2022 知识点练习->上午->12.专业英语答案解析

1. ADBCB

解析1：

本题考查英语专业知识。

译文：一个是软件工程师，另一个是DevOps工程师。最大的不同在于他们的关注点。软件工程师关注计算机软件如何满足客户的需求，而DevOps工程师关注的范围更广，包括软件开发、软件如何部署以及在软件持续运行时通过云提供操作支持。

软件工程师根据人们的安全性和功能需求创建计算机程序供人们使用。DevOps工程师也处理计算机应用程序，但将构建、部署和操作作为一个连续的自动匹配过程进行管理。软件工程师通常与企业的运营部门分开工作。他们创建业务客户所需的软件，然后监控其软件产品的性能，以确定是否需要升级或是否需要更大的改进。DevOps工程师与业务的运营部门合作，并管理工作流，以集成软件，使其与自动化流程顺利运行。这两种职业都需要计算机编程语言的知识。

选项翻译：

A、focus 关注点 B、process 过程 C、goal 目标 D、function作用

A、developing发展中的 B、deploying 使展开，部署 C、training 训练、培养 D、operational 操作的

A、developed 先进的，发达的 B、functional 功能的 C、constructed 构件 D、secure 保护

A、 single 单一的 B、 whole 完整的，全部的 C、continuous 连续的 D、independent 自主的，不相干的

A、develop 发展 B、integrate 整合 C、analyse 分析 D、maintain 维持

2. CBDDA

解析1：

设计面向对象的软件很难，而设计（可复用的）面向对象软件就更难了。

你必须找到合适的（相关的）对象，以适当的粒度将它们划分为类，定义类接口和继承，并在它们之间建立关键关系。

你的设计应该针对眼前的问题，但（一般）足以解决未来的问题和要求。

你也要避免重新设计，或者至少最小化它。

有经验的面向对象设计师会告诉你，一个可重用和灵活的设计是很难第一次就"正确"的。

在设计完成之前，他们通常会尝试多次重复使用，每次都是这样。

然而，有经验的面向对象设计师确实能做出好的设计。

同时，新的设计师们对可用的选项感到不安（第三题选项不知所措的意思），他们倾向于使用以前使用的非面向对象技术。

电影要花很长时间才能了解好的面向对象的设计。

经验丰富的设计师显然知道一些缺乏经验的东西。它是什么？有一件事专家设计师知道不能做的就是从第一原理解决每一个问题。相反，他们选择了过去对他们有效的解决方案。

当他们找到一个好的（解决方案）。他们一次又一次地使用它。这样的经验是他们成为专家的部分原因。

因此，您将在许多面向对象系统中找到第五题个类和通信对象的模式。

A. runnable可运行的

B.right对的

C. reusable可复用的

D. pertinent中肯的，相关的

A. clear清除

B. general总则

C. personalized个性化

D.customized定制

A. excited兴奋

B. shocken 震惊

C. surprised惊讶于

D. overwhelmed不知所措

A.tool工具

B. component组成部分

C. system系统

D. solution解决方案

A.recurring循环

B.right是吗

C.experienced经验丰富

D.past过去

3. BACDA

无论系统或应用程序设计、构造和测试得多么完善，错误或故障总是会不可避免地出现。一旦一个系统实现了，这个系统就进入运行和支持阶段。

系统支持是对用户的不间断的技术支持以及改正错误、遗漏或者可能产生的新需求所需的维护。在信息系统可以被支持之前，它必须首先投入运行。系统运行是信息系统的业务过程和应用程序逐日的、逐周的、逐月的和逐年的执行。

不像系统分析、设计和实现那样，系统支持不能明显地分解成一些系统支持项目必须执行的任务阶段。相反，系统支持包括4个进行中的活动，这些活动是程序维护、系统恢复、技术支持和系统改进。每个活动都是一类系统支持项目，这些活动由已经实现的系统遇到的特定问题、事件或机会触发。

A、 设计
B、 实施
C、 调查
D、 分析

A、支持
B、测试
C、实施
D、建造

A、结构
B、维护
C、执行
D、实施

A、划分
B、形成
C、组成
D、分解

A、触发
B、导致
C、引起
D、产生

4. ABABC

解析1：

你们正在开发一个全方位的企业应用系统。它必须支持各种不同的客户机，包括桌面浏览器。移动浏览器和本地移动应用程序。应用程序还可以向客户公开第三方的API。它还可以通过web服务或消息代理与其他应用程序集成。

应用程序通过执行业务逻辑、访问数据库、与其他系统交换消息以及返回HTML /JSON/XML 响应 来处理请求（HTTP请求和消息）。它有一些逻辑组件对应于应用程序的不同功能区域。

那么这个应用程序有什么样的部署体系结构呢?

将应用程序的体系结构定义为一组松散耦合的协作服务集合，对应于Scale Cube的y轴伸缩。

每个服务具有如下特征：

易于维护和测试——支持快速和频繁的开发和部署。

与其他服务的松散耦合——便团队能够独立工作（大部分时间在其服务器上），而不会受到对其他服务的更改的影响，也不会影响其他服务。

独立部署——允许团队部署他们的服务，而不必与其他团队协调。

能够被一个小团队开发，避免了大团队的高级沟通负责人，这对于高生产力是至关重要的。

服务通信使用HTTP/REST等同步原协议或AMQP等同步协议。服务可以彼此独立地开发和部署。每个服务都有自己的数据库，以便与其他服务解耦。服务之间的数据一致性是使用某种特定的模式来维护的。

5. CABAD
6. ABCDC

解析1： 项目工作手册不是单独的一篇文档，它是对项目必须产出的一系列文档进行组织的一种结果。
项目的所有文档都必须是该结构的一部分。这包括目标，外部规范说明，接口规范，技术标准，内部规范和管理备忘录（备忘录）。
技术说明几乎是必不可少的。如果某人就硬件和软件的某部分，去查看一系列相关的用户手册。他发现的不仅仅是思路，而且还有能追溯到最早备忘录的许多文字和章节，这些备忘录对产品提出建议或者解释设计。对于技术作者而言,文章的剪裁粘贴与钢笔一样有用。基于上述理由，再加上"未来产品"的质量手册将诞生于"今天产品"的备忘录，所以正确的文档结构非常重要。事先将项目工作手册设计好，能保证文档的结构本身是规范的，而不是杂乱无章的。另外，有了文档结构，后来书写的文字就可以放置在合适的章节中。使用项目手册的第二个原因是控制信息发布。控制信息发布并不是为了限制信息，而是确保信息能到达所有需要它的人的手中项目手册的第一步是对所有的备忘录编号，从而每个工作人员可以通过标题列表来检索是否有他所需要的信息。还有一种更好的组织方法，就是使用树状的索引结构。而且如果需要的话，可以使用树结构中的子树来维护发布列表。

7. DCACB

解析1： 创建一个清晰的项目，地图是一个重要的第一步。它让你识别风险，明确目标，并确定项目是否有意义。唯一比发布计划更重要的是不要太认真。
发布计划是为你的Web项目创建一个游戏计划，概述你认为你想要得到什么样的网站。该计划指导将网站的内容、设计元素和功能发布给公众、合作伙伴或内部。它还估算了项目将花费多长时间和资源。这个计划不是一个功能性的说明书——详细地定义了项目或者形成一个可以提交到银行的预算。基本上，你使用发布计划来对项目的可行性和价值进行初步的安全检查。发布计划是有用的路线图，但不要认为它们是州际公路系统的指南。相反，把它们看作是早期探险家使用的地图——一半是传闻和猜测。一半是希望和期望。
有一个项目走向的地图总是一个好主意。

8. BACBD

解析1： 语义网络的发展是一步一步的，每一步都建立在之前的基础之上。选择这种方法的现实理由就是因为很容易对一小步达成一致，而如果想要一步到位就难得多。通常，很多研究组织都是从不同方向考虑的，这种思想的竞争的方式是科学进步的驱动力。然而，从工程的角度来说是需要进行标准化的。因此，如果大多数研究者同意某个观点不同意另一个的时候，改正观点是有意义的。这样，即使再宏大的研究努力也会失败，可能会有局部的积极效果。
一旦一个标准被建立，许多组织和企业都会采纳，而不是等待并查看其他研究线是否会获得成果。语义网络的性质就是让企业和单个用户必须构建工具，添加内容并使用。我们不会等着整个语义网络被物化——因为实现它的全部内容需要再过十年时间（当然是按照今天所设想）。

9. ACBDA
10. CABDC

解析1： 软件实体的尺寸比任何其他人类构造更复杂，因为没有两个部分相同（至少在语句级上）。如果是，我们将两个相似的部分分成一个，一个（1），开放或关闭。在这方面，软件系统与计算机，建筑物或汽车有着深刻的区别，其中重复的元素很多。
数字电脑本身比大多数人理解的很多情况都要更复杂。这使得构思，描述和测试它们非常复杂。软件系统比计算机更多（2）数量级。
同样地，软件实体的放大不仅仅是较大尺寸的相同元素的重复；必然增加不同要素的数量。在大多数情况下，这些元素以（3）的方式彼此相互作用，并且整体的复杂性比线性增加更多。
软件的复杂性是（4）的属性，而不是偶然的。因此，消除其复杂性的软件实体的描述往往会抽象出其本质。数学和物理科学通过构建复杂现象的简化模型，从模型中导出属性，并通过实验验证这些属性，在三个世纪以来取得了长足的进步。这是因为模型中被忽略的复杂性（5）不是现象的基本属性。当复杂性是本质时，它不起作用。
开发软件产品的许多经典问题源于这一重要的复杂性，其非线性随着尺寸而增加。不仅技术问题，管理问题也来自于复杂性。

11. DBCAB

解析1： 在物理安全和信息安全领域，访问控制是选择性地限制访问某个地方或其他资源。访问行为可能意味着消耗，进入或使用。授权访问资源称为授权。
访问控制机制介于用户（或代表用户执行的进程）和系统资源之间。资源如应用程序、操作系统、防火墙、路由器、文件和数据库。系统必须先验证（验证）寻求访问权限的用户。通常，认证功能确定用户是否能被允许访问该系统。然后，访问控制功能确定该用户的特定请求的访问是否被允许。一个安全管理员维护一个授权数据库，该数据库指定该用户允许哪些资源的访问类型。访问控制功能查询此数据库以确定是否授权访问。审计功能监控和保存用户对系统资源的访问记录。
实际上，很多组件可以协同共享访问控制功能。所有的操作系统至少有一个基本的，在许多情况下是一个非常强大的访问控制组件。附加安全软件包可以添加到操作系统的本地安全控制功能。特定的应用程序或实用程序，如数据库管理系统，还包括访问控制功能。外部设备（如防火墙）也可以提供访问控制服务。

12. DABCA

解析1： 在这个世界上，似乎我们有太多的事情要去做，有太多的事情要去思考，那么需要做的最后一件事就是必须学习新事物。
而用例恰恰可以解决带有需求的问题：如果具有严格声明的需求，则很难描述事件的步骤和序列。
简单地说，用例可以将事件序列的说明放在一起，引导系统完成有用的任务。正如听起来一样简单，这很重要。在面对很多需求的时候，通常不太可能理解需求的作者真正想要系统做什么。在前面的例子中，通过指定特定行为发生的时间和条件，用例减少了需求的不确定性。这样的话，行为的顺序就可以当作是一种需求。用例特别适用于捕捉这类需求。尽管听起来可能很简单，但事实情况是由于常规的需求捕捉方法所侧重的是声明需求和"应该怎么样"的陈述，因此完全无法捕捉系统行为的动态方面。用例是一种简单而有效的表达系统行为的方式，使用这种方式所有参与者都很容易理解。
但是与任何事物一样，用例也存在自己的问题，在用例非常有用的同时，人们也可能误用它，结果就产生了比原来更为糟糕的问题。因此重点在于：如何有效地使用用例，而又不会产生比原来更严重的问题。

13. ADCAB

解析1：
为什么要有正式的文档?
首先，将决策写下来是关键的。只有写出后差距才能出现，矛盾才能突出。写的过程是需求成百上千的小决策的过程，这些的存在将清楚的、准确的政策从模糊的政策中区分出来。
其次，文档能够作为同其他人的沟通渠道。项目经理将会不断感到惊奇的是，许多理应被普遍认同的测量，全然不为团队的一些成员所知。既然他的基本工作是使每个人在一个方向上前进，他的主要工作就是交流，而不是决策制定，他的文档能很好的减轻这个负担。
最后，项目经理的文档给他提供了一个数据库和检验表。通过定期回顾他能知道自己所处的位置，并看到为需要对重点改变什么或方向作什么变动。
项目经理的任务是制定计划，并根据计划实现。但是只有书面计划是精确和可以沟通的。计划中包括了时间、地点、人物、做什么、资金。这些少量的关键文档封装了一些项目经理的工作。如果一开始就认识到它们的普遍性和重要性，那么就可以将文档作为工具友好地利用起来，而不会让它成为令人厌烦的繁重任务。通过遵循文档开展工作，项目经理能更清晰和快速地设定自己的方向。

14. CABCD

解析1： 大多数工程项目需要团队完成。虽然有些小规模的硬件或软件产品可以由个人完成，但是现代系统的规模大、复杂性高以及开发周期短的极高需求，使得一个人完成大多工程工作已经不再现实。系统开发是一个团队活动，团队的效率很大程度上决定工程的质量。
开发团队经常表现的像是棒球队或篮球队，即使棒球队或篮球队可能有多种不同专长，但是所有的队员都朝着一个目标努力。然而在系统维护和挣钱团队，工程师们的工作就像摔跤和田径队一样经常相对独立。
团队不仅仅是一群人碰巧在一起工作，团队工作需要实践，涉及到多种特殊的技能。团队需要共同的过程，需要达成一致的目标，需要有效地指导和领导。尽管指导和领导这样的团队的方法是众所周知的，但是它们并不明显。

15. ADABC

解析1：
云计算是用来描述各种计算概念的短语，包括大量计算机通过网络相互连接以实现分布计算，意思是同时在很多互联的计算机上运行程序或应用的能力。 云的架构分为基础设施层、平台层和应用层三层。基础设施层由虚拟计算、存储和网络资源构成。平台层用于一组软件资源重复使用的通用目的。应用层由一组所需的软件模块构成即软件即服务（SaaS）。基础设施层作为构建平台层的基础。相反，平台层是应用层的基础，为SaaS应用实现应用层。

16. DABAC

解析1： 不变只是愿望，变化才是永恒。—SWIFT
一个接一个的软件项目都是一开始设计算法，然后将算法应用到待发布的软件中，接着根据时间进度把第一次开发的产品发布给客户。
对于大多数项目，第一个开发的系统并不适用。它可能太慢、太大、难以使用，或者三者兼有。要解决所有的问题，除了重新开始以外，没有其他的办法—即开发一个更灵巧或者更好的系统。系统的丢弃和重新设计可以一步完成，也可以一块块地实现。所有大型系统的经验都显示，这是必须完成的步骤。而且，新的系统概念或新技术会不断出现，因此开发的系统必须被抛弃，但即使是最优秀的项目计划也不能无所不知地在最开始就解决这些问题。
因此，管理上的问题不再是"是否构建一个实验性的系统，然后抛弃它"，你必须这样做。现在的问题是"是否预先计划抛弃原型的开发，或者是否将该原型发布给用户"。从这个角度看待问题，答案更加清晰。将原型发布给用户，虽然可以获得时间，但是其代价高昂—对于用户，使用极其痛苦；对于重新开发的人员，分散了精力；对于产品，影响了声誉，即使是最好的再设计也难以挽回名声。
因此，为舍弃而计划，无论如何，你一定要这样做。

17. BADCD

解析1：

如今也是一样。进度灾难、功能的不适应加上系统缺陷，这些都是由于左手不知道右手在做什么。当工作进行的时候，一些团队慢慢改变他们自己程序的功能、规模和速度，并且直接或间接地改变关于输入效用的假设和由输出组成的使用。

例如，一个覆盖程序函数的实现者可能遇上问题，并且减少依赖于展现这个函数在应用程序中多么罕见的统计的速度。与此同时回到农场，他的邻居可能是设计的一个主要部分的监督人，这样它极度取决于函数的速度。这种变化在速度本身成为一个主要规划变化，它需要对外宣布，从概念系统来做衡量。

那么，团队应当用尽可能多的方式彼此交流。

非正式的。良好的电话服务和明确定义的组间依赖关系将鼓励成百上千的书面文件共同翻译上所依赖的调用。

会议。定期项目会议，一个接一个给技术简报的团队是无价的。许多小的误解在这种方式下得到化解。

工作薄。一个正式的项目工作薄必须一开始就准备。

18. BABDC

解析1：

计算机将变得更加先进，也将变得更容易使用。提高了处理速度将使计算机变得更加容易操作。虚拟现实是一种用人的感官与计算机交换的技术，也将更有利于人和计算机接口。另外，奇异的模型正在开发中，包括使用生活有机体的生物计算，使用具有特定性能的分子计算和使用DNA（遗传的基本单位）来存储数据和进行操作等技术。这些可能是未来计算平台的例子，到目前为止，计算机的能力有限，并且局限于严格的理论。科学家们也正是因为小型化的嵌入式芯片电路的物理限制而研究新的技术。也有晶体管产生热量方面的限制，哪怕是最小的晶体管。

19. CDACB

解析1：

基本上，云计算仅仅意味着将IT资源作为服务来提供。几乎所有的IT资源都可以作为一个云服务来提供：如应用程序，计算能力，存储容量，网络，编程工具，甚至通讯服务和协作工具。

开始大规模提供云计算的互联网服务提供商有谷歌，亚马逊和其他一些基础建设商。云架构的特点是：系统不断扩大，水平分布系统资源，抽象的虚拟服务和管理，然后不断配置，汇集资源。在这种体系结构中，数据主要存放在"互联网某处"的服务器和"云服务"与客户端运行的应用程序上。

要想建立一个能收取使用费的云网格，必须要建立经得起单个元素或节点失败的云网格。当用网格来处理一批日常工作时，需要定义一个明确的开始和结束点，云服务可以是连续的。但更重要的是，云可以扩展可用的文件存储，数据库，网络服务来适用于网络和企业应用。

20. DBDAC

解析1：

极限编程是一个专业软件开发方法，它包含简单，沟通，反馈和勇气四大价值观。成功的软件开发是一个团队努力的结果，而这个团队不只是开发团队，而是由客户，管理和开发人员一起组成的更大的团队。极限编程是一个简单的过程，为了成功而将一些人聚集在一起，它主要是针对十几个或更少人的项目团队的面向对象开发。原则上极限编程适用于任何需要迅速和灵活提供高质量的轻量级项目。

一个极限编程项目需要客户的全程指导，而客户、程序员和项目经理都是必须的人员。客户是指那些有软件需要被急切发展的人，我们需要和他们建立有效的沟通方式来确定他们的需求，引导项目走向成功。

21. BCADD

解析1：

Ravi就像很多研究过以瀑布模型为软件生命周期过程的软件开发项目经理一样，他被安排使用瀑布模型去开发一个即将启动的项目，而且这是他的第一个任务。然而，Ravi发现不能在项目中使用瀑布模型，因为客户想要该软件分阶段交付，而不是作为一个整体交付。

在很多其他的项目中也有类似的情况，现实生活中，本来就很少有能完全按标准来进行处理的问题，可能某标准处理前一个问题非常合适，但处理现在这个问题就不定适合了。最合适的方法就是对一个新的问题必须采用切合它自身的方法。为了适应变化的变更请求而不失去对项目的控制，你必须要支持项目的发展过程与一个需求变更管理过程。

22. CABAB

解析1：

无论何时，当人们在鸡尾酒会上向别人介绍自己从事"计算机"、"电信"或"电子金融"方面工作时，总梦想自己参与了一些高科技领域。事实是研究者们在那些高科技领域取得了根本性的突破，而其余的人都只是应用他们的研究成果而已。

我们使用电脑和其他新技术开发各种新产品能增强我们工作团队的合作，方便大家沟通。在人类的很多事业中，成功来源于所有参与者的共同努力，失败是因为大家不相互协作。很多失败的主要原因是我们过多关注技术而忽略了人性的一面，这并不是因为人性更重要，而是因为它更容易存在问题。

在短短几个月内，相比于获知为什么霍勒斯心中忐忑不安或者为什么苏珊是不满意的，开发新磁盘的安装驱动要容易实现得多，这是因为人类的交流是非常复杂的，而且从来就不干脆，总是不清不楚。但它的作用确实非常重要的。

如果你发现自己专注于技术而不是社会学。就如一个杂耍人物在黑暗的街道丢失了钥匙，而在临近的街道去寻找，而他的原因是："临近街道的灯光更好"。

23. BCDBA

解析1：

观察一下编程人员，你可能会发现，同厨师一样，某项任务的计划进度，可能受限于顾客要求的紧迫程度，但紧迫程度无法控制实际的完成情况。就像约好在两分钟内完成一个煎蛋，看上去可能进行得非常好。但当它无法在两分钟内完成时，顾客只能选择等待或者生吃煎蛋。软件顾客的情况类似。

我现在并不认为软件经理内在的勇气和坚持不如厨师，或者不如其他工程经理。但为了满足顾客期望的日期而造成的不合理进度安排，在软件领域中却比其他的任何工程领域要普遍得多。而且，非量化方法的采用，少得可怜的数据支持，加上完全借助软件经理的直觉，这样的方式很难生产出健壮可靠和规避风险的估计。

显然我们需要两种解决方案。开发并推行生产率图表、缺陷率、估算规则等，整个组织最终会从这些数据的共享上获益。或者在基于可靠基础的估算出现之前，项目经理需要挺直腰杆并坚持他们的估计，确信自己的经验和直觉总比从期望得出的估计要强得多。

24. BCBDA

解析1：

虚拟化是IT行业缓存和共享资源的一种方法，通过这种方法可以更好地利用资源，并且自动提供资源以满足需求。传统的IT环境通常是一个竖井，技术和人力资源都是围绕应用或商业功能来安排的。利用虚拟化的架构，人员、过程和技术都集中于满足服务的程度，生产量被动态地分配，资源得到优化，而且整个架构得以简化，变得很灵活。我们提供了广泛的虚拟化解决方案，允许客户为他们的IT资源的基础架构选择最适用的路线和优化的重点。

25. CBCAC

解析1：

网络访问控制（NAC）的作用是限制对网络的访问，只允许注册的终端和认证的用户访问网络。然而MAC不是一个完整的LAM安全解决方案，另外还要实现主动的和被动的安全手段。Nevis是第一个也是仅有的全面的LAN安全解决方案，它以l0Gbps的速率对每一个分组进行深度的安全处理，在提供高级别安全的同时能保证网络应用的可利用性和适当的性能。Nevis集成了NAC作为LAM的第一道安全防线。此外，企业还需要实现基于角色的网络访问控制，以及起关键作用的主动安全测试—实时的多级安全威胁检测和微秒级的安全威胁堵截。集中的安全策略配置、管理和报告使其能够迅速地对问题进行分析，对用户的活动进行跟踪，这些都是实时可见的，也是历史可查的。

26. ACBAD

解析1：

WebSQL是一种类似于SQL的查询语言，用于从Web中提取信息。它能够在Web超文本中巡航，这使得它成为自动操作一个页面中有关链接的有用工具，或是作为搜索从一个给定的URL可以到达的、所有匹配某种模式的页面的有用工具。WebSQL也提供透明地访问索引服务器的手段，这种服务器可以通过公共网关接口进行查询。

27. BCADB

解析1：

Cookies原来是由Netscape通信公司引入的，这是HTTP服务器方应用程序的一种通用机制，就像CGI脚本一样，它可以由HTTP连接的客户方用于存储和检索信息。Cookies的基本功能是弥补HTTP无状态的缺陷。它能通过简单而持续地维护客户方的状态来扩展基于WWW的应用能力。

28. DCDAB

解析1：

近十年来，统一建模语言（UML）已经成为工业标准，它可用来可视化、规范化说明、构建以及文档化软件密集系统中的开发制品。作为事实上的工业标准，UML能够方便项目相关人员的沟通并减少理解上的二义问题。UML 2.0标准扩宽了该语言的应用范围，它所具有的表达能力能够让用户对企业信息系统、分布式Web系统和嵌入式实时系统进行建模。

UML不仅能够对软件系统进行建模，实际上，它具有足够的能力去对法律系统中的工作流、病人监护系统中的结构和行为、飞行战斗系统和硬件系统进行建模。

为了理解UML，需要具备该语言的概念模型，这需要学习三个主要元素：UML的基本构造块，基本构造块的关系规则和应用这些构造块与规则的通用机制。

29. ACBDA

解析1：

编程为什么有趣？作为回报，其从业者期望得到什么样的快乐？首先是一种创建事物的纯粹快乐。如同小孩在玩泥巴时感到愉快一样，成年人喜欢创建事物，特别是自己进行设计。其次，快乐来自于开发对其他人有用的东西。第三是整个过程体现出魔术般的力量——将相互啮合的零部件组装在一起，看到它们精妙地运行，得到预先所希望的结果。第四是学习的乐趣，来自于这项工作的非重复特性。人们所面临的问题，在某个或其他方面总有些不同，因而解决问题的人可以从中学习新的事物：有时是实践上的，有时是理论上的，或者兼而有之。最后，乐趣还来自于工作在如此易于驾驭的介质上。程序员就像诗人一样，几乎仅仅工作在单纯的思考中，凭空地运用自己的想象来建造自己的"城堡"。很少有这样的介质——创造的方式如此灵活，如此易于精炼和重建，如此容易地实现概念上的设想。

然而程序毕竟同诗歌不同，它是实实在在的东西；可以移动和运行，能独立产生可见的输出；能打印结果，绘制图形，发出声音，移动支架。编程非常有趣，在于它不仅满足了我们内心深处进行创造的渴望，而且还愉悦了每个人内在的情感。