嵌套和条件嵌套。
- 数据结构要有利于程序的简化。
- 模块功能尽可能单一化，模块间的耦合能够清晰可见。利用信息隐蔽确保每一个模块的独立性。
- 对递归定义的数据结构尽量使用递归过程。
- 尽量不要修补结构差的程序,而应重新设计和编码。
- 对太大的程序，要分块编写、测试，后再集成

程序设计风格—可读性要求

数据说明

数据说明的先后次序规范化

每个类型说明中可按如下顺序排列

同一条说明语句中可按字母顺序排列

简单变量类型说明、数组说明、公用数据块说明、文件说明

整型量说明、实型量说明、字符量说明、逻辑量说明

例如：INTEGER cost, length, price, width

程序设计风格—正确性与容错性要求

程序首先是正确，其次是考虑优美和效率。

对所有的用户输入，必须进行合法性和有效性检查。

不要单独进行浮点数的比较。

所有变量在调用前必须被初始化。

改一个错误时可能产生新的错误，因此修改前首先考虑其影响。

单元测试也是编程的一部分，提交联调测试的程序必须通过单元测试。

单元测试时，必须针对类里的每一个public方法进行测试，测试其正确的输入，是否得到正确的输出；错误的输入是否有容错处理。

程序设计风格—可移植性要求

应当尽量使用语言的标准部分，避免使用第三方提供的接口，以确保程序不受具体的运行环境影响，和平台无关。

对数据库的操作，使用符合语言规范的标准接口类例如JDBC，除非程序是运行于特定的环境下，并且有很高的性能优化方面的要求。

程序中涉及到的数据库定义和操纵语句，尽量使用标准 SQL 数据类型和 SQL 语句

程序设计风格—输入和输出要求

√ 任何程序都会有输入输出，输入输出的方式应当尽量方便用户的使用。在需求分析和设计阶段就应确定基本的输入输出风格，要避免因设计不当带来操作和理解的麻烦。

√ 对所有的输入数据进行检验，从而识别错误的输入，以保证每个数据的有效性。

√ 检查输入项各种重要组合的合理性，必要时报告输入状态信息。

√ 输入的步骤和操作尽可能简单，并且要保持简单的输入格式。

√ 有些输入信息应提供缺省值。

√ 输入一批数据时，最好使用输入结束标志，而不要由用户指定输入数据数目。

√ 在以交互式方式进行输入时，要显示提示信息、选择项和取值范围，便于操作。同时，在输入数据的过程和输入数据结束时，也要在屏幕上给出状态信息。

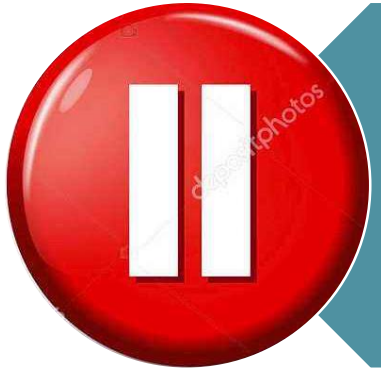
√当程序设计语言对输入格式有严格要求时，应保持输入格式与输入语句的要求一致。

√ 给所有的输出加上注解信息。

√ 按照用户的要求设计输出报表格式。

程序设计风格—重用性要求

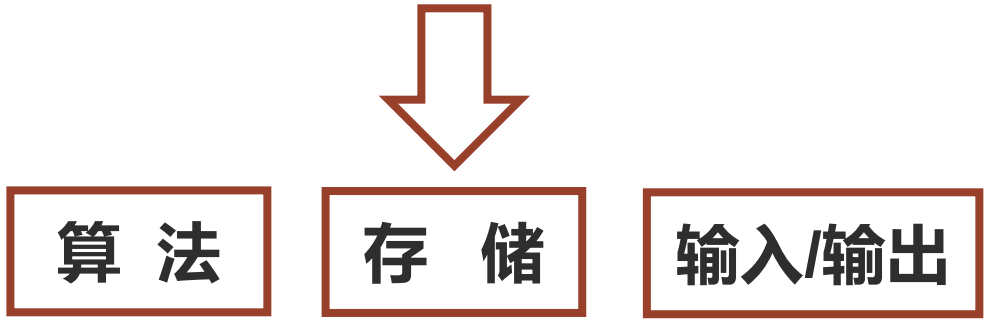
可重复使用的、功能相对独立的算法或接口。应该考虑封装成公共的控件或类。



相对固定和独立的程序实现方式和过程，应考虑做成程序模板，增强对程序实现方式的复用

程序的效率

程序的效率：程序的执行速度及程序所需占用的存储空间。



程序编码是最后提高运行速度和节省存储的机会，因此在此阶段不能不考虑程序的效率。

程序效率的几条准则

效率是一个性能要求，应当在需求分析阶段给出。软件效率以需求为准，不应以人力所及为准。

好的设计可以提高效率。

程序的效率与程序的简单性相关。

一般说来，任何对效率无重要改善，且对程序的简单性、可读性和正确性不利的程序设计方法都是不可取的。

效率的影响因素——算法

源程序的效率与详细设计阶段确定的算法的效率直接有关。在设计翻译转换代码，算法效率反映为程序的执行速度和存储容量的要求。

设计向程序转换过程中的指导原则

- **简化**：在编程序前，尽可能化简有关的算术表达式和逻辑表达式；
- **检查**：仔细检查算法中的嵌套的循环，尽可能将某些语句或表达式移到循环外面；
- **选用**：选用等效的高效率算法；
- **采用**：采用“快速”的算术运算；尽量采用整数算术表达式和布尔表达式；
- **避免**：尽量避免使用多维数组；尽量避免使用指针和复杂的表；不要混淆数据类型，避免在表达式中出现类型混杂；

效率的影响因素——存储器

大中型计算机系统,存储限制不是主要问题

内存分页功能的虚拟存储管理。效率与系统的分页功能相关。

结构化程序设计，将程序功能合理分块，模块（群）体积与页容量相匹配，减少调度。

在微型计算机系统，存储容量对软件设计和编码的制约很大。

选择可生成较短目标代码且存储压缩性能优良的编译程序，甚至汇编程序。

提高存储器效率的关键是程序的简单性。

第三章目录

不懂开发怎么做代码审查？

- 静态分析是指在不执行的情况下对代码进行评估的过程。包括：
  - 类型检查
  - 风格检查
  - 程序理解
  - BUG查找
  - 安全审查

循环语句的效率

```
for (row=0; row<100; row++)
{
    for ( col=0; col<5; col++ )
    {
        sum = sum + a[row][col];
    }
}
```

在多重循环中，如果有可能，应当将最长的循环放在最内层，最短的循环放在最外层，以减少CPU跨切循环层的次数：

```
for (col=0; col<5; col++ )
{
        for (row=0; row<100; row++)
        {
                sum = sum + a[row][col];
        }
}
```

指针的使用

```
   _UC  *puc_card_config_tab;
… …
   Get_Config_Table( AMP_CPM_CARD_CONFIG_TABLE,
                     &ul_card_config_num,
                     &puc_card_config_tab,
                     use_which_data_area
                   );
… …
   b_middle_data_ok  =
    generate_trans_middle_data_from_original_data(
                        puc_card_config_tab,
                        Ul_card_config_num)
…. …
```

'switch' Statement Should Include a Default Case

```
1. switch (formatType)
    2. {
    3.     case 1:
    4.         formatStr = "yyyyMMddHHmmss";
    5.         break;
    6.     case 2:
    7.         formatStr = "yyyy'-'MM'-'dd HH:mm:ss";
    8.         break;
    9.     case 3:
    10.         formatStr = "yyyy.MM.dd HH:mm:ss";
    11.         break;
    12.     case 4:
    13.         formatStr = "yyyy'年'MM'月'dd HH:mm:ss";
    14.         break;
    15. }
```

根据编码规范，每个switch流程控制语句都必须带一个default分支，以保证逻辑分支的完整性。
如果没有第15~16行的default代码，代码审查将给出警告。

```
1. switch (formatType)
2. {
3.     case 1:
4.         formatStr = "yyyyMMddHHmmss";
5.         break;
6.     case 2:
7.         formatStr = "yyyy'-'MM'-'dd HH:mm:ss";
8.         break;
9.     case 3:
10.        formatStr = "yyyy.MM.dd HH:mm:ss";
11.        break;
12.    case 4:
13.        formatStr = "yyyy'年'MM'月'dd HH:mm:ss";
14.        break;
15.    default:
16.        formatStr = "yyyy'-'MM'-'dd HH:mm:ss";
17. }
```

Complex Assignment

```
1. int i = 0;
2. int j = 0;
3. int k = 0;
4. int l = 0;
5. i *= ++j;
6.
7. k = j = 10;
8.
9. l = j += 15;
10.
11. i = j++ +20;
12.
13. i = (j = 25) + 30;
```

往往有些程序员热衷于将语言的语法发挥到极致，以资其对该语言语法精通的凭据。如果是为了练习语法、理解语法，无可厚非。但如果在需要充分协作沟通的软件项目中，简洁明了，清晰易懂将会受到推崇，晦涩难懂的语句将会受到奚落。

故此，大部分的软件公司的规范都对语句的**精简明了**提出了要求。

流程控制中存在不可到达的语句：死代码

有些流程控制由于测试条件恒为false，则流程中的程序无法到达。

```
1. int[] arr = new int[size];
2. if (arr == null) //由于arr不为空，则该测试逻辑不可能通过，程序无法进入该程序块中
3. {
4.     return null;
5. }
```

添加()清晰化复杂的表达式

写复杂的表达式时不应过度依赖运算操作符的计算优先顺序，而应养成使用“()”的好习惯，当一个逻辑表达式由多个逻辑运算组成时，应该用“()”划分不同的部分。

```
1. boolean a, b, c;
2. ...
3. if (a || b && c) //应该替换成if ((a || b) && c)
4. {
5. ...
6. }
```

代码规范例

```
1. i++;  j++;
2
3. if (val < 0)
4.     return;
5. while (val >= 10)
6.    val /= 10;
7.
8. void func () {
9.     long var = 0x0001111l;
10. }
```

不应将多行语句写在同一行代码中。

代码块应以“{}”框起来，虽然增长了代码，但代码**结构性**更强。

声明长整型使用大写的“L”类型指定符，而非小写的“l”，因为后者和数字1相似。

命名规范

- 骆驼（Camel）
- 帕斯卡（Pascal）
- 匈牙利

“匈牙利”法

- 该命名规则的主要思想是“在变量和函数名中加入前缀以增进人们对程序的理解”。
- 例如所有的字符变量均以ch为前缀，若是指针变量则追加前缀p。如果一个变量由ppch开头，则表明它是指向字符指针的指针。

“匈牙利”法的缺点

- “匈牙利”法最大的缺点是烦琐，例如

  int i, j, k;

  float x, y, z;
- 倘若采用“匈牙利”命名规则，则应当写成

  int iI, iJ, ik; // 前缀 i 表示int 类型

  float fX, fY, fZ; // 前缀 f 表示float 类型
- 如此烦琐的程序会让绝大多数程序员无法忍受。

骆驼式命名法

- 正如它的名称所表示的那样，是指混合使用大小写字母来构成变量和函数的名字。例如，下面是分别用骆驼式命名法和下划线法命名的同一个函数：

```
printEmployeePaychecks();

print_employee_paychecks();
```

第一个函数名使用了骆驼式命名法 -- 函数名中的每一个逻辑断点都有一个大写字母来标记；第二个函数名使用了下划线法----函数名中的每一个逻辑断点都有一个下划线来标记。

骆驼式命名法近年来越来越流行了，在许多新的函数库和Microsoft Windows这样的环境中，它使用得当相多。另一方面，下划线法是c出现后开始流行起来的，在许多旧的程序和UNIX这样的环境中，它的使用非常普遍。

帕斯卡（Pascal）

- 与骆驼命名法类似。只不过骆驼命名法是首字母小写，而帕斯卡命名法是首字母大写，如：
- public void DisplayInfo();

  string UserName;

  二者都是采用了帕斯卡命名法

- 在C#中，以帕斯卡命名法和骆驼命名法居多。

练习：读代码，找错误

```
1:
2:  char *report( short m, short n, char *p )
3:      {
4:      int result;
5:      char *temp;
6:      long nm;
7:      int i, k, kk;
8:      char name[11] = "Joe Jakeson";
9:
10:       nm = n * m;
11:       temp = p == "" ? "null" : p;
12:       for( i = 0; i < m; i++ )
13:           { k++; kk = i; }
14:       if( k == 1 ) result = nm;
15:       else if( kk > 0 ) result = 1;
16:       else if( kk < 0 ) result = -1;
17:       if( m == result ) return temp;
18:       else return name;
19:       }
```

PC-Lint

- C/C++代码静态分析工具
- C/C++语言的语法拥有其它语言所没有的灵活性，这种灵活性带来了代码效率的提升，但相应也使得代码编写具有很大的随意性，另外C/C++编译器不进行强制类型检查，也不做任何边界检查，这就增加了代码中存在隐患的可能性。
- PC-lint 在全球拥有广泛的客户群，许多大型的软件开发组织都把PC-Lint 检查作为代码走查的第一道工序。PC-Lint不仅能够对程序进行全局分析，识别没有被适当检验的数组下标，报告未被初始化的变量，警告使用空指针以及冗余的代码，还能够有效地帮你提出许多程序在空间利用、运行效率上的改进点。

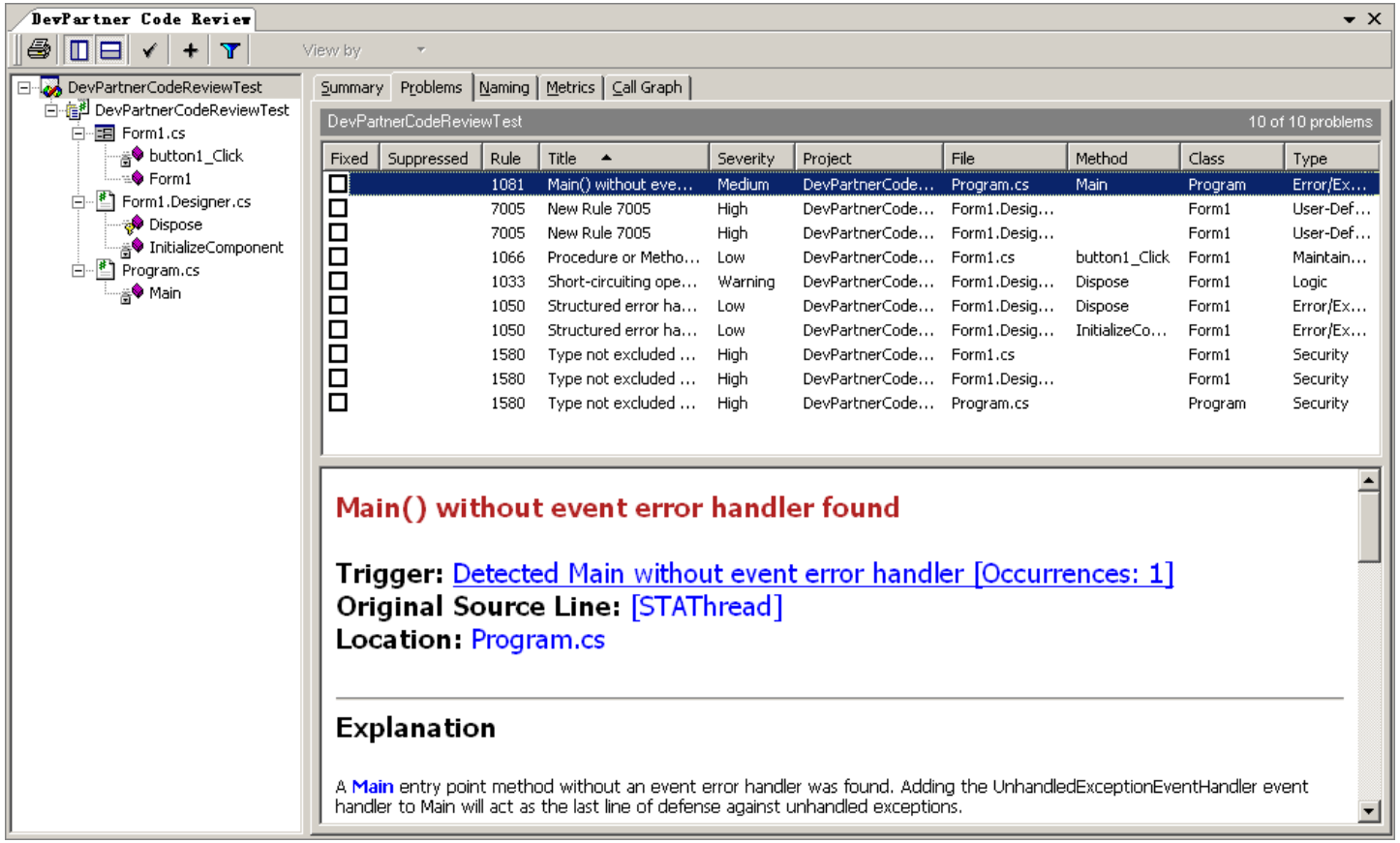
FxCop

- FxCop是一款专门用于分析.NET托管代码程序集的工具。它能检查.NET代码是否满足Microsoft.NET Framework 设计规范。它使用反射、MSIL解析等技术来分析程序集，可以检查超过200个缺陷，包括库设计、本地化、命名规范、性能、安全等方面的内容。

DevPartner

- DevPartner Studio Professional Edition是一款与Visual Studio紧密结合的测试工具。它能帮助检测和诊断各种.NET代码问题，包括代码评审（Code Review）、错误检测、性能分析、覆盖率分析、内存分析等功能。

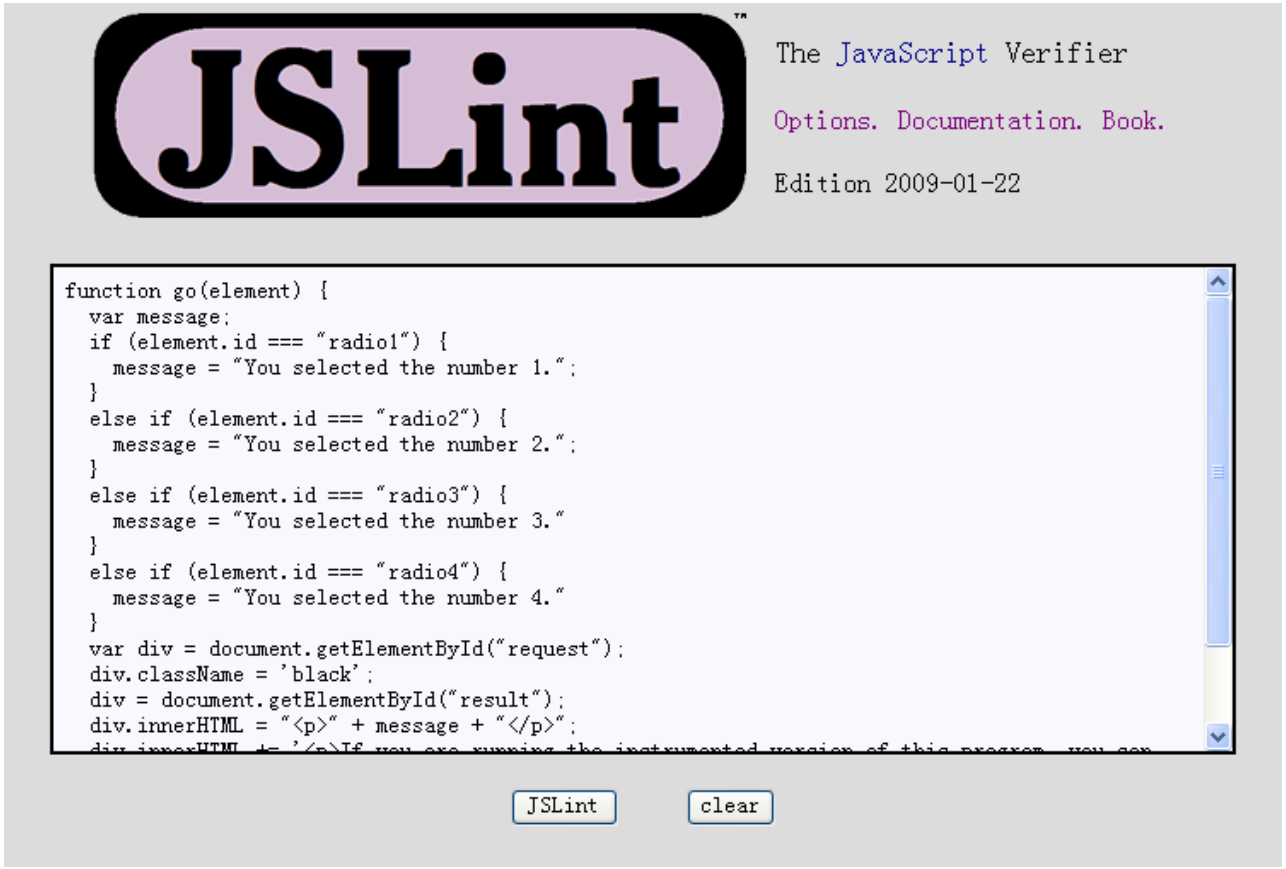
DevPartner的Code Review功能



在Rule Manager中编辑代码规则

- DevPartner通过扫描和匹配代码中符合正则表达式的代码来判断代码是否违反了规则，因此可以说正则表达式是制定规则、识别代码缺陷的关键。

JavaScript代码检查工具-JSLint



第三章目录

版本管理的意义

目的：标识、控制和追踪软件开发和实施过程中产生的各种软件产品版本。

适用范围：适用于软件源代码、产品版本的管理。通常内部使用，不对外发布。

作用：主要是开发者自己对产品进行测试，检查产品是否存在缺陷、错误，验证产品功能与说明书、用户手册是否一致。

常见软件版本

Alpha版

内部测试版，用作内部测试

Beta版

外部测试版，典型性用户的外部测试

Demo版

演示版，演示部分功能，为发售造势

Enhanced版

增强版或加强版，增加了新功能、新游戏场景的正式发售版本。

Free版

自由版，没有版权，免费给大家使用的版本。

Full Version版

完全版，最终正式发售的版本。

Shareware版

共享版，为了吸引客户，带有限制的版本。

Release版等。

发行版，可从Internet上免费下载

常见版本命名规则

GNU版	主版本号 . 子版本号 [. 修正版本号 [. 编译版本号]]	示例 : 1.2.1, 2.0, 5.0.0 build-13124
Windows版	主版本号 . 子版本号 [修正版本号 [. 编译版本号]]	示例: 1.21, 2.0
Net.Framework版	主版本号.子版本号[.编译版本号[.修正版本号]]	示例： 4, 4.5, 4.6, 4.7

目前主要有三种版本控制工具：CVS、SVN、Git.

第三章目录

软件工程职业道德规范和实践要求的简明版摘要

IEEE-CS 和ACM软件工程道德和职业实践联合工作组认为，软件工程是一种有益和受人尊敬的职业，应该遵循一定的道德和职业规范。

软件工程师应当承担起公众健康、安全和利益的承诺目标，坚持公众利益、客户和雇主、产品专业标准、职业判断独立性等原则。

软件工程师应当参与终生学习，不断提高自己的专业素养和职业道德水平。

软件工程职业道德规范和实践要求的完整版详解规范意向和细节

规范意向：软件工程师应当以公众利益为目标，尤其是在适当的情况下，应当负责、公益、安全、公正、合作、教育。

规范细节：项目风险应确认记录并报告。
软件/文档涉及社会问题应报告。

雇主或客户原则：不接受外部工作，避免利益冲突，遵循专业标准。

- 产品软件工程师原则：保证质量、成本、进度，有适当目标，解决道德、法律问题。通过教育、实践保证胜任。

软件工程师应坚持的原则

公众利益原则

软件工程师应以公众利益为首要目标，在涉及公众利益的问题上保持公正性和透明度，同时积极履行社会责任。

客户和雇主原则

软件工程师应当尊重客户的意愿和需求，按照客户需求提供高质量的软件服务。同时，他们应当遵守职业道德规范，尊重雇主的利益和形象。

专业标准原则

软件工程师应当遵循行业认可的相关标准和规范，不断提高自己的专业素养和技能水平。

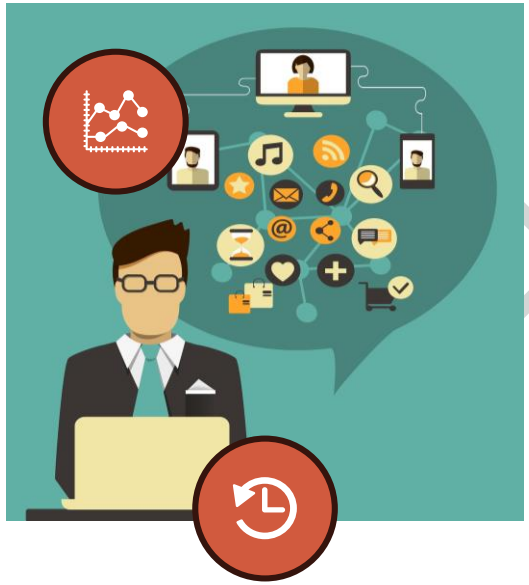
职业判断独立性原则

软件工程师应当保持独立性和客观性，不参与有悖于职业道德的行为，并对自己的职业判断负责。

软件工程师的基本责任

遵循原则

软件工程师应当坚持职业道德原则，严格遵守本规范的要求，不断提高自己的道德水平。



尊重他人

软件工程师应当尊重他人的意见和权利，积极合作、诚信待人，保持良好的人际关系。

考虑影响

软件工程师应当充分考虑自己的行为对公众、客户、雇主和其他相关方的利益和形象的影响。

周密思考

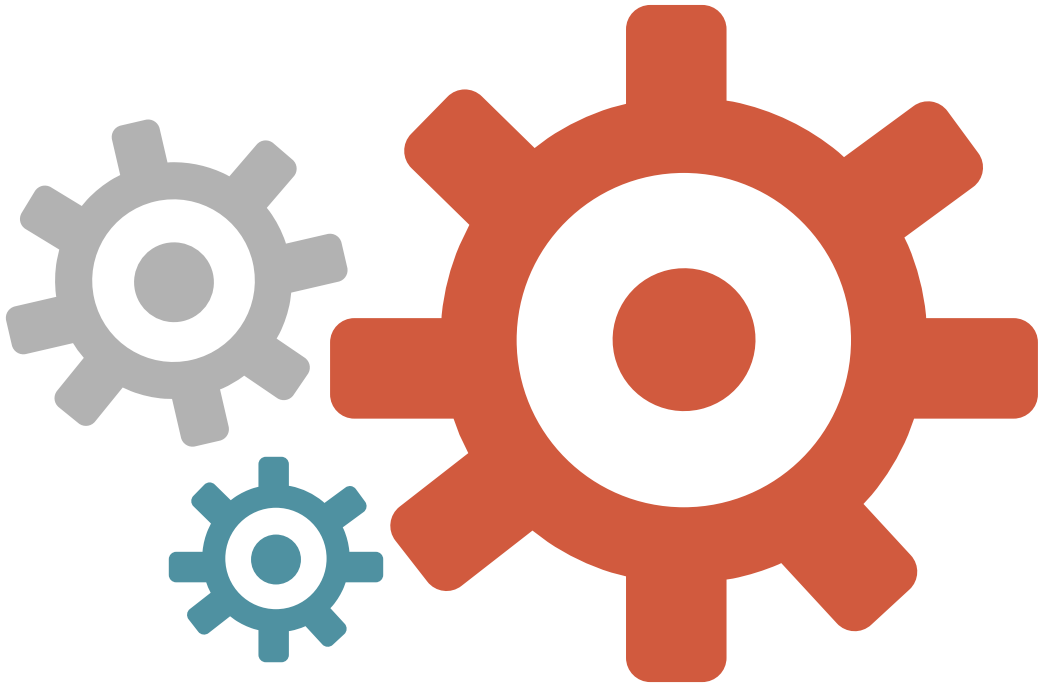
软件工程师应当充分考虑各种因素，制定合理的方案并选择合适的技术路线，确保项目的成功实施。

软件工程师应履行其实践承诺

01 软件工程师应明确并理解他们的实践承诺，以便将软件工程职业作为一项有益和受人尊敬的职业。

02 软件工程师应明确自己的道德原则和职业责任，并以此指导自己的行为。

03 软件工程师应积极参与终生学习，不断提高自己的专业素养和实践能力。



软件工程师的职业目标

01

软件工程师的职业目标是致力于提高软件工程实践的道德水平和专业性。

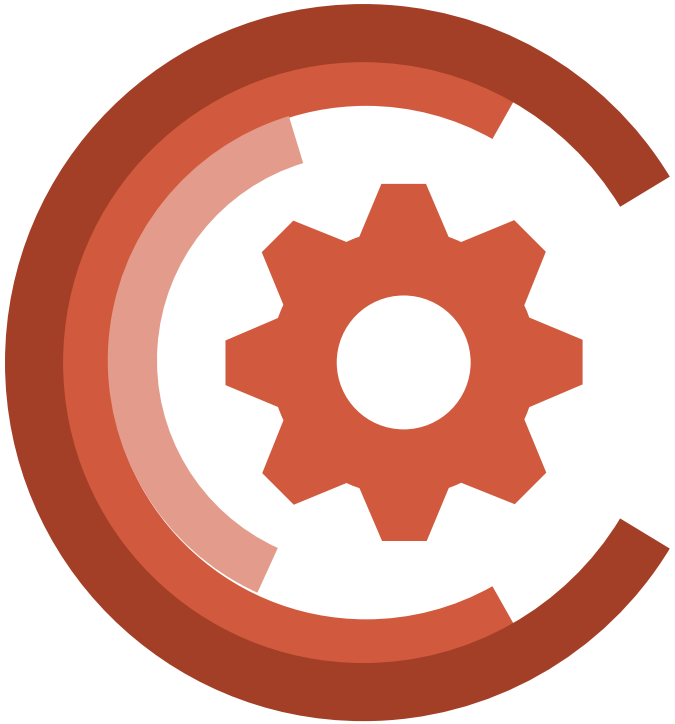
02

软件工程师应通过遵守职业道德规范和实践要求，确保所开发的软件对公众健康、安全和利益没有负面影响。

03

软件工程师应以实践承诺和职业目标为指导，推动软件工程职业的发展和进步。

软件工程师应当以公众利益为目标



01

软件工程师应当关注公众利益，并将公众利益放在首位。他们应当通过自己的专业知识和技能，为公众提供高质量、安全、可靠和实用的软件服务。

02

软件工程师在开发软件时，应当考虑到社会、环境和可持续发展等因素，从而更好地为公众服务。例如，他们可以开发环保的软件，提高公共卫生服务的软件等。

03

软件工程师应当积极参与社会公益活动，为社会做出贡献。例如，他们可以参与公益捐赠、志愿者活动等，为社会发展做出自己的贡献。

在适当的情况下软件工程师应当遵循的原则

在适当的情况下，软件工程师应当遵循职业道德规范和法律要求，从而保持良好的职业操守和道德水准。

在面临利益冲突时，软件工程师应当保持公正性和客观性，不接受贿赂、不偏私，并尊重他人的权利和利益。

软件工程师应当积极参与职业培训和学习，不断提高自己的专业素养和技能水平，以更好地为公众服务。

软件工程师的保密责任

保守商业机密

软件工程师应妥善保管涉及公司或客户商业机密的资料和信息，不得在未经授权的情况下向任何人透露或披露，也不得在个人博客或社交媒体上公开或讨论。

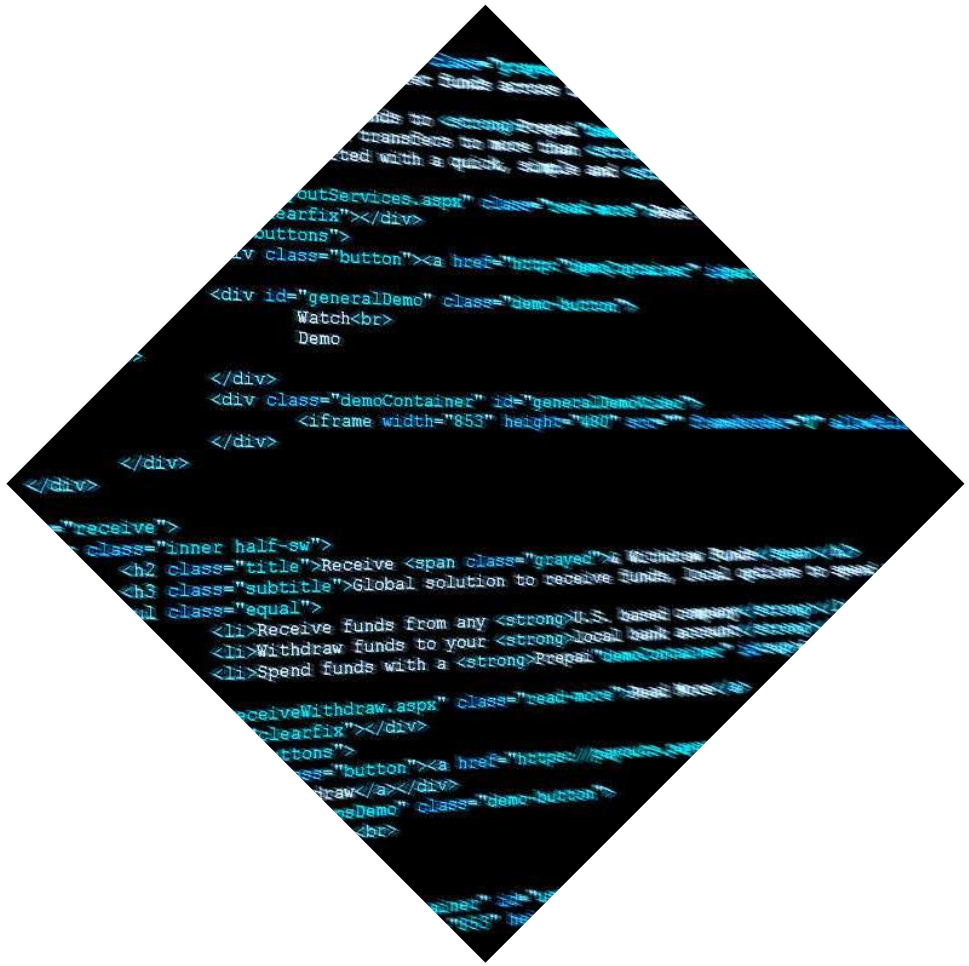
保护个人隐私

软件工程师应尊重他人的隐私，不泄露、传播或讨论他人的私人信息，包括但不限于姓名、地址、电话号码、身份证号码等。

不泄露源代码

软件工程师在工作中应妥善保管源代码，不得在未经授权的情况下向任何人透露或披露，也不得在个人博客或社交媒体上公开或讨论。

软件工程师应遵循的标准版权归属



尊重版权

软件工程师应尊重他人的版权，不盗用他人的成果或未经授权使用他人的知识产权。

获得授权使用版权

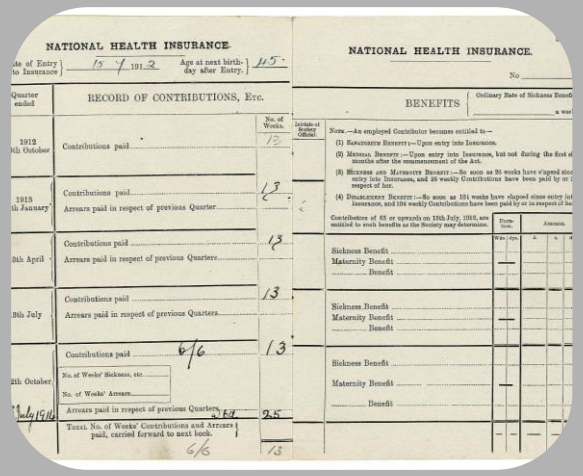
如果需要使用他人的版权保护的成果或知识产权，应获得相应的授权或付费使用，避免侵权行为。

配合版权持有人维权

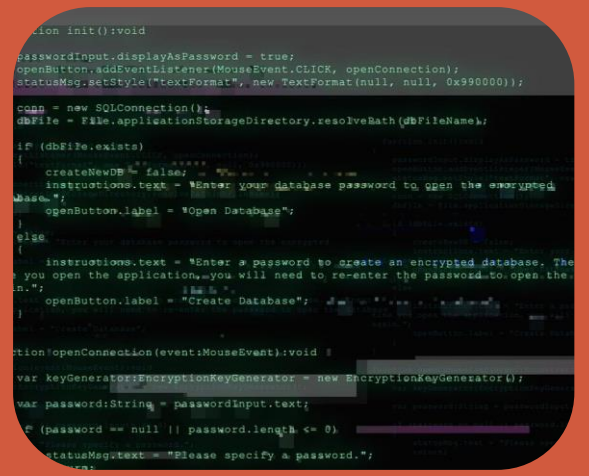
如果他人指控侵权行为，软件工程师应积极配合版权持有人进行维权处理，并承担相应的法律责任。

项目风险应确认记录并报告

应识别、评估并记录项目中的所有风险。这些风险包括但不限于技术风险、商业风险、竞争风险、人际风险等。



应建立一套完善的风险管理流程，包括风险识别、评估、应对和监控等环节。



对于无法预测或控制的风险，应向项目组或上级领导及时报告，并一起制定相应的应对策略。

涉及社会问题的软件/文档应报告

1

如果在项目过程中发现涉及社会问题的相关内容，如性别歧视、种族歧视、暴力等，应向项目组或上级领导报告。

2

应建立一套完善的社会问题审查机制，确保所开发的软件/文档符合社会伦理和法律法规的要求。

3

如果发现有涉及社会问题的内容，应立即向相关主管部门或专业机构报告，并积极配合调查处理。

不接受外部工作，避免利益冲突

不参与可能影响公正性和独立性的项目

软件工程师不应为可能影响其公正性和独立性的雇主或客户工作。这包括但不限于参与可能对个人或团体产生不利影响的项目。

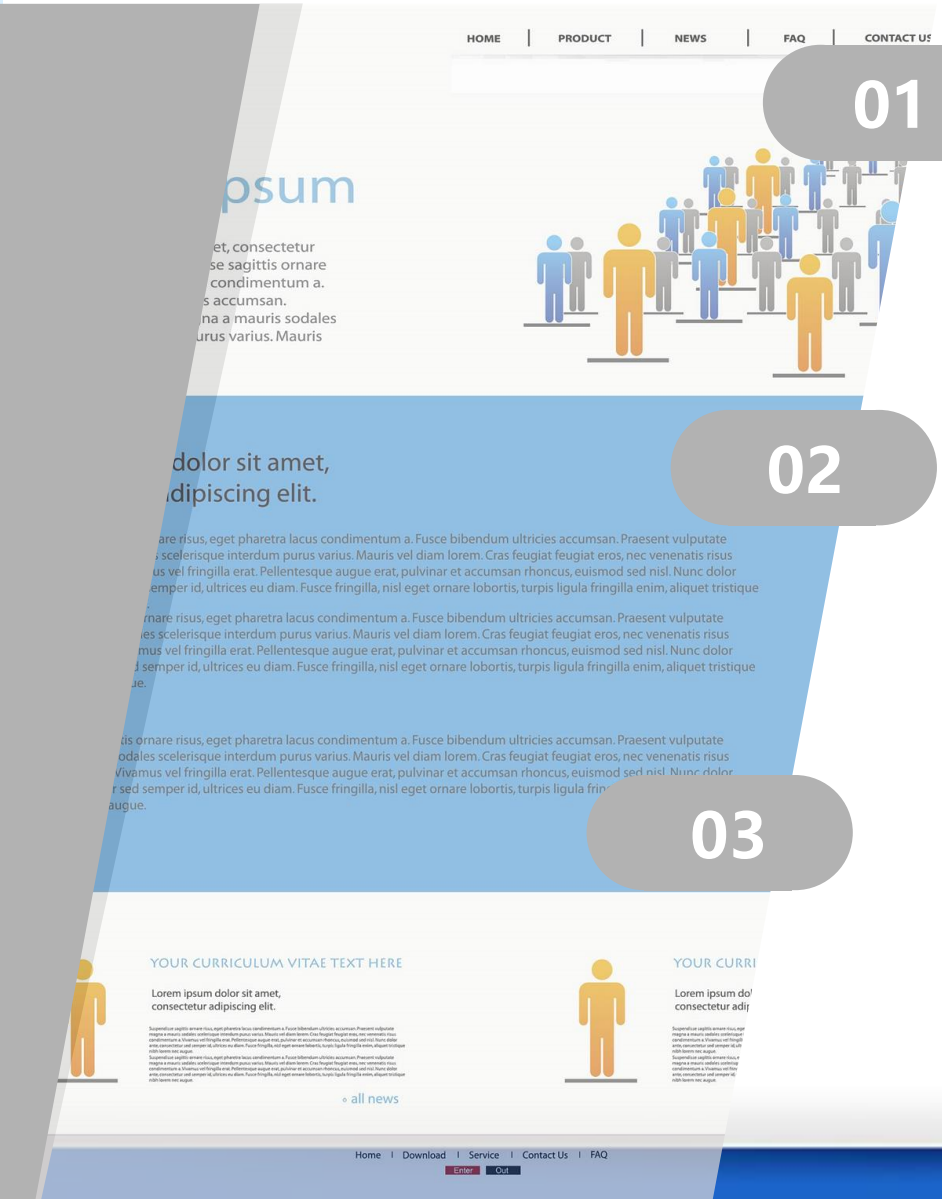
避免利益冲突

软件工程师应当避免任何可能影响其公正性和独立性的利益冲突。这包括但不限于与雇主或客户存在利益关系、与项目有直接或间接关联的个人或实体有密切联系。

遵循职业道德规范

软件工程师应当遵循职业道德规范，保持职业操守和道德标准，以公正、诚实、客观的态度从事工作。

遵循专业标准



01

了解专业标准

软件工程师应当了解与所从事的软件工程领域相关的一系列专业标准。这些标准可能包括但不限于IEEE、ACM、ISO等组织发布的相关标准。

02

遵循专业标准

软件工程师应当按照专业标准进行工作，以确保软件工程的有益性和受人尊敬。这包括但不限于遵循软件需求分析、规格说明、设计、开发、测试和维护的标准。

03

提升专业能力

软件工程师应当不断提升自己的专业能力和知识水平，以适应新技术和行业发展的需求。同时，他们应当积极参与行业交流和学习，不断提高自己的专业素养。

解决道德、法律问题

尊重他人

软件工程师应尊重他人的劳动成果和隐私，不进行恶意竞争和诋毁行为。

保护知识产权

软件工程师应尊重知识产权，包括版权、专利等，以便其他专业人员有机会和能力进行交流和学习。

遵守法律法规

软件工程师应遵守国家法律法规，尊重社会道德和伦理准则，以确保产品的合法性和安全性。

通过教育、实践保证胜任

参与终生学习

软件工程师应积极参与终生学习计划，以保持其专业知识和技能更新，以适应不断变化的技术和市场环境。

提供优质教育

软件工程师应以教育为其职业发展提供支持和帮助，通过培训、讲座和实践经验等方式向其他专业人员传授知识和技能。

实践经验

软件工程师应通过实践经验来提高其专业水平和能力，积极参与项目开发和实施过程，以积累经验并提高技能水平。

尊重隐私，使用合法数据

01

软件工程师应当理解并尊重隐私的重要性，在使用他人数据时，应当遵守相关法律法规和道德准则。

02

软件工程师应当只收集和使用了必要的数据，并确保这些数据的收集和处理方式符合相关法律法规的要求。

03

软件工程师应当避免在软件中存储敏感数据，如用户隐私信息等，以减少数据泄露的风险。

维护数据完整性，保持职业态度

01

软件工程师应当采取必要的措施来确保数据的完整性，以防止数据被篡改或损坏。

02

软件工程师应当对数据进行审慎处理，并对数据进行必要的过滤和清洗，以消除数据中的干扰和噪声。

03

软件工程师应当保持职业态度，对职业道德规范保持敬畏之心，避免出现不合规行为。

计算机职业道德

根据计算机信息系统及计算机网络发展过程中出现过的种种案例，以及保障每一个法人权益的要求，美国计算机伦理协会总结、归纳了以下计算机职业道德规范，称为“计算机伦理十戒”：

计算机伦理十诫

(1) 不应该用计算机去伤害他人。

(2) 不应该影响他人的计算机工作。

(3) 不应该到他人的计算机里去窥探。

(4) 不应该用计算机去偷窃。

(5) 不应该用计算机去做假证明。

(6) 不应该复制或利用没有购买的软件。

(7) 不应该未经他人许可的情况下使用他人的计算机资源。

(8) 不应该剽窃他人的精神作品。

(9) 应该注意你正在编写的程序和你正在设计系统的社会效应。

(10) 应该始终注意，你使用计算机是在进一步加强你对同胞的理解和尊敬。