软件工程术语

前　言

本标准主要变化如下：

a) 吸收了IEEE Std 610.12.1990 IEEE Standard Glossary of Software Engineering Terminology中的全部术语。

b) 原国标中与上述文本中相重叠的术语，参照上述文本中的定义，进行修改与整理。

c) 吸收了GB/T 5271.20－1994信息技术 词汇 20部分 系统开发中的术语及其定义。

d) 吸收了GB/T 8566－2001中的术语及其定义。

e) 吸收了GB/T 18905.1－2002中的术语及其定义。

f) 吸收了ISO/IEC 15504-9:1999中的术语及其定义。

g) 吸收了统一建模语言（UML-Unified Modeling Language）和元对象设施（MOF-Meta Object Facility）中的术语。

本标准结构如下：

a) 词条按英文对应词字母顺序排列。

b) 如果一个术语有一个以上的定义，则分别加以说明。定义中的()，括号中的词与括号前的可以互换；[]中的词是可有和可省略的。

c) 凡必要的地方用例子来说明定义。

d) 为了说明本标准中一个术语与另一些术语的关系，使用了下述词语：

－－见：指具有相同含义的或本质上相同意义的术语；

－－相对：指具有相反含义的或本质上不同意义的术语；

－－同义词：指同义的术语；

－－参见：……：指让读者参见推荐使用的或与之关系密切的术语。

软件工程术语

1 范围

本标准定义软件工程领域中通用的术语，适用于软件开发、使用维护、科研、教学和出版等方面。

2 术语定义及缩略语

2.1

第一代语言 1GL

first generation language 的缩略语。见： 机器语言 machine language（2.890）。

2.2

第二代语言 2GL

second generation language 的缩略语。见： 汇编语言 assembly language（2.86）。

2.3

第三代语言 3GL

third generation language 的缩略语。见： 高级语言 high order language（2.702）。

2.4

第四代语言 4GL

fourth generation language 的缩略语。

2.5

第五代语言 5GL

fifth generation language 的缩略语。

2.6

异（反）常终止 abend(abnormal end 的缩略语)

过程在完成前终止。参见： 夭折 abort（2.8）和异常 exception（2.575）。

2.7

异（反）常终止 abnormal end (abend)

过程在完成前终止。参见： 夭折 abort（2.8）和异常 exception（2.575）。

2.8

夭折，异常中止 abort

在一过程完成之前被迫终止。参见： 异常终止 abend（2.6）和异常 exception（2.575）。

2.9

绝对地址 absolute address

显式地址 explicit address

特定地址 specific address

一种地址，它永久地赋予某一设备或存储单元，用以标识该设备或单元而不需要转换或计算。

参见： 绝对汇编程序 absolute assembler（2.10）、绝对代码 absolute code（2.11）、绝对指令 absolute instruction（2.12）和绝对装入程序 absolute loader（2.13）。

相对： 相对地址 relative address（1331 条）、可重定位地址 relocatable address（2.1342）

和符号地址 symbolic address（2.1636）。

2.10

绝对汇编程序 absolute assembler

产生绝对代码的汇编程序。

相对： 重定位汇编程序 relocating assembler（2.1346）。

2.11

绝对代码 absolute code

特定代码 specific code

一种代码，其中，所有的地址都是绝对地址。

相对： 重定位代码 relocating code（2.1343）。

2.12

绝对指令 absolute instruction

一种计算机指令，其中，所有地址都是绝对地址。

参见： 直接指令 direct instruction（2.483）、有效指令 effective instruction（2.532）、立即指令 immediate instruction（2.722）和间接指令 indirect instruction（2.745 条）。

2.13

绝对装入程序 absolute loader

一种装入程序,它将绝对机器代码读至主存储器,它从由汇编程序或编译程序赋予的初始地址开始。在执行中，代码中无地址调整。

相对： 重定位装入程序 relocating loader（2.1347）。

2.14

绝对机器代码 absolute machine code

每当使用时，必须将其装入固定存储单元，且不能再定位的机器语言代码。

相对： 可重定位机器代码 relocatable machine code（2.1344）。

2.15

抽象数据类型 abstract data type

只对数据的特性和在其上的操作作出规定而不管数据如何表示以及操作如何实现的一种数据类型。

2.16

抽象机 abstract machine

a) 过程或机器的一种表示；

b) 像一台机器那样处理输入的一种模块。

2.17

抽象 abstraction

a) 对某一问题的概括，它抽取与某一特定目标相关的本质内容而忽略其非本质内容；

b) 形成上述概括的过程。

参见： 数据抽象 data abstraction（2.388）。

2.18

验收准则 acceptance criteria (criterion)

系统或部件必须满足的准则，其目的是使用户、客户或其他授权实体能够予以接受。

参见：需求 requirement（2.1361）和测试准则 test criteria（2.1699）。

2.19

验收测试 acceptance testing

a) 确定一系统是否符合其验收准则，使客户能确定是否接收此系统的正式测试；

参见： 合格性测试 qualification testing（2.1291）、系统测试 system testing（2.1669）。

b) 使用户、客户或其他授权实体确定是否接受系统或部件的正式测试。

参见： 运行测试 operational testing（2.1065）和合格性测试 qualification testing（2.1291）。

相对： 开发测试 development testing（2.468）。

2.20

可达性 accessibility

组成软件的各部分便于选择使用或维护的程度。

2.21

访问（存取）控制机制 access-control mechanism

为使某一计算机系统或其某一部分允许被获准者和防止未获准者接触、访问而设计的硬件或软件的特性、操作过程或管理过程。

2.22

准确[度] accuracy

a) 无误差的一种品质；

b) 正确性或无误差的一种定性评估，该评估越高，对应的误差越小或正确性越高；

c) 对误差大小的一种度量，最好将其表示成相对误差的函数，此度量越高，对应的误差越小；

d) 对误差程度的一种定量测量。

相对： 精度 precision（2.1160）。

2.23

需方 acquirer

从供方获得或得到一个系统、产品或服务的一个机构。

注：需方可以是买主、客户、拥有者、用户或采购人等。

2.24

获取 acquisition

得到一个系统、一个产品或一项服务的过程。

2.25

动作 action

对某一可执行语句，形成对计算步骤一种抽象的规格说明。典型的动作引起系统状态的某种变更，并通过向对象发送消息或者个性链接或属性予以实施。

2.26

措施项 action item

a) 已安排给个人或小组处理的工作清单中的一个单元；

b) 一项已被接受的措施建议。

2.27

措施建议 action proposal

更改过程或过程有关项的文档化建议，它将预防未来再出现已被缺陷预防活动标识出的缺陷。

参见： 软件过程改进建议 software process improvement proposal（2.1517）。

2.28

动作序列 action sequence

决定动作的序列的一种表示式。

2.29

动作状态 action state

表示执行某一原子动作（有代表性的是启用某一操作）的一种状态。

2.30

激活 activation

执行某一动作。

2.31

主动类 active class

一种其实例是主动对象的类。

参见： 主动对象 active object（2.33）。

2.32

活动文件 active file

尚未超过终止时间的文件。

2.33

主动对象 active object

一种拥有某一线程并能初启控制活动的对象。主动类的一个实例。

参见： 主动类 active class（2.31）和线程 thread（2.1732）。

2.34

活动冗余 active redundancy

在容错系统中，为防止故障或允许从故障中恢复，所使用的同时操作的冗余部件的技术。

相对： 备用冗余 standby redundancy（2.1565）。

2.35

活动 activity

a) 一个过程的组成元素；

注：对基线的变更要经有关机构的正式批准。

b) 为实现某个目的而采取的任一步骤或执行的任一职能，既可是脑力的，也可是体力的。活动包括经理和技术人员为完成项目和组织的任务而作的全部工作。

2.36

活动图 activity graph

用于对涉及一个或多个类目的进程建模的状态机的一种特例。

相对： 状态图 state diagram（2.1570）。

2.37

施动者[类] actor[class]

各用况的使用者在与某些用况交互时所扮演角色的一种紧凑集合。一个施动者对每一用况都有一个与之通信的角色。

2.38

实际指令 actual instruction

见： 有效指令 effective instruction（2.532）。

2.39

实参 actual parameter

在调用子程序时，用来指定数据或要传输给该子程序的程序元素的数值或表达式。

见： 自变量 argument（2.74）。

相对： 形参 formal parameter（2.645）。

2.40

适应性 adaptability

使不同的系统约束条件和用户需求得到满足的容易程度。

见： 灵活性 flexibility（2.634）。

2.41

适应数据 adaptation data

一种数据，它用于适应给定安装站点的或在它的操作环境中有给定条件的程序。

2.42

适应参数 adaptation parameter

给定一特定值的变量，其目的是以适应安装站点或在它的操作环境中给定条件的程序。例如变量 Installation_Site_Latitude。

2.43

适应性维护 adaptive maintenance

为使软件产品在变更了的环境下仍能使用而进行的维护。

相对： 纠正性维护 corrective maintenance（2.354）和完善性维护 perfective maintenance（2.1130）。

2.44

地址 address

a) 标识一寄存器、设备、存储器特定部分、或其他一些数据来源或目的地的一个数、字符或一组字符；

b) 用来指定一设备或一个数据项；

c) 由标识的数、字符或一组字符引用一设备或存储单元。

参见： 绝对地址 absolute address（2.9）、有效地址 effective address（2.531）、隐含地址 implied address（2.731）、间接地址 indirect address（2.744）、相对地址 relative address（2.1331）、可重定位地址 relocatable address（2.1342）、符号地址 symbolic address（2.1636）和虚拟地址 virtual address（2.1836）。

2.45

地址字段 address field

地址部分 address part

计算机指令的任一字段，此字段包含地址、导出地址必需的信息或操作数值。

相对： 操作字段 operation field（2.1060）。

2.46

地址格式 address format

a) 在计算机指令中地址字段数和排列；

参见： n 地址指令 n - address instruction（2.1002）和 n+1 地址指令 n - plus-one address instruction（2.1008）。

b) 在地址中元素数和排列，例如需要标识具体通道、磁盘扇区和在磁盘存储中的记录的元素。

2.47

地址修改 address modification

在地址上执行的任何算术的、逻辑的或句法的操作。

参见： 有效地址 effective address（2.531）、索引地址 indexed address（2.740）、相对地址 relative address（2.1331）和可重定位地址 relocatable address（2.1342）。

2.48

地址部分 address part

见： 地址字段 address field（2.45）。

2.49

地址空间 address space

a) 计算机程序能访问的地址范围；

注： 在某些系统中，这可能是与其他程序分开的程序能访问的物理地址集和引用可由其他程序访问的存储单元的虚拟地址集。

b) 中央处理单元能寻址的存储单元数。

2.50

寻址异常 addressing exception

当程序计算地址时，超出了可用的地址边界所发生的异常。

参见： 数据异常 data exception（2.395）、操作异常 operation exception（2.1059）、溢出异常 overflow exception（2.1082）、保护异常 protection exception（2.1274）和下溢异常 underflow exception（2.1786）。

2.51

传入的 afferent

在软件系统中，从子模块至超级模块的数据或控制流。

相对： 传出的 efferent（2.534）。

2.52

聚集[类]型 aggregate [class]

在聚集（整体）与某一构件部分之间，规定整体一部分联系的关联一种特别形式。

参见：组合 composite（2.265）。

2.53

代数语言 algebraic language

一种编程语言，它允许语句的结构重新装配代数表达式，例如 Y = X + 5。例如 FORTRAN。

参见： 算法语言 algorithmic language（2.56）、表处理语言 list processing language（2.860）和逻辑编程语言 logic programming language（2.873）。

2.54

算法 algorithm

a) 用有限步数求解某问题的一套明确定义的规则的集合：例如，求 sin（x）到给定精度的一系列算术运算的顺序的完整规格说明；

b) 为执行特定任务的任何运算序列。

2.55

算法分析 algorithm analysis

对一算法的检查，其目的在于确定与其预期用途有关的正确性，确定其运行特性，或为了更充分地理解某一算法以便对其进行修改、简化或改进。

2.56

算法语言 algorithmic language

为处理算法而设计的编程语言、例如，ALGOL。

参见： 代数语言 algebraic language（2.53）、表处理语言 list processing language（2.860）和逻辑编程语言 logic programming language（2.873）。

2.57

分配的基线 allocated baseline

在配置管理中，初始批准的规格说明，它支配作为较高级配置项的一部分的配置项的开发。

参见： 分配的配置标识 allocated configuration identification（2.58）。

相对：开发配置 developmental configuration（2.470）、功能基线 functional baseline（2.659）和产品基线 product baseline（2.1213）。

2.58

分配的配置标识 allocated configuration identification

在配置管理中，当前批准的规格说明，它支配作为较高级配置项的一部分的配置项的开发。每一种规格说明定义从较高级配置项分配的功能特征，建立测试要求，以证明分配的功能特征的成功，描绘与其他相关的配置项必需的接口需求并建立设计限制（若有）。

参见： 分配的基线 allocated baseline（2.57）。

相对： 功能配置标识 functional configuration identification（2.663）和产品配置标识 product configuration identification（2.1215）。

2.59

别名 alias

a）某一项目的另一个名字；

b） 一个替换标号。例如,可以使用一个标号和一个或多个别名来指示计算机程序中同一元素或点。

2.60

分配 allocation

a） 在系统或程序的部件间分发需求、资源或其他实体的过程；

b） 上述 a 中分发的结果。

2.61

分析 analysis

在软件开发过程中，其首要目的是对问题领域构建成某一模型的阶段。分析侧重于做些什么，设计则侧重于如何去做。

相对： 设计 design（2.439）。

2.62

分析阶段 analysis phase

参见： 需求阶段 requirements phase（2.1365）。

2.63

分析模型 analytical model

用一组可解方程来表示一个过程或现象。

相对： 仿真 simulation（2.1453）。

2.64

反常 anomaly

从文档或软件操作观察到偏离以前验证过的软件产品或引用的文档的任何事件。

2.65

先期缓冲 anticipatory buffering

一种缓冲技术，在预期到数据需要时，数据存储在缓冲器中。

参见： 动态缓冲 dynamic buffering（2.517）和简单缓冲 simple buffering（2.1451）。

2.66

先期调页 anticipatory paging

一种存储分配技术，在预料到需要那些页时，页从辅助存储器传送至主存储器。

相对： 按需调页 demand paging（2.431）。

2.67

应用领域 application domain

一组已界定的相关系统(即处理一个特定类型问题的各系统)。在一个应用领域内，开发和维护工作一般需要专门的技能和(或) 资源。例如工资和人事系统、指挥和控制系统、编译程序及其专家系统。

2.68

应用[程序]生成器 application generator

一种代码生成器，它在具体的应用领域为解决一个或多个问题，产生代码。例如，工资生成器。

2.69

面向应用的语言 application-oriented language

a） 一种计算机语言，具有用于某种单一应用领域的手段或记号。例如，计算机辅助指令和硬件设计语言；

b) 一种面向问题的语言，其语句包含或汇集了用户职业的术语。

参见：写作语言 authoring language（2.106）、规格说明语言 specification language（2.1550）和查询语言 query language（2.1303）。

2.70

应用问题 application problem

一种由最终用户提出，并要求通过信息处理来解决的问题。

2.71

应用软件 application software

设计用于实现用户的特定需要而非计算机本身问题的软件。例如，导航（浏览）、工资、过程控制软件。

相对： 支持软件 support software（2.1632）和系统软件 system software（2.1667）。

2.72

体系结构设计 architectural design

a) 定义硬件和软件部件和它们的接口集合以建立计算机系统开发的框架的过程；

参见：功能设计 functional design（2.665）。

b) 体系结构设计过程的结果。

2.73

体系结构 architecture

系统或部件的组织结构。

参见： 部件 component（2.261）、模块 module（2.977）、子程序 subprogram（2.1618）、例程 routine（2.1402）、程序体系结构 program architecture（2.1226）和系统体系结构 system architecture（2.1647）。

2.74

自变量，变元 argument

实参 actual parameter

a) 一独立的变量、例如，在等式 $E = mc^2$ 中的变量 m ；

b) 独立变量的一特定值。例如，值 m = 24kg ；

c) 用于软件模块调用中的一个常数、变量或表达式以规定传送至此模块的数据或程序元素。

相对： 形参 formal parameter（2.645）。

2.75

数组 array

由单个名和一个或多个索引标识的数据项所组成的 n 维有序集，该集中的每个元素可以分别寻址。例如，矩阵、表或向量。

2.76

制品 artifact

产品 product

由某一种软件开发过程所使用的或产生的一种信息的物理件。制品的实例有模型、源文件、文字和二进制可执行文件。制品可构成可部署构件的实现。

相对： 构件 component（2.261）。

2.77

人工智能 artificial intelligence

a) 计算机科学的一个分支，专门研制执行通常与人的智能有关联的功能（例如，推理、学习和自改进）的数据处理系统；

b) 某一设备执行通常与人的智能有关联的功能（例如，推理、学习和自改进）的能力。

2.78

人工语言 artificial language

参见： 形式语言 formal language（2.644）。

2.79

汇编 assemble

把用汇编语言表示的程序翻译成等效的机器语言程序，有时还要连接子程序。实现汇编的常用方法是用机器语言操作码代替汇编语言操作码，并用绝对地址、中间地址、浮动地址或虚拟地址来代替符号地址。

相对： 编译 compile（2.252）、反汇编 disassemble（2.487）和解释 interpret（2.808）。

2.80

汇编并运行 assemble-and-go

一种操作技术，它在计算机程序的汇编、连接、装入和运行之间没有停顿。

2.81

汇编的起点 Assembled origin

由汇编程序、编译程序或连接编辑器赋予计算机程序的初始存储单元的地址。

参见： 偏移量 offset（2.1043）和启动地址 starting address（2.1567）。

相对： 装入的起点 loaded origin（2.867）。

2.82

汇编程序 assembler

用于把用汇编语言表示的程序转换为等效的用机器语言表示的程序的计算机程序。

参见： 绝对汇编程序 absolute assembler（2.10）、交叉汇编程序 cross-assembler（2.373）和重定位汇编程序 relocating assembler（2.1346）。

相对： 编译程序 compiler（2.254）和解释程序 interpreter（2.809）。

2.83

汇编程序代码 assembler code

见： 汇编代码 assembly code（2.85）。

2.84

汇编程序语言 assembler language

见： 汇编语言 assembly language（2.86）。

2.85

汇编代码 assembly code

汇编程序代码 assembler code

用汇编程序能识别和处理的形式表示的计算机指令和数据定义。(assembler code)。

相对： 编译程序代码 compiler code（2.255）、解释代码 interpretive code（2.810）和机器代码 machine code（2.887）。

2.86

汇编语言 assembly language

汇编程序语言 assembler language

低级语言 low level language

第二代语言 second generation language

a) 一种编程语言，它非常接近给定计算机的指令集，允许用符号命名操作和地址且通常是程序指令与计算机指令一一对应，且能提供使用宏指令的便利；

b) 一种特定机器语言，其指令通常和计算机指令一一对应。

相对：第五代语言 fifthgeneration language(2.623)、第四代语言 fourth generation language（2.654）、高级语言 high order language（2.702）和机器语言 machine language（2.890）。

2.87

断言 assertion

一种逻辑表达式，它规定必须存在的一种程序状态，或规定在程序执行过程中某一特定点上程序变量必须满足的条件集合。类型包括输入断言、循环断言、输出断言。

参见： 不变式 invariant（2.818）和正确性证明 proof of correctness（2.1270）。

2.88

评估的能力 assessed capability

按相关文件的规定，所完成的一次或多次最近的、相关的过程评估的输出。

2.89

评估约束 assessment constraints

评估小组选择的自由度方面的限制，它与评估执行和评估输出的使用相关。

2.90

评估指标 assessment indicator

实践或工作产品的客观属性或特性，它用以支持鉴定某已实施过程的性能或能力。

2.91

评估输入 assessment input

为能启动过程评估所需的信息汇集。

2.92

评估工具 assessment instrument

在评估中所用的一个或一组工具，它用以帮助评估师评价过程性能或能力、处理评估数据和记录评估结果。

2.93

评估输出 assessment output

评估所产生的所有实质性的结果（见： 评估记录 assessment record（2.96））。

2.94

评估参与者 assessment participant

在评估范围内具有职责的个人。

注：包括但并不仅限于如发起人、评估师、组织单位成员等。

2.95

评估目的 assessment purpose

用作评估输入的一部分并定义执行评估理由的一种陈述。

2.96

评估记录 assessment record

对那些与评估有关的信息进行的有序的、文件化的汇集，它有助于理解和验证评估所产生的过程轮廓。

2.97

评估范围 assessment scope

对评估界限的规定，作为评估输入的一部分，包括被评估组织对该次评估的限制、应包含的过程及运行过程的环境（见： 过程环境 process context（2.1194））。

2.98

评估发起者 assessment sponsor

被评估组织内部或外部的、要求进行评估并对执行评估提供资金或其他资源的个人。

2.99

赋值语句 assignment statement

一种计算机程序语句，它用于表达一系列操作，或用于把操作数赋给指定变量、或符号、或变量和

符号两者。例如，Y = X + 5。

参见： 清除 clear（2.213）、初始化 initialize（2.756）和复位 reset（2.1372）。

相对： 控制语句 control statement（2.343）和声明 declaration（2.416）。

2.100

关联 association

规定其实例间连接的多个类目之间的语义联系。

2.101

关联类 association class

一种兼有关联与类双重性质的模型元素。可以将关联类视为具有类性质的关联，或视为具有关联性质的类。

2.102

关联端 association end

对某一关联，将其连接到类目上的那个端点。

2.103

原子类型 atomic type

原语类型 primitive type

一种数据类型，它的每一个成员由单个不可分解的数据项构成。

相对： 复合类型 composite type（2.268）。

2.104

属性 attribute

一个项的特征。例如，项的颜色、尺寸、类型。

参见： 质量属性 quality attribute（2.1295）。

2.105

审核（审计） audit

a) 为评估工作产品或工作产品集是否符合软件需求、规格说明、基线、标准、过程、指令、代码以及合同和特殊要求而进行的一种独立的检查；

参见： 功能配置审核 functional configuration audit（2.662）、物理配置审核 physical configuration audit（2.1139）和代码审核 code audit（2.219）。

b) 通过调查研究确定已制定的过程、指令、规格说明、代码和标准或其他的合同及特殊要求是否恰当和被遵守，以及其实现是否有效而进行的活动。

2.106

写作语言 authoring language

用于开发计算机辅助指令的课件的高级编程语言。

参见： 写作系统 authoring system（2.107）。

2.107

写作系统 authoring system

与写作语言形成一体化的编程系统。

2.108

自动设计工具 automated design tool

帮助进行软件设计的综合、分析、模拟或文档编制的软件工具。自动设计工具的例子如：仿真器、分析工具、设计表示处理器和文件生成器。

2.109

自动测试用例生成器 automated test case generator

参见：自动测试生成器 automated test generator（2.111）。

2.110

自动测试数据生成器 automated test data generator

参见：自动测试生成器条 automated test generator（2.111）。

2.111

自动测试生成器 automated test generator

一种软件工具，它以计算机程序和准则作为输入，产生满足这些准则要求的测试输入数据，有时还确定预期的结果。

2.112

自动验证系统 automated verification system

a) 一种软件工具，它以计算机程序及其规格说明的表示作为输入〈可能借助人的帮助〉，产生该程序的正确与否的证明；

参见： 自动验证工具 automated verification tools（2.113）。

b) 验证过程的自动部分或全部的任何软件工具。

2.113

自动验证工具 automated verification tools

a) 用于评估软件开发过程中的产品的一类软件工具。这些工具有助于验证正确性、完全性、一致性、可跟踪性、可测试性，以及检查是否遵守了标准。软件验证工具包括设计分析器、自动验证系统；

b) 静态分析器、动态分析器和标准实施器。

2.114

辅助类 auxiliary class

以实现二级逻辑式或控制流为典型方式，来支持另一更靠中央或更基础的类的一种衍型类。辅助类与焦点类一起使用具有代表性，且在设计期间，对规定各构件的二级业务逻辑或控制流特别有用。

参见： 焦点类 focus class（2.639）。

2.115

可用性 availability

a) 软件（系统或部件）在投入使用时可操作和可访问的程度或能实现其指定的系统功能的概率；

参见：容错 error tolerance（2.570）、故障容忍 fault tolerance（2.617）和健壮性 robustness（2.1397）。

b) 系统正常工作时间和总的运行时间之比；

c) 在运行时，某一配置项实现指定功能的能力。

2.116

可用性模型 availability model

用于预测、估计、判定可用性的模型。

2.117

比对测试 back-to-back testing

程序的两个或多个变体用同样的输入执行，比较输出，在发生差别情况下分析错误。

参见： 变异测试 mutation testing（2.1001）。

2.118

后台 background

在作业调度中，一种计算环境，低优先级的或不要求用户交互的进程在此环境中执行。

参见： 后台处理 background processing（2.119）。

相对： 前台 foreground（2.641）。

2.119

后台处理 background processing

当高优先级的进程不用计算机资源时，低优先级进程的执行或不要求用户交互的进程的执行。

相对：前台处理 foreground processing（2.642）。

2.120

备份，后备 backup

a) 在故障或外部引起的灾害事件中可用于替换或帮助恢复原始的项的系统、部件、文件、过程或人；

b) 像在上述 a 中为建立或指定系统、部件、文件、过程或人；

c) 发生系统失效或灾害时，为恢复数据文件或软件、重新起动处理、使用备份计算机设备而做的准备。

2.121

后备程序员 backup programmer

主程序员组的助理人员、响应包括分派由组正在开发的软件的有效部分，帮助主程序员评审其他组成员的工作，当需要时代替主程序员且对正在开发的软件有全局技术了解。

参见： 主程序员 chief programmer（2.205）。

2.122

后向执行 backward execution

见： 可逆的执行 reversible execution（2.1391）。

2.123

后向恢复（反向恢复） backward recovery

a) 文件再建至一给定的状态。这是由恢复自那个状态以来文件的所有变更来实现的；

b) 一种类型的恢复，在此情况下，系统、程序、数据库或其他系统资源恢复到能执行所要求的功能的以前的状态。

相对： 前向恢复 forward recovery（2.650）。

2.124

基地址 base address

作为一个基准点的地址，它用于与一个相对地址相加，以确定要访问的存储单元的地址。

参见： 索引地址 indexed address（2.740）、相对地址 relative address（2.1331）和自相关地址 self-relative address（2.1429）。

2.125

基线 baseline

a) 业已经过正式审核与同意，可用作下一步开发的基础，并且只有通过正式的修改管理过程方能加以修改的规格说明或产品；

b) 在配置项目生存周期的某一特定时间内，正式指定或固定下来的配置标识文件和一组这样的文件。基线加上根据这些基线批准同意的改动构成了当前配置标识；

参见： 分配的基线 allocated baseline（2.57）、开发配置 developmental configuration（2.470）、功能基线 functional baseline（2.659）和产品基线 product baseline（2.1213）。

c) 任何协议或在一给定时间赋予或固定的结果，如要变更，要求证明和批准。

对于配置管理，有以下三种基线：

功能基线－最初通过的功能配置；

分配基线－最初通过的分配的配；

产品基线－最初通过的或有条件地通过的产品配置。

2.126

基线配置管理 baseline configuration management

正式评审和同意并作为其后开发工作依据的基线的建立。一些软件工作产品，如软件设计和代码，

在预定点上具有建立的基线，并且对其实施严格的更改控制过程。

参见： 基线管理 base management（2.127）。

2.127

基线管理 baseline management

在配置管理中，技术的和管理的应用指导指定的文档和他们的变更，这些文档是在配置项的生存期期间的某一特定时间正式标识和建立基线的。

2.128

批处理的 batch

用于说明一种系统或一种操作方式，在其中，输入被集合起来并一次一起处理，而不是每个输入在到达时就处理。且一个作业，一旦启动，一直到完成没有附加的输入或用户交互。

相对： 会话的 conversational（2.348）、交互的 interactive（2.792）、联机的 on-line（2.1045）和实时的 real time（2.1313）。

2.129

浴盆曲线 bathtub curve

在系统或部件中失效数作为时间函数的图形。由图形的通常形状得名：失效的减量周期（早期失效周期），跟着是相对稳定周期（定失效率周期）跟着是失效增量周期（磨损失效周期）。

2.130

开始-结束块 begin-end block

由 begin 和 end 分隔符括起来的设计或程序语句序列。其特征是具有单一的入口和单一的出口。

2.131

行为 behavior

对某一操作或事件的可观察效果（包括该操作或事件的结果）。

2.132

行为特征 behavioral feature

模型元素的一种动态特征，例如一个操作或方法。

2.133

行为模型方面 behavioral model aspect

着重系统中各实例的行为（包括方法、协作和状态）一种模型面貌。

2.134

基准 benchmark

a) 针对能做的测量或比较的一种标准；

b) 能用于比较系统、部件或上述 a 中的标准的过程、问题或测试；

c) 一种恢复文件。

2.135

投标方 bidder

已提交建议，并作为签订一个或多个产品设计、开发和(或)制造合同候选者的个人、合伙组织、公司或协会。

2.136

总揽测试 big-bang testing

一类集成测试，它把软件元素、硬件元素或两者组合在一起，作为整个系统进行测试而不是分阶段测试。

2.137

二元关联 binary association

两个类的之间的一种关联。N 元关联的一个特例。

2.138

二进制数字[位] binary digit(bit)

a) 能用 0 或 1 表示的信息单元；

b) 能保持 a 中信息单元的计算机存储的元素；

c) 用于表示二进制数字系统，0 或 1 两个数字之一的数字。

参见： 字节 byte（2.170）和字 word（2.1850）。

2.139

[捆]绑 bind

把一个值赋给标识符。例如，赋值给一参数或在计算机程序中，赋绝对地址给符号地址。

参见： 动态绑定 dynamic binding（2.515）和静态绑定 static binding（2.1579）。

2.140

绑定 binding

把一个值或指定的对象〈referent〉赋给某一标识符。例如，把一个值赋给一个参数或把绝对地址、虚拟地址或设备标识符分配给计算机程序中的符号地址或标号。

参见：动态绑定 dynamic binding（2.515）和静态绑定 static binding（2.1579）。

2.141

黑盒 black box

a) 一个系统或部件，它的输入、输出和通用功能是已知的，但它的内容或实现是未知的或无关的；

相对： 白盒 glass box（2.677）。

b) 属于一种方法，它把系统或部件按 a)那样对待。

参见： 封装 （2.542encapsulation）。

2.142

黑盒测试 black-box testing

见： 功能测试 functional testing（2.669）a）。

2.143

块【名】、阻塞【动】 block

a) 由某些技术或逻辑原因形成的被当作一个实体看待的一串记录、一串字或一字符串；

b) 作为一个单元对待的一组连续的存储单元、计算机程序语句、记录、字、字符或位；

参见： 块结构语言 block-structured language（2.146 ）和定界符 delimiter（2.429）。

c) 被当作一个单元而加以传送的一组二进制位数或 N 进制位数。通常对这组二进制位数或 N 进制位数采用某种编码步骤以达到出错控制的目的；

d) 作为一个单元来处理的事物，如字、字符或数字的集合；

e) 参见： 程序块 program block（2.1227）；

f) 系统中的某些操作因某种原因，暂时不能继续执行。

2.144

块分配 block allocation

见： 调页 paging（2.1101）。

2.145

框图 block diagram

系统资源图 system resources chart

表示某一系统、计算机或设备的图（见图 1），图中主要部分由加有适当注释的几何图形来表示，用以说明这些主要部分的基本功能及其功能关系。

参见： 盒图 box diagram（2.156）、泡图 bubble chart（2.161）、图 graph（2.684）、输入处理输出图 input-process output chart（2.763）和结构图 structure chart（2.1605）。

对照： 流程图 flowchart（2.637）。

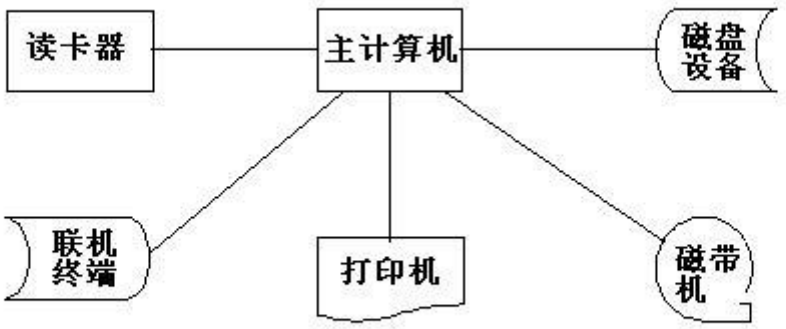

图1 框图

2.146

块结构语言 block-structured language

一种设计或程序设计语言，在这种语言中，定义语句序列、调用块，它们通常是由 begin 和 end 界限符界定。在一块中定义的变量和标号，在此块外是不识别的。例子包括 Ada、ALGOL、PL/1。

参见： 结构化编程语言 structured programming language（2.1610）和程序块 program block（2.1227）。

2.147

分块因子 blocking factor

块中的记录、字、字符或位的个数。

2.148

布尔量 Boolean

一种枚举类型，其值只有“真”或“假”两种。

2.149

布尔表达式 Boolean expression

一种求出布尔值的表达式。

2.150

引导 boot

通过清除存储器和重新加载操作系统来初始化计算机系统。从 bootstrap 导出。

2.151

引导程序 bootstrap

初始程序装入 initial program load

a) 常驻或很容易装入计算机的一种短的计算机程序，它的执行能把另一个较大的程序，如操作系统或其装入程序引入内存；

b) 一组指令，它能使其他的指令被装入直到计算机程序全部都装入内存中为止；

c) 借助自身的动作而使其达到所希望的状态的一种技术或设备。例如，一段机器子程序，其前几条指令足以使其余部分指令从输入设备输入到计算机中；

d) 用于建立计算机程序另一版本的部分计算机程序；

e) 使用一引导程序；

f) 能起 a 中规定的作用的程序。

2.152

引导[程序]装入程序 bootstrap loader

使用预置计算机操作以装入引导程序（bootstrap）的一种短的计算机程序。

2.153

自底向上 bottom-up

用于说明一种方法，这种方法从层次结构的最低层软件组成部分开始，逐级向上直至最高层组成成分为止，例如，自底向上设计、自底向上程序设计、自底向上测试等。

参见： 关键部分优先 critical piece first（2.368）。

相对： 自顶向下 top-down（2.1746）。

2.154

自底向上设计 bottom-up design

从最基本的或原始的部分着手，逐级进入到较高层部分的系统设计方法。

相对： 自顶向下设计 top-down design（2.1747）。

2.155

边界值 boundary value

相应于为系统或部件规定的最小或最大的输入、内部、输出的数据值。

2.156

盒图 box diagram

箱图 Chapin chart

Nassi-Shneiderman 图 Nassi-Shneiderman chart

程序结构图 program structure diagram

一种控制流程图（见图 2），它由矩形构成，又可分解以表示顺序步，if-then-else 条件，重复和 case 条件。

参见：框图 block diagram（2.145）、泡图 bubble chart（2.161）、流程图 flowchart（2.637）、图 graph（2.684）、输入-处理-输出图 input-process-output（2.763）和结构图 structure chart（2.1605）。

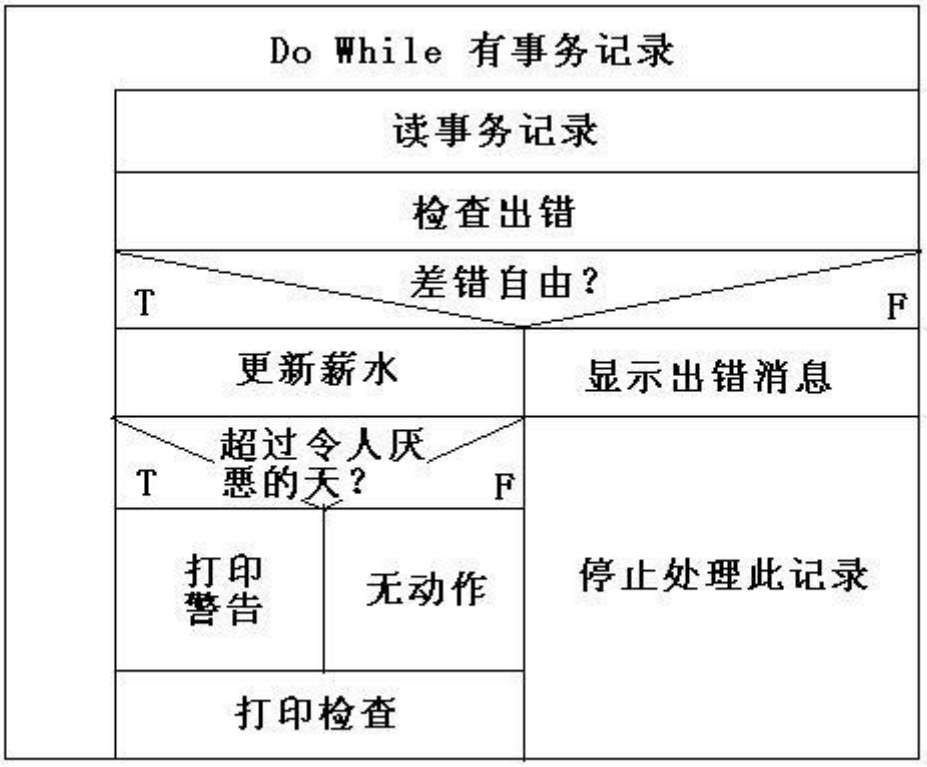

图2 盒图（box diagram）

2.157

分支 branch

a) 一种计算机程序结构，在此结构中，可选择两个或多个程序语句的可替换的集之一执行。

参见： 分情况 case（2.186）、跳转 jump（2.830）、“转至 go to（2.683）”语句和“若-则-否则”语句 if-then-else（2.718）；

b) 计算机程序中选择两个或多个程序语句的可替换的集之一的执行点。同义词： 分支点 branchpoint (2.159)；

c) 在a)中程序语句的任一可替换集；

d) 执行在a)中的选择。

2.158

分支测试 branch testing

一种测试，设计用于执行计算机程序中每一个判定点的每个出口。

相对： 路径测试 path testing（2.1122）和语句测试 statement testing（2.1575）。

2.159

分支点 branchpoint

见： 分支 branch（2.157）b）。

2.160

断点 breakpoint

计算机程序中能暂停执行，允许人工或自动监控程序性能或结果的点。类型包括代码断点（code breakpoint）、数据断点（data breakpoint）、动态断点（dynamic breakpoint）、跋断点（epilog breakpoint）、可编程断点(programmable breakpoint)、序断点(prolog breakpoint)和静态断点(static breakpoint)。

注： 当程序中的一点和在此点引起执行暂停的事件都设置时，认为设置了一断点，当程序执行暂停时，认为断点启动。

2.161

泡图 bubble chart

一种数据流程、数据结构或其他图（见图 3），其中，实体用圆圈（circle）表示，关系用圆圈间的连接表示。

参见： 框图 block diagram（2.145）、盒图 box diagram（2.156）、流程图 flowchart（2.637）、图 graph(2.684)、输入-处理-输出图 input-process-output chart(2.763)和结构图 structure chart（2.1605）。

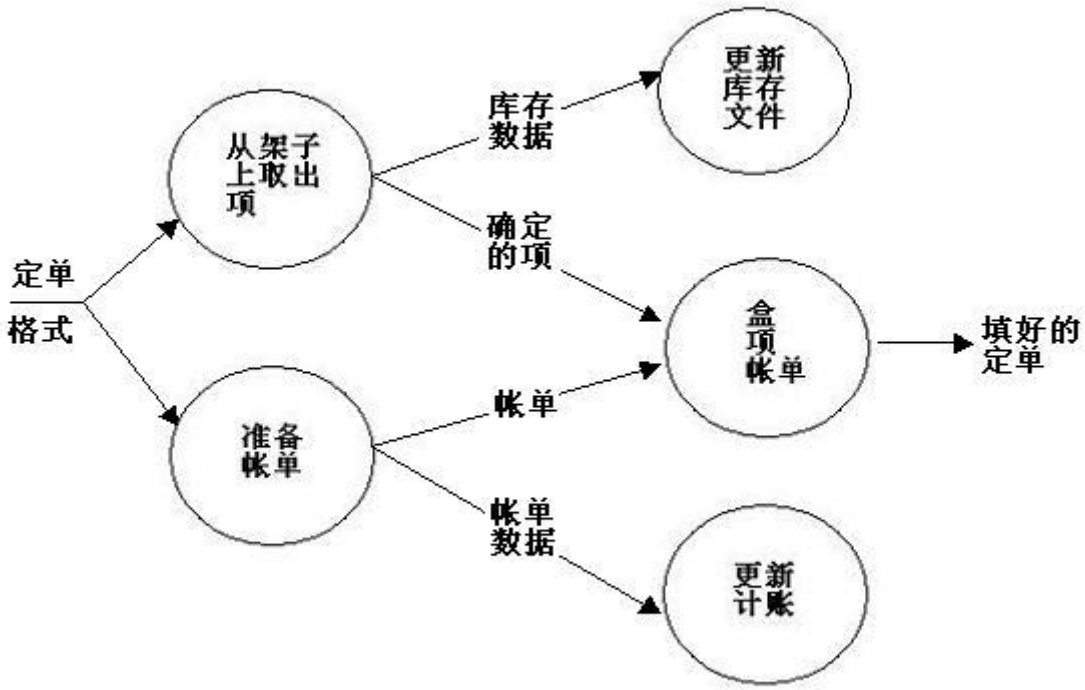

图3 泡图

2.162

缓冲器，缓冲区 buffer

a) 用于临时存储数据的设备或存储区域以补偿数据流的速率、事件发生的时间或能由设备或进程处理的数据总数（包括在传送或使用中的数据）的差别；

b) 实现 a 中目标的例程；

c) 分配、调度或使用像在 a 中的设备或存储区域。

参见： 先行缓冲 anticipatory buffering（2.65）、动态缓冲 dynamic buffering（2.517）和简单缓冲 simple buffering（2.1451）。

2.163

隐错 bug

见： 出错 error（2.561）和故障 fault（2.609）。

2.164

隐错播种 bug seeding

参见： 差错播种 error seeding（2.569）和故障播种 fault seeding（2.615）。

2.165

构建版 build

软件产品（系统或部件）的一个工作版本，其中包含最终产品将提供的能力的一个规定的子集。

2.166

构建版块 building block

较高一级程序或模块使用的一个单元或模块。

2.167

老化周期 burn-in period

见： 早期失效周期 early-failure period（2.526）。

2.168

忙 busy

用于说明系统或部件在操作、在服务和在使用。

参见： 关机 down（2.502）、空闲 idle（2.716）和开机 up（2.1797）。

2.169

忙时 busy time

在计算机性能工程中，系统或部件在工作、服务、使用期间的时间周期。

参见：关机时间 down time(2.503)、空闲时间 idle time(2.717)、设置时间 set-up time(2.1444)和开机时间 up time（2.1798）。

2.170

字节 byte

a) 作为一个单元进行操作的一组邻接的二进制数字，它通常短于计算机字(经常暗指一八位组)；

b) 能容纳如 a 中的位组的计算机存储元素。

参见： 位 bit（2.138）和字 word（2.1850）。

2.171

调用 call

插进 cue

a) 从一个软件模块至另一个软件模块的控制传送，通常，隐含着控制将返回调用的模块。

相对： 转至 go to（2.683）；

b) 一条计算机指令，它从一软件模块把控制传送至另一软件模块，如 a)中那样，且经常规定传送或返回的参数、

c) 像在 a)中那样，从一个软件模块把控制传送至另一个软件模块，并且，常常传送参数至其他模块。

参见： 按名调用 call by name（2.17）、引址调用 call by reference（1.175）、按值调用 call by value（2.176）和调用顺序 calling sequence（2.181）。

2.172

按址调用 call by address

见： 引址调用 call by reference（2.175）。

2.173

按单元调用 call by location

见： 引址调用 call by reference（2.175）。

2.174

按名调用 call by name

传送参数的一种方法，在此方法中，调用模块向被调用模块提供表示要传送的参数的符号表达式，服务例程计算此表达式并向被调用的模块提供结果值。

注： 因为每次表达式计算的是被调用模块中所用的形式参数，参数的值在被调用模块执行期间是可以变更的。

相对： 引址调用 call by reference（2.175）和按值调用 call by value（2.176）。

2.175

引址调用 call by reference

按址调用 call by address

按单元调用 call by location

传送参数的一种方法，在此方法中，调用模块向被调用模块提供要传送的参数的地址。

注： 在这种方法中，被调用模块有能力变更由调用模块存储的参数的值。

相对： 按名调用 call by name（2.174）和按值调用 call by value（2.176）。

2.176

按值调用 call by value

传送参数的一种方法，在此方法中，调用模块向被调用模块提供要传送的参数的实际值。

注： 在此方法中，被调用模块不能变更由调用模块存储的参数的值。

相对： 按名调用 call by name（2.174）和引址调用 call by reference（2.175）。

2.177

调用图 call graph

调用树 call tree

排列图 tier chart

标识在系统或计算机程序中的模块并表示它们的调用关系的图（见图1）。

注： 结果不需要与在结构图中表示的相同。

参见： 控制流图 control flow diagram（2.339）、数据流图 data flow diagram（2.397）、数据结构图 data structure diagram（2.406）和状态图 state diagram（2.1570）。

相对： 结构图 structure chart（2.1605）。

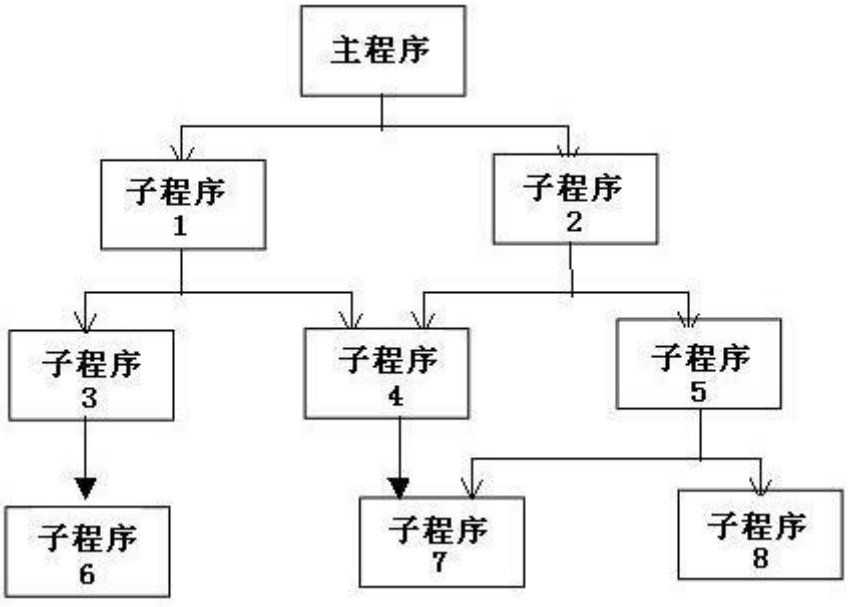

图4 调用图

2.178

调用表 call list

在调用软件模块中所用的自变量的有序列表。

2.179

调用踪迹 call trace

见： 子例程踪迹 subroutine trace（2.1619）。

2.180

调用树 call tree

见： 调用图 call graph（2.177）。

2.181

调用顺序 calling sequence

计算机指令的顺序，并且可能包括执行对另一模块调用必需的数据。

2.182

能力维 capability dimension

一组过程属性,它包括过程参考模型的能力方面和过程能力。

注：将属性条理化成能力等级,包括一个有序的过程能力度量。

2.183

能力成熟度模型 capability maturity model （CMM）

对软件组织在其不断改进的过程中，定义、实施、测量、控制和改进其软件过程所历经的进化阶段的描述。该模型便于软件组织确定其当前软件过程的能力和标识对改进软件质量和过程最关键的问题，从而指导软件组织选择过程改进策略。

2.184

势 cardinality

集合中元素的个数。

相对： 多重性势域 multiplicity（2.995）。

2.185

计算机辅助软件工程 CASE

Computer-Aided Software Engineering 的缩略语。

2.186

[分]情况语句 case

多路排它选择结构 multiple exclusive selective

单一入口、单一出口，其间有多路分支，对控制表达式的每个值，规定要执行的处理。并在所有情况下，返回控制至立即跟随在全部结构之后的语句（见图 5）。

参见： 多路包含选择结构 multiple inclusive selective（2.993）。

相对： 转至 go to（2.683）、跳转 jump（2.830）和“若-则-否则”语句 if-then-else（2.718）。

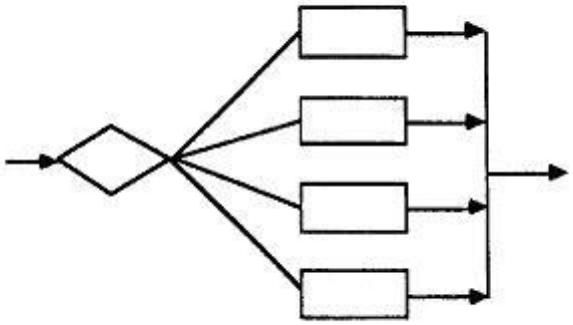

图5　case 结构

2.187

灾难性失效　catastrophic failure

关键软件的失效。

2.188

原因分析　causal analysis

为确定缺陷的根源所作的缺陷分析。

2.189

原因分析会议　causal analysis meeting

在完成特定任务后，为分析任务完成期间所暴露的缺陷而举行的会议。

2.190

配置控制委员会，变更控制委员会　CCB

Configuration Control Board 的缩略语。

Change Control Board 的缩略语。

2.191

关键设计评审　CDR

Critical Design Review 的缩略语。

2.192

认证　certification

a)　一个系统、部件或计算机程序符合其规定的需求，对操作使用是可接受的一种书面保证。例如，一计算机系统是安全的允许在定义的环境中操作的书面的认可；

b)　为使系统获准投入运行性使用，对系统遵循规定的需求是可接受的所做的正式演示；

c)　验证系统或部件遵循规定的需求，且其操作使用是可接受的过程。

2.193

链表　chained list

连接表　linked list

一种表，在这种表中各个项目可以是分散的，但每项都含有指出下一项位置的指针或标识符。

2.194

变更控制　change control

提议作一项变更并对其进行估计、同意或拒绝、调度和跟踪的过程。

见：　配置控制 configuration control（2.307）。

2.195

变更控制委员会 change control board

见： 配置控制委员会 configuration control board（2.308）。

2.196

变更转储 change dump

差异转储 differential dump

其内容自规定时间或事件以来已经变更的那些存储单元有选择性转储。

参见： 动态转储 dynamic dump（2.519）、内存转储 memory dump（2.928）、事后转储 postmortem dump（2.1153）、有选择转储 selective dump（2.1425）、快照转储 snapshot dump（2.1466）和静态转储 static dump（2.1582）。

2.197

过渡系统 change-over system

一种临时的信息处理系统，它用于为由某一运转系统到其后继系统的转换提供便利。

2.198

通道容量 channel capacity

在给定的通道上单位时间能传送信息的最大数量、通常用每秒多少位或波特率来测量。

参见： 内存容量 memory capacity（2.926）和存储容量 storage capacity（2.1593）。

2.199

箱图 Chapin chart

见： 盒图 box diagram（2.156）。

2.200

字符 character

用于表示信息的字母、数字或其他符号。

2.201

字符类型 character type

一种数据类型,它的成员能承担规定字符的值并能由字符操作符操作,例如,连接(concatenation)。

相对： 枚举型 enumeration type（2.558）、整型 integer type（2.784）、逻辑型 logical type（2.878）和实型 real type（2.1314）。

2.202

特性 characteristic

见： 数据特性 data characteristic（2.392）和软件特性 software characteristic（2.1475）。

2.203

检查，检出 checkout

在操作的或支持的环境中的测试举动，以保证软件产品在安装后按要求执行。

2.204

检查点 checkpoint

计算机程序中的一个点，在此点，检验或记录程序的状态（state）、状况（status）或结果。

2.205

主程序员 chief programmer

主程序员组的领导者、一高级程序员，它的职责包括产生赋予组的软件的关键部分，协调组的活动，评审其他组成员的的工作且对正在开发的软件有全面的技术了解。

参见： 后备程序员 backup programmer（2.121）和主程序员组 chief programmer team（2.206）。

2.206

主程序员组 chief programmer team

一软件开发组，它由一主程序员、一后备程序员、一秘书 / 图书管理员和附加的程序员，若需要，

还有专家组成，它使用设计的规程以增强组通信并优化每个成员的技巧。

参见： 后备程序员 backup programmer（2.121）、主程序员 chief programmer（2.205）和无我程序设计 egoless programming（2.536）。

2.207

子 child

在某一泛化联系中，对另一元素，即父的泛化。

参见： 子类 subclass（2.1613）和子类型 （2.1623subtype）。

相对： 父 parent（2.1107）。

2.208

配置项 CI

configuration item 的缩略语。

2.209

类[别] class

对共享相同属性、操作方法、联系和语义的各对象的集合的一种描述。类可使用一组接口来规定为其环境提供操作的汇集。

参见： 接口 interface（2.796）。

2.210

类目 classifier

描述结构和行为特征的一种机制。类目是类、接口数据类型和构件等建模元素的总称。

2.211

分类 classification

将对象指派给某一目的机制。

参见： 动态分类 dynamic classification（2.518）、多重分类 multiple classification（2.991）和静态分类 static classification（2.1581）。

2.212

类图 class diagram

一种图，它显示了由一组声明性（静态）模型元素（如类、类型等）以及它们的内容和彼此之间的关系所形成的集合。

2.213

清除 clear

将一变量、寄存器或其他存储单元设置为零、空白或其他空（null）值。

参见： 初始化 initialize（2.756）和复位 reset（2.1372）。

2.214

客户 client

向另一方请求服务的一方。

相对： 供方 supplier（2.1629）。

2.215

闭循环 closed loop

一种没有出口的循环，它的执行只能由循环所在的计算机程序或过程的外部的介入中断。

相对： “直至”语句 UNTIL（2.1795）、 “当”语句 WHILE（2.1847）。

2.216

闭式子例程 closed subroutine

存储在给定单元而不是在每次调用时拷贝至计算机程序中的子例程。

相对： 开式子例程 open subroutine（2.1052）。

2.217

配置管理 CM

configuration management 的缩略语。

2.218

[代]码，编码【动】 code

a) 在软件工程中，用编程语言或由汇编程序、编译程序或其他转换程序的正式输出表示的计算机指令和数据定义；

参见： 源代码 source code（2.1541 ）、目标代码 object code（2.1031）、机器代码 machine code（2.887）和微代码 microcode（2.945）。

b) 用编程语言表示计算机程序；

c) 赋予具体含义的字符或位模式。例如，状态码。

2.219

代码审核 code audit

由某人、某小组、或借助某种工具对源代码进行的独立的审查，其目的是验证其是否符合软件设计文件和程序设计标准。还可能对正确性和有效性进行估计。

参见： 审核 audit（2.105）、静态分析 static analysis（2.1577）、审查 inspection（2.764）和走查 walk-through（2.1843）。

2.220

代码断点 code breakpoint

控制断点 control breakpoimt

给定的计算机指令执行时启动的断点。

参见： 动态断点 dynamic breakpoint（2.516）、跋断点 epilog breakpoint（2.559）、可编程断点 programmable breakpoint（2.1249）、序断点 prolog breakpoint（2.1268）和静态断点 static breakpoint（2.1580）。

相对： 数据断点 data breakpoint（2.391）。

2.221

代码生成（器）程序 code generator

源代码生成（器）程序 source code generator

a) 一个例程，常常属于编译程序的一部分，它把计算机程序从某种中间级表示（通常为语法分析程序的输出）变换成较为低级的表示，如程序执行的机器代码；

b) 一种软件工具，它接受计算机程序的需求或设计作为输入并产生实现需求或设计的源代码。

参见： 应用程序生成（器）程序 application generator（2.68）。

2.222

代码审查 code inspection

参见： 审查 inspection（2.764）。

2.223

职业道德条文标准 code of ethics standard

一种标准，它描述处理在职业中或职业间能接受的行为标准的一组道德原则的特征。

2.224

代码评审 code review

把软件代码呈现给项目人员、管理人员、用户、客户或其他感兴趣的人员用于评论或批准的会议。

相对： 设计评审 design review（2.452）、形式合格评审 formal qualification review（2.646）、需求评审 requirements review（2.1366）和测试就绪评审 test readiness review（2.1717）。

2.225

代码踪迹 code trace

见： 执行踪迹 execution trace（2.582）。

2.226

代码走查 code walk-through

参见： 走查 walk-through（2.1843）。

2.227

编码 coding

a) 在软件工程中，用编程语言表示计算机程序的过程。

b) 逻辑和数据从设计规格说明（设计描述）转换为编程语言。

参见： 软件开发过程 software development process（2.1491）。

2.228

内聚度（内聚性） cohesion

模块强度 modual strength

单个程序模块所执行的诸任务在功能上的互相关联的程度。其种类包括一致的（coincidental）、通信的（communicational）、功能的（functional）、逻辑的（logical）、过程的（procedural）、顺序的（sequential）和暂时的（temporal）。

相对： 耦合度 coupling（2.358）。

2.229

一致内聚度 coincidental cohesion

内聚度的一种类型，在其中，由软件模块执行的任务与其他的无关。

相对： 通信内聚度 communicational cohesion（2.246）、功能内聚度 functional cohesion（2.661）、逻辑内聚度 logical cohesion（2.874）、过程内聚度 procedural cohesion（2.1179）和顺序内聚度 sequential cohesion（2.1439）。

2.230

协作 collaboration

对某一操作或类目（例如一个用况），如何通过以特定方式扮演特定角色的一组类目与关联来实施的规格说明。协作定义了一个交互。

参见： 交互 interaction（2.790）。

2.231

协作图 collaboration diagram

一种图，它表示围绕某一模型的结构用类目与关联或者实例与链接所组织的交互。与时序图不同的是，协作图展现各实例间的联系。时序图与协作图所表达的信息相似，但展现的方式不同。

参见： 时序图 sequence diagram（2.1437）。

2.232

命令 command

能输入至计算机系统以启动一动作或影响计算机程序的执行的一个表达式。例如，命令”log on”启动一计算机会话。

2.233

命令驱动的 command-driven

用于说明一系统或操作方式，在其中，用户通过命令指挥系统。

相对： 选单驱动的 menu-driven（2.931）。

2.234

命令语言 command language

一组过程性的操作符及与之有关的语法（一种语言），用来表示计算机系统的命令（指明交给操作系统执行的功能）。

参见： 命令驱动 command-driven（2.233）。

2.235

注释 comment

a) 嵌入至计算机程序、命令语言、作业控制语句或数据之间的说明信息，其目的在给读者提供澄清性材料，并不影响机器的解释工作；

b) 加到或散置在源语言语句当中的描述、附注或解释，在目标语言中这些是无效的。

2.236

承诺 commitment

一个可视的、并希望各方遵守的协议。

2.237

公共 common

见： 公共存储 common storage（2.245）。

2.238

公共区域 common area

见： 公共存储 common storage（2.245）。

2.239

公共块 common block

见： 公共存储 common storage（2.245）。

2.240

（缺陷的）共同原因 common cause (of a defect)

过程或系统固有的缺陷的原因。它影响过程的每个输出和工作在此过程中的每个人。

参见： 特殊原因 special case (of a defect)（2.1545）。

2.241

公共耦合 common coupling

见： 公共环境耦合 common-environment coupling（2.243）。

2.242

公共数据 common data

见： 全局数据 global data（2.680）。

2.243

公共环境耦合 common-environment coupling

公共耦合 common coupling

耦合度的一种类型，在其中，两个软件模块访问公共数据区域。

相对： 内容耦合 content couping（2.327）、控制耦合 control coupling（2.336）、数据耦合 data couping（2.393）、混合耦合 hybrid coupling（2.714）和相依耦合 pathological coupling（2.1123）。

2.244

共同特征 common features

能力成熟度模型关键过程域的细分的类别。共同特征是指示一个关键过程域的实施和制度化是否是有效、可重复和持久的属性。能力成熟度模型的共同特征如下：

a) 执行承诺：组织为保证过程得以建立和持续下去而必须采取的行动。执行承诺一般涉及组织方针的制定和高层管理者的支持；

b) 执行能力：为了切实有效地实施软件过程，项目组或组织必须具备的先决条件。执行能力一般涉及资源、组织结构和培训；

c) 执行活动：对为实施关键过程域所必需的角色和规程的描述。执行活动一般涉及制定计划和规程，进行工作，跟踪它并在必要时采取纠正措施；

d) 测量分析：对过程进行测量和对测量结果进行分析所作的必要描述。测量和分析一般包括

可能采取的旨在确定执行活动的状态和有效性的测量例子；

e) 验证实施：确保活动按照已建立的过程进行的措施。验证一般包括管理者和软件质量保证组所做的评审和审核。

2.245

公共存储 common storage

公共区域 common area

公共块 common block

能由软件系统中两个或更多个模块访问的主存储器的一部分。

参见： 全局数据 global data（2.680）。

2.246

通信内聚度 communicational cohesion

内聚度的一种类型，在其中，由软件模块执行的任务使用相同的输入数据或产生相同的输出数据。

相对： 一致内聚度 coincidental cohesion（2.229）、功能内聚度 functional cohesion（2.661）、逻辑内聚度 logical cohesion（2.874）、过程内聚度 procedural cohesion（2.1179）和顺序内聚度 sequential cohesion（2.1439）。

2.247

紧缩 compaction

在微程序编程中，转换一微程序为功能等效但比原始的更快或更短的微程序的过程。

参见： 局部紧缩 local compaction（2.869）和全局紧缩 global compaction（2.679）。

2.248

比较器 comparator

用来比较两个计算机程序、文件或数据集合的一种软件工具，目的是找出其共同点或不同的地方。比较的典型对象是源代码、目标代码、数据库文件的相似版本或测试结果。

2.249

兼容性 compatibility

a) 两个或两个以上系统或部件，当共享相同的硬件或软件环境执行它们所要求的功能可得到同样结果的能力；

b) 两个或两个以上系统或部件交换信息的能力。

参见： 互操作性 interoperability（2.807）。

2.250

兼容的评估模型 compatible assessment model

用于执行评估的可操作的模型，其满足已定义的、与参考模型一致的要求（模型的目的、范围、要素和标志、对参考模型的映射及结果解释）。

2.251

称职的评估师 competent assessor

某个具有执行过程评估的技能、能力资格和经验的个人。

2.252

编译 compile

将高级语言程序变换成与之等价的浮动的或绝对的机器代码。

相对： 汇编 assemble（2.79）、反编译 decompile（2.418）和解释 interpret（2.808）。

2.253

编译并执行 compile-and-go

一种操作技术，用此技术在计算机程序的编译、连接、装入和执行之间没有停顿。

2.254

编译程序 compiler

用于进行编译的一种计算机程序。

参见： 交叉编译程序 cross-compiler（2.373）、增量编译程序 incremental compiler（2.737）和根编译程序 root compiler（2.1401）。

相对： 汇编程序 assembler（2.82）和解释程序 interpreter（2.809）。

2.255

编译程序代码 compiler code

用编译程序能识别和处理的形式表示的计算机指令和数据定义。

相对：汇编代码 assembly code(2.85)、解释代码 interpretive code(2.810)和机器代码 machine code（2.887）。

2.256

编译程序的编译程序 compiler compiler

见： 编译程序生成器 compiler generator（2.257）。

2.257

编译程序的生成程序 compiler generator

编译程序的编译程序 compiler compiler

元编译程序 meta-compiler

用来构造编译程序的一部分或全部的翻译程序或解释程序。

2.258

编译时间 compile time

在某一软件模块的编译期间所出现的某一事物。

参见： 建模时间 modeling time（2.970）和运行时间 running time（2.1408）。

2.259

完代码 completion code

用批程序与作业流处理器通信的代码，以影响在输入流中后续步的执行。

2.260

复杂性 complexity

系统或部件设计或实现难于理解和验证的程度，由下述因素确定，如：接口的数量和错综程度，条件转移的数量和错综程度，嵌套的深度，数据结构的类型，以及其他一些系统特性。

相对： 简单性 simplicity（2.1452）。

2.261

部件，构件 component

a) 构成系统的部分（系统或程序的基本部分）。部件可以是硬件或软件且可以进一步划分为其他部件；

注： 术语“模块（module）”、“部件（component）”和“单元（unit）”常常可以互换使用或在不同的方法中，定义作为另一个的子元素，取决于上下文。这些术语的关系尚未标准化。

b) 成熟的、可重用的部件。

2.262

构件图 component diagram

一种图示各构件之间的组织与依赖关系的图。

2.263

构件标准 component standard

描述数据或程序的构件特征的标准。

2.264

部件测试 component testing

模块测试 module testing

对独立的硬件或软件部件或有关部件组的测试。

参见： 集成测试 integration testing（2.788）、接口测试 interface testing（2.800）、系统测试 system testing（2.1669）和单元测试 unit testing（2.1792）。

2.265

组合[类] composite [class]

通过一个组合关系与一个或多个类相关联的类。

参见： 组合 composition（2.269）。

2.266

组合聚合 composite aggregation

见： 组合 composite（2.265）。

2.267

组合状态 composite state

一种由并发（正交）子状态或顺序（互斥）子状态所组成的状态。

参见： 子状态 substate（2.1620）。

2.268

复合（组合）类型 composite type

一种数据类型，它的每个成员由多个数据项构成。例如，一种称为 PAIRS 的数据类型，它的成员是有序的对。

相对： 原子类型 atomic type（2.103）。

2.269

组合 composition

组合聚合 composite aggregation

聚合的一种形式，它要求做为实例在一个时刻最多包含在一个组合之中，而且该组合对象负责其组成部分的创建与撤消。组合可以递归。

2.270

计算机 computer

a) 能执行大量计算，包括许多算术运算和逻辑运算，而在运行期间无需操作员干预的一种功能装置；

b) 由一台或多台相联的处理机和外围设备组成的一种可编程序的功能装置，这种装置由内部存储的程序控制，可执行大量的计算（许多算术运算和逻辑运算）而无需人的干预。

2.271

计算机辅助软件工程 computer-aided software engineering(CASE)

在软件工程中起辅助作用的计算机的使用。可以包括软件工具的应用程序到软件设计、需求踪迹、代码生成、测试、文档生成和其他软件工程活动。

2.272

计算机数据 computer data

计算机设备和计算机设备之间或计算机设备内部通信用的数据。这种数据可以是外部的（计算机可读形式），也可以是驻留在计算机设备内的，可以是模拟信号，也可以是数字信号。

2.273

计算机语言 computer language

设计用于允许人与计算机通信的语言。

参见： 设计语言 design language（2.447）、查询语言 query language（2.1303）和编程语言 programming language（2.1252）。

2.274

计算机网络 computer network

由两个或两个以上按一定的协议互连的计算机所组成的复合体。

2.275

计算机性能评价 computer performance evaluation

一种功能训练，它测量计算机系统的性能和研究改进性能的方法。

参见： 系统特征（轮廓） system profile（2.1661）、吞吐量 throughput（2.1735）、利用率 utilization（2.1817）和工作量模型 workload model（2.1857）。

2.276

计算机程序 computer program

计算机指令和数据定义的组合，它允许计算机硬件执行计算或控制功能。

参见： 软件 software（2.1469）。

2.277

计算机程序摘要 computer program abstract

对计算机程序的简短叙述，它为用户提供足够的信息，使他们能据此确定该计算机程序是否适合其需要及所拥有的资源。

2.278

计算机程序注解 computer program annotation

参见： 注释 comment（2.235）。

2.279

计算机程序认证 computer program certification

参见： 认证 certification（2.192）。

2.280

计算机程序部件 computer program component(CPC)

见： 计算机软件部件 computer software component（2.289）。

2.281

计算机程序配置标识 computer program configuration identification

参见：配置标识 configuration identification（2.309）。

2.282

计算机程序配置项 computer program configuration item(CPCD)

见： 计算机软件配置项 computer software configuration item（2.290）。

2.283

计算机程序开发计划 computer program development plan

参见： 软件开发计划 software development plan（2.1490）。

2.284

计算机程序确认 computer program validation

参见： 确认 validation（2.1819）。

2.285

计算机程序验证 computer program verification

参见： 验证 verification（2.1826）。

2.286

计算机资源分配 computer resource allocation

计算机资源赋予当前和等待的作业、例如，主内存、输入／输出设备和辅助存储器赋予在计算机系统中并发执行的作业。

参见： 动态资源分配 dynamic resource allocation（2.522）和存储分配 storage allocation

（2.1591）。

2.287

计算机资源 computer resource

为一给定的目的可用的计算机装备、程序、文档、服务、设施、供给和个人。

参见： 计算机资源分配 computer resource allocation（2.286）。

2.288

计算机安全 computer security

保护数据和资源防止偶然的或恶意的动作的破坏，这些动作可以是丢失、或非授权的修改、破坏、访问、泄露或采集。

2.289

计算机软件部件 computer software component(CSC)

计算机软件配置项的功能的或逻辑的显著的部分，典型地是两个或多个软件单元的集合。

2.290

计算机软件配置项 computer software configuration item(CSCI)

为配置管理设计的软件的集合，它在配置管理过程中，作为单个实体对待。

参见： 配置项 configuration item（2.311）。

相对： 硬件配置项 hardware configuration item（2.691）。

2.291

计算机系统 computer system

由一台或多台计算机和相关软件组成的一种功能装置（系统）。

2.292

计算机字 computer word

见： 字 word（2.1850）。

2.293

计算中心 computer center

设计为各种用户通过计算机和硬件操作以及通过其雇员的服务以提供计算机服务的设施。

2.294

概念阶段 concept phase

a) 软件开发生存期中的时段，在此时段中，描述用户需求并通过文档（例如，说明需求、预先的计划报告、项目启动备忘录、可行性研究、系统定义以及相关于项目的文档、管理、过程和策略）进行评价。

b) 软件开发项目的初始阶段，在此时段中，通过文档（例如，说明需求、先进的计划报告、项目启动备忘录、可行性研究、系统定义以及关于项目的文档、管理、过程和策略）对用户需求进行描述和评价。

2.295

概念系统设计 conceptual system design

涉及规定系统组织的逻辑方面、过程和全系统的信息流程的一种系统设计活动。

2.296

具体类 concrete class

能够直接用来创建实例的类。

2.297

并发性 concurrency

在同一时间间隔内多个活动的出现。并发能通过多个线程的交错或同时执行来获得。

参见： 线程 thread（2.1732）。

2.298

并发的 concurrent

并行的 parallel

用于说明两个或多个活动在同一时间间隔内发生，指交错的活动或或同时执行的活动。

相对： 同时的 simultaneous（2.1455）。

2.299

并发进程 concurrent processes

可以同时地在多处理机上执行或异步地在单处理机上执行的若干进程。各并发进程可以相互作用，一个进程在接受另一进程的信息之前或一外部事件出现之前可以把执行挂起。

相对： 顺序进程 sequential processes（2. 1441）。

2.300

并发子状态 concurrent substate

一种能与包含在同一组合状态中的另一子状态同时保有的子状态。

参见： 组合状态 composite state（2.267）。

相对： 互斥子状态 disjoint substate（2.496）。

2.301

条件码 condition code

见： 状态码 status code（2.1585）。

2.302

条件分支 conditional branch

见： 条件跳转 conditional jump（2.304）。

2.303

条件控制结构 conditional control structure

一种程序设计控制结构，它允许程序中使用根据指定条件的满足情况而加以选择的控制流。例如，按情况、若……则……否则……。

2.304

条件跳转 conditional jump

只有当规定条件满足时，才发生的转移。

相对： 无条件跳转 unconditional jump（2.1785）。

2.305

配置 configuration

a) 计算机系统、部件或网络按照其功能部件的特点、数量、主要特性和交互连接而定义的安排。具体地讲，配置一词可以指硬件配置或软件配置；

b) 在配置管理中，在技术文档中制定的并在产品中体现的硬件、软件的功能和（或）物理特性。参见： 配置项 configuration item（2.311）、形式、适合和功能 form,fit,and function（2.643）和版本 version（2.1829）。

2.306

配置审核 configuration audit

对所要求的全部配置项均已产生出来，当前的配置与规定的需求相符所作的证明。技术文件说明书完全而准确地描述了各个配置项目，并且曾经提出的所有变更请求均已得到解决的过程。

见： 功能配置审核 functional configuration audit（2.662）和物理配置审核 physical configuration audit（2.1139）。

2.307

配置控制 configuration control

变更控制 change control

配置管理的元素，它由评价、协调、批准或不批准和在配置项的配置标识正式建立以后，配置项变更的实现组成。

参见： 配置控制委员会 configuration control board（2.308）、偏离 deviation（2.473）、工程变更 engineering change（2.546）、接口控制 interface control（2.796）、版本通知 notice of revision（2.1028）、规格说明变更通知 specification change notice（2.1549）和放弃 waiver（2.1842）。

相对：配置标识 configuration identification（2.309）和配置状态报告 configuration status accounting（2.315）。

2.308

配置控制委员会 configuration control board

变更控制委员会 change control board

对提出的配置项的变更负责进行评价、批准或不批准，并保证批准的变更的实现的权力机构（一组人）。

参见： 配置控制 configuration control（2.307）。

2.309

配置标识 configuration identification

a) 配置管理的元素，它由为系统选择配置项并在技术文档中记录它们的功能和物理特征组成；

相对： 配置控制 configuration control（2.307）和配置状态报告 configuration status accounting（2.315）。

b) 对于配置项当前批准的技术文档如在规格说明、描述、相关清单和引用的文档中阐明的那样。

参见： 分配的配置标识 allocated configuration identification（2.58）、功能的配置标识 functional configuration identification（2.663）、产品配置标识 product configuration identification（2.1215）和基线 baseline（2.125）。

2.310

配置索引 configuration index

在配置管理中所用的文档，它提供做成产品的配置项的报告。

参见： 配置项开发记录 configuration item development record（2.312）和配置状态报告 configuration status accounting（2.315）。

2.311

配置项 configuration item(CI)

为配置管理设计的硬件、软件或两者的集合，它在配置管理过程中作为一单个实体来对待。

参见：硬件配置项 hardware configuration item（2.691）、计算机软件配置项 computer software configuration item（2.290）、配置标识 configuration identification（2.309）和关键项 critical item（2.366）。

2.312

配置项开发记录 configuration item development record

在配置管理中使用的描述配置项基于配置审核和设计评审的结果的开发状态的文档。

参见： 配置索引 configuration index（2.310）和配置状态报告 configuration status accounting（2.315）。

2.313

配置管理 configuration management(CM)

应用技术的和管理的指导和监控方法以标识和说明配置项的功能和物理特征，控制这些特征的变更，记录和报告变更处理和实现状态并验证与规定的需求的遵循性。

参见： 基线 baseline（2.125）、配置标识 configuration identification（2.309）、配置控制

configuration control（2.307）、配置状态报告 configuration status accounting（2.315）和配置审核 configuration audit（2.306）。

2.314

配置管理库系统 configuration management library system

存取软件基线库内容的工具和规程。

2.315

配置状态记录 configuration status accounting

一种配置管理的元素，它由记录和报告为有效地管理某一配置所需的信息组成。此信息包括列出经批准的配置标识表、建议变更的配置状态和经批准变更的实现状态。

参见： 配置索引 configuration index（2.310）和配置项开发记录 configuration item development record（2.312）。

相对： 配置控制 configuration control（2.307）和配置标识 configuration identification（2.309）。

2.316

配置单元 configuration unit

可放入配置管理库系统和从库中检索的一个配置项或部件的最低层次的实体。

2.317

禁闭 confinement

a) 在被核准访问期间，防止对数据做未经核准的变更、使用、破坏和抛弃的措施。

参见： 完整性 integrity（2.789）；

b) 对程序和进程施加的限制，其目的是使它们不能访问或影响未经核准的数据、程序或进程。

2.318

连接 connection

a) 程序的某一部分对程序另一部分的标识符（即，在另外地方发现的标识）的引用。

参见： 接口 interface（2.795）；

b)为了传递信息而在功能部件之间建立的关系。

2.319

接续的 consecutive

关于两个顺序的事件或项的发生，而没有任何其他事件或项的插入、即，一个立即跟在另一个之后。

2.320

一致性 consistency

在文档或系统或部件的各部分之间，一致、标准化、无矛盾的程度。

参见： 可踪迹性 traceability（2.1752）。

2.321

常量 constant

值不能变更的数量或数据项。例如数据项 FIVE，具有不变值 5。

参见： 数字形式常数 figurative constant（2.624）和文字的 literal（2.862）。

相对： 变量 variable（2.1822）。

2.322

定失效率周期 constant-failure period

在系统或部件生存期中的时间周期，在此周期中，硬件失效几乎按一致的比例发生。

参见： 浴盆曲线 bathtub curve（2.129）。

相对： 早期失效周期 early-failure period（2.526）和磨损失效周期 wearout-failure period（2.1845）。

2.323

约束 constraint

一种语义条件或限制。在 UML 中，某些约束是预定义的，另一些则是用户定义的。约束是 UML 中三种可扩展性机制之一。

参见： 加标值 tagged value（2.1672）和衍型 stereotype（2.1588）。

2.324

构造的能力 constructed capability

由组织单位的元素或不同组织的元素所构成的能力，为满足某些特殊要求的目的而组合。

2.325

容器 container

a) 一种类型包含其他实例而存在，并提供访问或迭代其内容的操作的实例（例如：数组、列表和集合）；

b) 一种为包容其他构件而存在的构件。

2.326

包容层次 containment hierarchy

由模型元素与其间存在的包容联系所组成的一种命名空间层次。包容层次形成一个图。

2.327

内容耦合 content coupling

耦合的一种类型，其中，一个软件模块的部分或全部内容包括在另一模块的内容中。

相对： 公共环境耦合 common environment coupling（2.243）、控制耦合 control coupling（2.336）、数据耦合 data coupling（2.393）、混合耦合 hybrid coupling（2.714）和相依耦合 pathological coupling（2.1123）。

2.328

上下文，语境 context

a) 进程在走走停停的运行过程中需要保存（在进程退出运行时）和恢复（在进程进入运行时）的进程环境；

b) 用于特定目的（例如：规定某一操作）的有关建模元素的集合的一种视图。

2.329

邻接分配 contiguous allocation

一种存储分配技术，存储的程序或数据分配给一等于或大于其尺寸的存储块，所以，逻辑上邻接的程序或数据被赋予物理邻接的存储单元。

相对： 调页 paging（2.1101）。

2.330

偶然性因素 contingency factor

对规模、成本或进度计划的一种调整(增加)，以便考虑由于不完全的规格说明以及在应用领域上缺乏估计经验等而可能对这些参数估计过低。

2.331

不断迭代 continuous iteration

没有出口的循环。

2.332

合同 contract

通过法律约束当事双方的一个协议，或是在一个机构内部为了提供服务的一个内部协议，该协议提供的服务适用于一个系统或系统一部分的供应、开发、生产、操作或维护。

2.333

合同条款和条件 contract terms and conditions

合同规定的法律、财务和管理方面的内容。

2.334

合同要求的审核 contractually required audit

合同要求的审核，它一般由需方或由独立的机构主持进行。此过程对产品或服务提供一个独立的评价，以决定产品或服务是否符合它们的需求。

2.335

控制断点 control breakpoint

见： 代码断点 code breakpoint（2.220）。

2.336

控制耦合 control coupling

软件模块为了显式目的至另一个模块的通信信息，影响另一个模块的执行的一种耦合类型。

相对： 公共环境耦合 common environment coupling（2.243）、内容耦合 content coupling（2.327）、数据耦合 data coupling（2.393）、混合耦合 hybrid coupling（2.714）和相依耦合 pathological coupling（2.1123）。

2.337

控制数据 control data

选择一程序中的操作方式或子方式，给顺序流指向，或者直接影响软件操作的数据。例如，循环控制变量（loop control variable）。

2.338

控制流 control flow

控制流 flow of control

在计算机程序执行期间操作执行的顺序。

相对： 数据流 data flow（2.396）。

2.339

控制流图 control flow diagram

描述在系统或程序执行期间，可能执行的所有可能的顺序集的图。

类型包括： 盒图 box diagram（2.156）、流程图 flowchart（2.637）、输入-处理-输出图 input-process-output（2.763）和状态图 state diagram（2.1570）。

参见： 调用图 call graph（2.177）和结构图 structure chart（2.1605）。

相对： 数据流图 data flow diagram（2.397）。

2.340

控制流踪迹 control flow trace

见： 执行踪迹 execution trace（2.582）。

2.341

控制语言 control language

见： 作业控制语言 job control language（2.826）。

2.342

控制程序 control program

见： 监督程序 supervisory program（2.1628 条）。

2.343

控制语句 control statement

可在可替换程序语句集之间选择，或影响操作执行顺序的程序语句。例如，if-then-else、case 语句。

相对： 赋值语句 assignment statement（2.99）、声明语句 declaration statament（2.416）。

2.344

控制存储器 control store

在微程序计算机中，微程序驻留的计算机存储器。

参见： 微字 microword（2.953）和超微存储 nanostore（2.1011）。

2.345

控制结构 control structure

通过计算机程序决定控制流的构造。

参见： 条件控制结构 conditional control structure（2.303）。

2.346

控制变量 control variable

见： 循环控制变量 loop －control variable（2.882）。

2.347

约定 conventions

一种要求，它用于规定遵守纪律的统一的方法，以在软件产品中提供一致性。即，为了安排数据的统一的模式或格式。

参见： 实践 practices（2.1156）和标准 standards（2.1562）。

2.348

会话式[的] conversational

用于说明操作的一种交互系统或方式，其中，用户与系统之间的交互，类似于人机对话。

参见： 交互的 interactive（2.792）、联机的 on-line（2.1045）和实时的 real time（2.1313）。

相对： 批命令 batch（2.128）。

2.349

会话式编译程序 conversational compiler

见： 增量编译程序 incremental compiler（2.737）。

2.350

转换 conversion

对现有软件进行修改，使之在不同环境工作时能具有等同的功能，例如，把一个程序从 FORTRAN 变换成 Ada。把在一台计算机上运行的程序变换成能在另一台计算机上运行的程序。

2.351

拷贝【动】【名】，复制【动】，副本【名】 copy

a) 从源读数据，源数据保持不变，以与源可以不同的物理格式，写相同的数据至别处。例如，从磁盘复制数据至磁带上；

相对：移动 move（2.983）。

b) 如 a)中的复制过程的结果。例如，数据文件的副本。

2.352

核心转储 core dump

见： 内存转储 memory dump（2.928）。

2.353

协同例程 coroutines

在上次操作暂停点开始执行的例程且不要求返回控制至调用它的程序或子程序。

相对： 子例程 subroutine（2.1619）。

2.354

纠正性维护 corrective maintenance

为改正硬件或软件的故障而进行的维护。

参见： 软件维护 software maintenance（2.1507）。

相对： 适应性维护 adaptive maintenance（2.43）和完善性维护 perfective maintenance（2.1130）。

2.355

正确性 correctness

a) 系统或部件在它的规格说明、设计和实现中无故障的程度；

b) 软件、文档或其他项满足规定的需求的程度；

c) 软件、文档或其他项满足用户需要和期望的程度（不管是否规定）。

2.356

正确性证明 correctness proof

见： 正确性的证明 proof of correctness（2.1270）。

2.357

计数器 counter

用于记录在计算机程序执行期间，给定的事件发生数的变量。例如，记录循环执行的时间数的变量。

2.358

耦合 coupling

在软件模块之间相互依赖的方式和程度。类型包括公共环境耦合（common environment coupling）、内容耦合（content coupling）、控制耦合（control coupling）、数据耦合（data coupling）、混合耦合（hybrid coupling）和相依耦合（pathological coupling）。

相对： 内聚度 cohesion（2.228）。

2.359

课件 courseware

一种计算机软件，它用于课程教学或自学。是计算机辅助教学软件的一个组成部分。

2.360

计算机程序部件 CPC

computer program component 的缩略语。

见： 计算机软件部件 computer software component（2.289）。

2.361

计算机程序配置项 CPCI

computer program configuration item 的缩略语。

见： 计算机软件配置项 computer software configuration item（2.290）。

2.362

崩溃 crash

计算机系统或部件的突然的和完全地失效。

2.363

关键的，临界的 critical

系指：

a) 由于设计不当，一个系统或一个软件的某些环节或部分在运行时超出了临界范围，或存在着潜在的、未检测出的错误，会导致死机、人员伤害、任务失效、数据丢失、财经上的损失或灾难性的设备损坏等严重后果。或指：

b) 要使用的软件开发技术的成熟程度和有关的风险。

2.364

关键计算机资源 critical computer resource (CCR)

由于对计算机资源的潜在需求有可能超出可用量而可能成为项目风险源的计算资源参数。例如，目标计算机的内存容量和宿主计算机的磁盘空间。

2.365

关键设计评审 critical design review(CDR)

a) 一种评审，它指导验证一个或多个配置项的详细设计满足规定需求、在配置项和设备、设施、软件或个人的其他项之间建立兼容性、对于每一个配置项评定风险区域、（若有可能）评定可生产性分析的结果、评审初始硬件产品规格说明、评价初始测试计划和评价初始操作和支持文档是否足够；

参见： 概要设计评审 preliminary design review（2.1164）和系统设计评审 system design review（2.1650）。

b) 任何硬件或软件部件像 a)中那样的评审。

2.366

关键项 critical item

在配置管理中，一个配置项，它因为特殊的工程或逻辑的考虑，要求批准的规格说明，以在部件级建立技术的或库存的控制。

2.367

关键路径 critical path

为使整个项目按进度进行，必须按计划完成的项目的一系列相互依赖的任务。

2.368

关键部分优先 critical piece first

软件开发的一种途径。它首先把注意力集中在软件系统中最关键部分的实现。关键部分可以根据所提供的服务、风险程度、困难程度或其他一些准则来确定。

参见： 自底向上 bottom-up（2.153）和自顶向下 top-down（2.1746）。

2.369

临界区 critical region

见： 临界段 critical section（2.370）。

2.370

临界段，关键段 critical section

将要被执行的一段代码。其执行与另一关键段的代码的执行是互斥的。如果一些代码段竞相使用一计算机资源和数据项时，就要求这些段互斥地执行。

2.371

关键软件 critical software

它的失效会在安全性上有重大的冲击或可能在财政上或社会上引起重大的损失的软件。

2.372

关键性 criticality

严重性 severity

需求、模块、错误、故障、失效或其他项对系统的开发和操作冲击的程度。

2.373

交叉汇编程序 cross-assembler

在一台计算机上为另一台不同的计算机产生目标代码的汇编程序。

2.374

交叉编译程序 cross-compiler

在一台计算机上为另一台不同计算机产生汇编代码或目标代码的编译程序。

2.375

交叉引用生成器 cross-reference generator

交互引用器 cross-referencer

一种软件工具，它接受计算机程序的源代码作为输入，并输出标识每个程序的变量、标记和其他标

识符并指示程序中哪个语句定义、设置或使用它们的清单。

2.376

交叉引用清单 cross-reference list

标识计算机程序中的每一个变量、标记和其他标识符并指示程序中的哪个语句定义、设置或使用它们的清单。

2.377

交叉引用器 cross-referencer

见： 交叉引用生成器 cross-reference generator（2.375）。

2.378

计算机软件部件 CSC

computer software component 的缩略语。

2.379

计算机软件配置项 CSCI

computer software configuration item 的缩略语。

2.380

插进指令 cue

见： 调用 call（2.171）。

2.381

课程标准 curriculum standard

描述由教育机构提供的知识体的研究课程的特征的标准。

2.382

顾客 customer

供方提供的产品的接收者。

注 1： 在合同上，顾客称为采购方。

注 2： 顾客也可能是诸如，最终顾客、用户、受益者或采购者。

注 3： 顾客可以是来自于组织的内部或外部。

2.383

切换 cutover

在给定的瞬间，某一系统到其后继系统的功能转移。

2.384

周期 cycle

a) 一种时间阶段，期间完成一组事件；

参见： 软件开发周期 software development cycle（2.1483）和软件生存期 software life cycle（2.1506）。

b) 以相同的顺序有规律地重复的一组操作，在每次重复时可能有些变化。例如，一计算机读周期。

参见： 传递 pass（2.1115）。

2.385

周期窃取 cycle stealing

中央处理单元暂停一个或多个周期，允许发生其他操作，例如，从主存储器传送数据以响应从输入输出控制器来的输出请求的过程。

2.386

循环搜索 cyclic search

一种存储分配技术，在其中，每次搜索适当的存储块从上一次最后分配的块之后开始。

2.387

数据 data

a) 事实、概念或指令以适合于由人或自动装置进行通信、解释或处理的方式的表示；

参见： 数据类型 data type（2.408）、计算机数据 computer data（2.272）、控制数据 control data（2.337）、出错数据 error data（2.564）、可靠性评价 reliability evaluation（2.1337）和软件经验数据 software experience data（2.1500）。

b) 有时用作文档 documentation（2.496）的同义词。

2.388

数据抽象 data abstraction

a) 由定义数据类型和它们的功能特征忽视它们的详细表示以抽取数据的基本特征的过程；

参见： 封装 encapsulation（2.542）和信息隐藏 information hiding（2.751）。

b) 在 a)中过程的结果。

2.389

数据分析 data analysis

对实际的或计划好的系统中的数据及其流程的一种系统性调研。

2.390

数据库，数据基 database

a) 一数据集，或一数据集的部分或全体，它至少包括足够为一给定目的或给定数据处理系统使用的一个文件；

b) 对一系统来说是基本的数据集合。

2.391

数据断点 data breakpoint

存储断点 storage breakpoint

当访问规定的数据项时启动的断点。

参见： 动态断点 dynamic breakpoint（2.516）、跋断点 epilog breakpoint（2.559）、可编程的断点 programmable breakpoint（2.1249）、序断点 prolog breakpoint（2.1268）和静态断点 static breakpoint（2.1580）。

相对：代码断点 code breakpoint（2.220）。

2.392

数据特性 data characteristic

固有的、可能非本质的特色、品质和数据的属性（例如，到达的速度、格式、值范围或字段值之间的关系）。

2.393

数据耦合 data coupling

输入-输出耦合 input-output coupling

一个软件模块的输出作为另一个模块的输入的一种耦合类型。

相对： 公共环境耦合 common environment coupling （2.243）、内容耦合 content coupling（2.327）、控制耦合 control coupling（2.336）、混合耦合 hybrid coupling（2.714）和相依耦合 pathological coupling（2.1123）。

2.394

数据字典 data dictionary

a) 软件系统中使用的所有数据项的名字及与这些数据项有关的特性（例如，数据项长度、表示等）的集合；

b) 分层数据流程图中涉及的数据流、数据元素、文件、数据基和进程之定义的集合。

2.395

数据异常　data exception

当程序企图不正确地使用或访问数据时发生的异常。

参见：　地址异常 address exception（2.50）、操作异常 operation exception（2.1059）、溢出异常 overflow exception（2.1082）、保护异常 protection exception（2.1274）和下溢异常 underflow exception（2.1786）。

2.396

数据流　data flow

在计算机程序执行期间，执行数据传送、使用和转换的顺序。

相对：　控制流 control flow（2.338）。

2.397

数据流图　data flow diagram(DFD)

数据流程图　data flowchart

数据流图　data flow graph

描述数据源、数据接收端、数据存储和在数据上如节点上执行的处理以及数据的逻辑流如节点之间的连接的一种图（见图 6）。

相对：　流程图 flow diagram（2.635）和数据结构图 data structure diagram（2.406）。

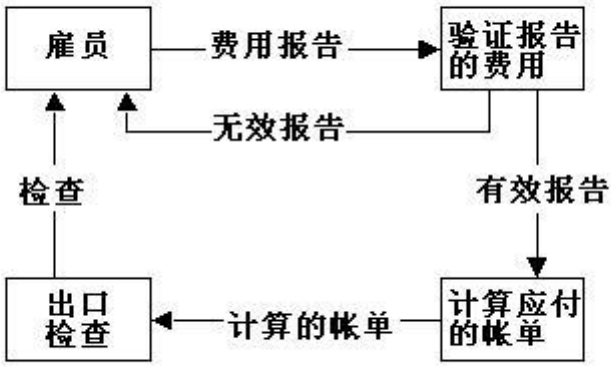

图6　数据流图

2.398

数据流图　data flow graph

见：　数据流图 data flow diagram（2.397）。

2.399

数据流图　data flow diagram(DFD)

数据流程图　data flowchart

数据流图　data flow graph

描述数据源、数据接收端、数据存储和在数据上如节点上执行的处理以及数据的逻辑流如节点之间的连接的一种图。

相对：　流程图 flow diagram（2.635）和数据结构图 data structure diagram（2.406）。

2.400

数据流程图　data flowchart(flow chart)

见： 数据流图 data flow diagram（2.397）。

2.401

数据输入表 data input sheet

用工作表形式描述的系统或部件要求的和任选的输入数据的用户文档。

2.402

数据库存 data inventory

某一在信息处理系统中的全部数据及其特性（包括相互依赖关系）。

2.403

数据敏感故障 data-sensitive fault

模式敏感故障 pattern-sensitive fault

引起在响应某些具体的数据模式时失效的故障。

相对： 程序敏感故障 program-sensitive fault（2.1242）。

2.404

数据结构 data structure

在设计以支持特定的数据操作函数的数据元素间的物理的或逻辑的关系。

2.405

以数据结构为中心的设计 data structure-centered design

一种软件设计技术，在此技术中，系统的结构是从系统必须处理的数据集的结构的分析中导出的。

参见： 输入-处理-输出 input-process-output（2.762）、模块分解 modular decomposition（2.973）、面向对象设计 object-oriented design（2.1037）、快速原型 rapid prototyping（2.1307）、逐步细化 stepwise refinement（2.1587）、结构冲突 structure clash（2.1606）、结构设计 structured design（2.1607）、事务分析 transaction analysis（2.1760）和转换分析 transform analysis（2.1764）。

2.406

数据结构图 data structure diagram

描述一组数据元素、它们的属性和它们之间逻辑关系的图（见图7）。

参见： 实体联系图 entity-relationship diagram（2.552）。

相对： 数据流图 data flow diagram（2.397）。

图7 数据结构图

2.407

数据踪迹 data trace

见： 变量踪迹 variable trace（2.1823）。

2.408

数据类型 data type

一类数据，它的特色取决于该类的成员和可对它们施行的操作，例如，字符型、枚举型、整型、实型、逻辑型。

参见： 强类型 strong typing（2.1601）。

2.409

数据 datum

data 的单数形式。

2.410

死锁 deadlock

由于两个或多个设备或进程，每一个等待赋给另一个的资源引起计算机处理停顿的情况。

参见： 锁定 lockout（2.872）。

2.411

反汇编程序 deassembler

见： 反汇编程序 disassembler（2.488）。

2.412

解块 deblock

分解块的部分。

相对： 块 block（2.143）。

2.413

调试，排错 debugging

检测、定位和改正计算机程序中的故障。技术包括使用断点（breakpoint）、桌面检查（desk checking）、转储（dump）、审查（inspection）、逆向执行（reversible execution）、单步操作（single-step operation）和追踪（trace）。

2.414

调试模型 debugging model

参见： 出错模型 error model（2.565）。

2.415

判定表 decision table

a) 在叙述一问题中要考虑的所有可能发生的情况及对每一组可能发生的情况将要采取的行动的一张表；

b) 对一组情况及其相应动作以矩阵形式或列表形式所做的表示。

2.416

声明 declaration

一种程序语句，它虽不执行但影响汇编程序或编译程序对程序中其他语句的解释。例如，标识一名字，规定此名字代表什么并可能赋它初始值的语句。

参见： 伪指令 pseudo instruction（2.1280）。

相对： 赋值语句 assignment statement（2.99）和控制语句 control statement（2.343）。

2.417

声明语言 declarative language

一非过程语言，它允许声明一组事实并表达使用这些事实的疑问或问题。

参见： 交互的语言 interactive language（2.793）和基于规则的语言 rule-based language

（2.1403）。

2.418

反编译 decompile

将已经编译的计算机程序从它的机器代码版本转换至重新装配但可能不同于原始的高级语言程序。

相对： 编译 compile（2.252）。

2.419

反编译程序 decompiler

反编译计算机程序的软件工具。

2.420

解耦 decoupling

使软件模块相互更加独立的过程，它用以减少变更的冲突和在独立模块中的错误。

参见： 耦合 coupling（358）。

2.421

缺陷 defect

见： 故障 fault（2.609）。

2.422

缺陷密度 defect density

产品中标识出的缺陷数目除以产品成分的规模（用该产品的标准度量术语表示）。

2.423

缺陷预防 defect prevention

与标识缺陷或潜在缺陷并预防将它们引入产品有关的活动。

2.424

缺陷根源 defect root cause

致使缺陷得以引入的根本原因（例如缺乏有效的过程）。

2.425

定义的过程 defined process

为达到某个特定目的，对一组活动的可操作性定义。

注：已定义的过程可以由一套标准、规程、培训、工具和方法体现其特性。

2.426

定义性模型 defining model(MOF)

某一存储库基于其中的模型。任何数目的存储库都能有同一定义性模型。

2.427

定义阶段 definition phase

参见： 需求阶段 requirement phase（2.1365）。

2.428

委派 delegation

为响应消息，某一对象向另一对象发布消息的能力。委派能用作对继承的一个替代。

相对： 继承 inheritance（2.752）。

2.429

定界符 delimiter

用于指示相关位组、字符、字或语句的开始或结束的一个字符或一组字符。

2.430

交付 delivery

a) 软件研制周期中的一个阶段，在此阶段将产品提交给它的客户或计划中的用户供其使用；

b) 软件研制周期中的一个阶段，在此阶段产品由其预定的用户接受。

2.431

按需调页 demand paging

一种存储分配技术，在此技术中，页只有当需要时，才从辅助存储器传送至主存储器。

相对：先期调页 anticipatory paging（2.66）。

2.432

简并 demodularization

在软件设计中，组合相关的软件模块的过程，通常为了优化系统性能。

参见： 向下压缩 downward compression（2.505）、侧向压缩 lateral compression（2.841）和向上压缩 upward compression（2.1800）。

2.433

演示 demonstration

一种动态分析技术，它依靠在执行期间对系统或部件的行为的观察，不需要执行后分析，以检测错误、对开发标准的违反和其他问题。

参见： 测试 testing（2.1726）。

2.434

依赖[关系] dependenty

两个建模元素之间的一种关系，对其中一个建模元素（独立元素）的更改，将影响另一建模元素（依赖元素）。

2.435

部署图 deployment diagram

一种图，用于显示运行的处理结点以及居于其上的构件、进程和对象的配置的图。构件表示代码单元在运行时的体现。

参见： 构件图 component diagram（2.262）。

2.436

导出元素 derived element

一种能从另一元素算出的如下模型元素：为明晰起见加以展现，或为设计目的予以纳入，即便它并未添加语义信息。

2.437

导出类型 derived type

一种数据类型，它的成员和操作来自其他数据类型，按照某种特定的规则得到的。

参见： 子类型 subtype（2.1623）。

2.438

描述标准 description standard

描述产品的特征或提供以帮助了解、测试、安装、操作或维护产品的过程的标准。

2.439

设计 design

a) 为使一软件系统满足规定的需求而定义系统或部件的体系结构、部件、接口和其他特征的过程。

b) 设计过程的结果。

2.440

设计分析 design analysis

a) 对一设计进行估计以确定其相对于预定需求的正确性、符合设计标准的程度、系统效率和是否符合其他一些准则；

b) 对其他替代性设计途径的估计。

2.441

设计分析器 design analyzer

一种自动设计工具。它接收有关程序的设计方面的信息，并产生以下方面的输出，如模块层次图、控制和数据结构的图形表示，以及被访问的数据块的一览表。

2.442

设计描述 design description

设计文档 design document

设计规格说明 design specification

一种文档，它描述系统或部件的设计、典型内容包括系统或部件体系结构、控制逻辑、数据结构、输入输出格式、接口描述和算法。

参见： 产品规格说明 product specification（2.1219）。

相对：需求规格说明 requirement specification（2.1367）。

2.443

设计文档 design document

见： 设计描述 design decription（2.442）。

2.444

设计元素 design element

设计中的基本部件或构件块。

2.445

设计实体 design entity

设计的元素（部件），它从结构上和功能上与其他元素相区别并分别命名和引用。

2.446

设计审查 design inspection

见： 审查 inspection（2.764）。

2.447

设计语言 design language

一种具有专门构造，有时还可验证的规格说明语言。用以开发、分析和说明硬件或软件设计。类型包括硬件设计语言、程序设计语言。

参见： 需求规格说明语言 requirement specification language（2.1368）。

2.448

设计级别 design level

软件项的设计分解（例如，系统、子系统、程序或模块）。

2.449

设计方法学 design methodology

进行设计的系统途径。由专门选择的工具、技术、准则的有序应用所构成。

2.450

设计阶段 design phase

软件生存周期中的一段时间。在这段时间内，进行体系结构、软件组成部分、接口和数据的设计，为设计编制文件，并对其进行验证，以满足预定需求。

参见： 详细设计 detailed design（2.462）和概要设计 preliminary design（2.1163）。

2.451

设计需求 design requirement

规定或强制系统或系统部件的设计的需求。

相对： 功能需求 functional requirement（2.667）、实现需求 implementation requirement（2.730）、接口需求 interface requirement（2.798）、性能需求 performance requirement（2.1133）

和物理需求 physical requirement（2.1140）。

2.452

设计评审 design review

a) 把系统、硬件或软件设计提交给项目人员、管理者、用户、客户或有关人士供其评审或批准的过程或会议；类型包括关键设计评审（critical design review）、概要设计评审（preliminary design review）和系统设计评审（system design review）。

相对： 代码评审 code review（2.224）、形式合格评审 formal qualification review（2.646）、需求评审 requirement review（2.1366）和测试就绪评审 test readinees review（2.1717）。

b) 对现有的或提出的设计所做的正式评估和审查，其目的是找出可能会影响产品，过程或服务工作的适用性和环境方面的设计缺陷并采取补救措施，以及（或者）找出在性能、安全性和经济方面的可能的改进。

2.453

设计规格说明 design specification

见： 设计描述 design description（2.442）。（一种描述设计要求的正式文档，按照这种文档对系统或系统组成部分〈如，软件配置项〉进行设计。典型内容包括系统或系统组成部分算法、控制逻辑、数据结构设定与使用（set-use）信息、输入输出格式和接口描述。）

2.454

设计标准 design standard

一种标准，它描述设计或数据或程序部件的设计描述的特性。

2.455

设计单元 design unit

设计元素的逻辑相关的集合。在 Ada PDL 中，设计单元由 Ada 编译单元表示。

2.456

设计视图 design view

特别适合于软件项目活动需要的设计实体属性信息的子集。

2.457

设计验证 design verification

见： 验证 verification（2.1826）。

2.458

设计走查 design walk-through

见： 走查 walk-through（2.1843）。

2.459

桌面检查 desk checking

一种静态分析技术，在此技术中，可视地检查代码清单、测试结果或其他文档，通常，这样的核对由产生它们的人员进行，以标识错误、对标准的违反或其他问题。

参见： 审查 inspection（2.764）和走查 walk-through（2.1843）。

2.460

目的地址 destination address

数据被传送至的设备或存储单元的地址。

相对： 源地址 source address（2.1540）。

2.461

破坏性读 destructive read

一种读操作，它变更被访问的单元中的数据。

相对： 非破坏性读 nondestructive read（2.1024）。

2.462

详细设计 detailed design

a) 推敲并扩充初步设计，以获得关于处理逻辑、数据结构和数据定义的更加详尽的描述，直到设计完善到足以能实现的地步；

参见： 软件开发过程 software development process（2.1491）。

b) 详细设计过程的结果。

2.463

开发方 developer

在软件生存周期中执行开发活动（包括需求分析、设计直至验收）的一个机构。

2.464

开发周期 development cycle

见： 软件开发周期 software development cycle（2.1483）。

2.465

开发生存周期 development life cycle

见： 软件开发周期 software development cycle（2.1483）。

2.466

开发方法学 development methodology

编制软件的系统方法。它确定开发的各个阶段，规定每一阶段的活动、产品、验证步骤和完成准则。

2.467

开发规格说明 development specification

见： 需求规格说明 requirement specification（2.1367）。

2.468

开发测试 development testing

在系统或部件的开发期间的正式的或非正式的测试活动，通常由开发者在开发环境中进行。

参见： 合格性测试 qualification testing（2.1297）。

相对： 验收测试 acceptance testing（2.19）和运行测试 operational testing（2.1065）。

2.469

开发基线 developmental baseline

见： 开发配置 developmental configuration（2.470）。

2.470

开发配置 developmental configuration

在配置管理中，软件和相关的技术文档，此文档定义在开发期间计算机软件配置项的展开的配置。

注：开发配置是在开发者的控制之下，因此，不能称为基线。

相对： 分配基线 allocated baseline（2.57）、功能基线 functional baseline（2.659）和产品基线 product baseline（2.1213）。

2.471

开发配置管理 developmental configuration management

为指定和控制软件和相关的技术文档，来定义开发期间处在不断进化中的软件工作产品的配置，而运用技术手段和行政的管理手段。开发配置管理处在开发者的直接控制之下。置于开发配置管理之下的配置项不是基线，虽然在开发过程的某些点上，它们可能被基线化并置于基线配置管理之下。

2.472

开发过程 development process

在软件开发期间，为给定目的（例如构造模型或实现模型）而进行的一组部分有序的步骤。

2.473

偏离 deviation

a) 从规定需求的脱离；

b) 所写的授权，在项的制造之前，允许规定的单元数或规定的周期时间偏离具体的性能或设计要求。

注： 与工程变更不同，偏离并不要求定义影响项的文档的修订。

参见： 配置控制 configuration control（2.307）。

相对： 工程变更 engineering change（2.546）和放弃 waiver（2.1842）。

2.474

设备（器件） device

设计用于服务或执行功能的一种机制或装置的一部分。

2.475

诊断的 diagnostic

用于说明故障或失效的检测和隔离。例如，诊断消息、诊断手册。

2.476

诊断手册 diagnostic manual

一种文档，它表示对于系统或部件执行诊断过程必需的信息，标识故障和对故障的补救措施。典型地，描述的是系统或部件的诊断特性和支持的可用的诊断工具。

参见： 安装手册 installation manual（2.766）、操作员手册 operator manual（2.1068）、程序员手册 programmer manual（2.1250）、支持手册 support manual（2.1631）和用户手册 user manual（2.1813）。

2.477

对角微指令 diagonal microinstruction

一种微指令，能规定为执行机器语言指令所需的有限数量的同时操作。

注： 对角微指令在尺寸和功能上都处在水平微指令和垂直微指令之间。“斜的”的含义仅涉及到一种折衷，而不是涉及微指令的任何物理特征。

相对： 横向微指令 horizontal microinstruction（2.708）和纵向微指令 vertical microinstruction（2.1832）。

2.478

差异转储 differential dump

见： 变更转储 change dump（2.196）。

2.479

图 diagram

对模型元素的汇集，在大多数情况下绘制成以弧（联系）与顶点（其他模型元素）相连的图解的一种图形表示。UML 支持的图有：类图（class diagram）、对象图（object diagram）、用况图（use case diagram）、时序图（sequence diagram）、协作图（collaboration diagram）、状态图（statechart diagram）、活动图（activity graph）、构件图（component diagram）和部署图（deployment diagram）。

2.480

有向图 digraph

见： 有向图 directed graph（2.485）。

2.481

直接地址 direct address

一级地址 one-level address

标识一操作数的存储单元的地址。

参见： 直接指令 direct instruction（2.483）。

相对： 立即数据 （2.721 条 immediate data）、间接地址 indirect address（2.744）和 n 级地址 n-level address（2.1007）。

2.482

直接插入子例程 direct insert subroutine

见： 开式子例程 open subroutine（2.1052）。

2.483

直接指令 direct instruction

包含操作数直接地址的计算机指令。

参见： 绝对指令 absolute instruction（2.12）和有效指令 effective instruction（2.532）。

相对： 立即指令 immediate instruction（2.722）和间接指令 indirect instruction（2.745）。

2.484

直接测度 direct measure

不依赖于任何其他属性度量的一种对属性的度量。

2.485

有向图 directed graph

有向图 digraph

一种图，其中方向隐含在节点间的连接中（见图 8）。

相对： 无向图 undirected graph（2.1787）。

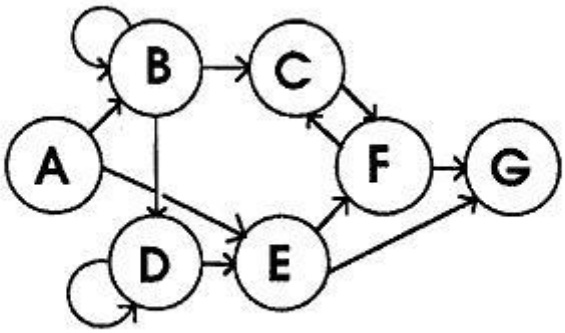

图8 有向图

2.486

目录 directory

数据项和关于这些数据项的信息构成的清单。

2.487

反汇编 disassemble

将已汇编的计算机程序的机器语言版本转换为类同于但可能并不等同于原始的汇编语言程序。

相对： 汇编 assemble（2.79）。

2.488

反汇编程序 disassembler

反汇编程序 deassembler

对计算机程序进行反汇编的软件工具。

2.489

离散型 discrete type

一种数据类型，它的成员能假定是任何不同值的集合。离散型可以是枚举型或整数型。

2.490

互斥子状态 disjoint substate

一种不能与包含在同一组合状态中的另一子状态同时保有的子状态。

相对： 并发子状态 concurrent substate（2.300）。

2.491

分发单元 distribution unit

一种作为一组分配到某一过程或处理器的对象或构件的集合。分发单元能用运行时间的组合或聚集加以表示。

2.492

多样冗余 diverse redundancy

见： 多样性 diversity（2.493）。

2.493

多样性 diversity

在容错系统中，同样的功能可以用不同的方法实现。例如，用不同的处理器、存储媒体、编程语言、算法或开发组。

参见： 软件多样性 software diversity（2.1492）。

2.494

空操作 do-nothing operation

见： 空操作 no-operation（2.1019）。

2.495

文档，文件 document

a) 一种数据媒体和其上所记录的数据。它具有永久性并可以由人或机器阅读。在软件工程中的例子，包括：项目计划、规格说明书、测试计划、用户手册；

b) 编制文件；

c) 在计算机程序中增加注释。

2.496

文档汇集、文档编制，文档管理，文档 documentation

a) 关于一给定主题的文件集合；

参见用户文档 user documentation（2.1809）、软件文档 software documentation（2.1493）和系统文档 system documentation（2.1653）。

b) 文档管理可能包括下述活动：对文档的标识、获取、处理、存储和发放；

c) 产生或修订一个文档的过程；

d) 为了对活动、需求、过程或结果进行描述、定义、规定、报告或认证的任何书面或图示的信息。

2.497

文档级 documentation level

见： 文档等级 level of documentation（2.846）。

2.498

文档树 documentation tree

一种图形，它规定给定系统的全部文档并表示它们之间的相互关系（见图 9）。

参见： 规格说明书树 spicification tree（2.1551）。

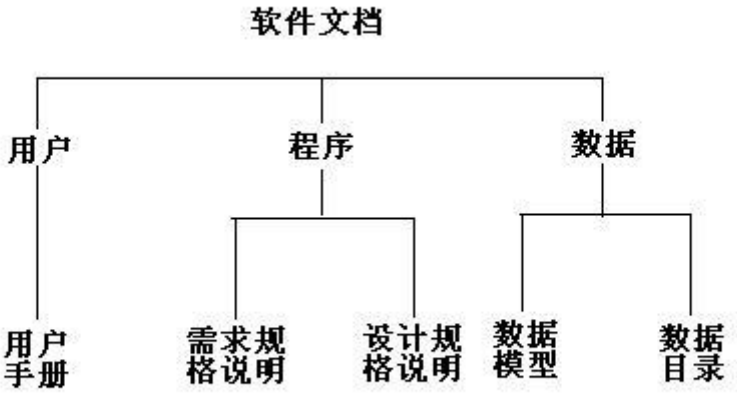

图9　文档树

2.499

文档化的规程　documented procedure

对完成给定任务将要采取措施步骤的书面描述。这种描述是以文档形式存在的。

参见：　规程 procedure（2.1181）。

2.500

领域　domain

由为业内人士所理解的一套概念和术语所表征的一种知识或活动范围。

2.501

双操作数指令　double-operand instruction

见：　二地址指令 two-address instruction（2.1777）。

2.502

停机　down

用于说明不操作或已经退出服务的系统或部件所处的状态。

参见：　忙 busy（2.168）、崩溃 crash（2.362）和空闲 idle（2.716）。

相对：　开机 up（2.1797）。

2.503

停机时间　down time

系统或部件不操作或退出服务的时间周期。

参见：　忙时 busy time（2.169）、闲时 idle time（2.717）、平均修复时间 mean time to repair（2.921）和设置时间 set-up time（2.1444）。

相对：　开机时间 up time（2.1798）。

2.504

向下兼容　downward compatible

关于硬件或软件与它自己的以前版本兼容。例如，程序处理由它自己的以前的版本创建的文件。

相对：　向上兼容 upward compatible（2. 1799）。

2.505

向下压缩　downward compression

在软件设计中，去模块化的一种方式，在此方式中，上级模块复制至子模块体中。

相对：侧向压缩 lateral compression（2.841）和向上压缩 upward compression（2.1800）。

2.506

驱动程序 driver

a) 一软件模块，它引用而且或许控制和监控一个或多个软件模块的执行；

参见： 测试驱动程序 test driver（2.1705）。

b) 一种计算机程序，它控制外部设备，有时，重新格式化与设备之间传送的数据。

2.507

[程序]对编 dual coding

一种开发技术。由不同的程序员或不同的程序设计小组，根据同一份规格说明书开发出功能上完全相同的程序的两个版本。所获得的源代码可以采用同一种语言，也可以采用不同的语言。双份编码的目的在于提供错误检测，提高可靠性，提供附加的文件说明，或使系统的程序设计错误或编译程序错误影响最终结果的概率降低。

参见： 软件多样性 software diversity（2.1492）。

2.508

虚参数 dummy parameter

见： 形参 formal parameter（2.645）。

2.509

转储 dump

a) 计算机程序的执行状态的某些方面的显示，通常是内部的存储器或寄存器的内容；类型包括：变更转储（change dump）、动态转储（dynamic dump）、内存转储（memory dump）、事后转储（postmortem dump）、选择性转储（selective dump）、快照转储（snapshot dump）和静态转储（static dump）。

b) 文件或设备的内容的显示；（为了某一专门目的。如允许存储器另作它用，或作为预防故障和错误的措施、或为了进行与排除错误有关的工作，将一存储器〈通常是内部存储器〉的全部或部分内容写到外部媒体上。）

c) 复制内部存储器的内容至外部媒体；

d) 产生如在 a)、b)、c)中的显示或副本。

2.510

双元选择构造 dyadic selective construct

if-then-else 构造，在其中，对分支的两个出口都规定处理。

相对： 一元选择构造 monadic selective construct（2.980）。

2.511

动态的 dynamic

用于说明在计算机程序执行期间发生的事件或过程、例如，动态分析（dynamic analysis）和动态绑定（dynamic binding）。

相对： 静态的 static（2.1576）。

2.512

动态分配 dynamic allocation

把可编址的存储器和其他资源分配给正在执行的程序。

见： 动态资源分配 dynamic resource allocation（2.522）。

2.513

动态分析 dynamic analysis

在执行期间，基于它的行为评价系统或部件的过程。

参见： 演示 demonstration（2.433）和测试 testing（2.1726）。

相对： 静态分析 static analysis（2.1577）。

2.514

动态分析器 dynamic analyzer

借助对程序执行情况的监控，帮助对计算机程序进行估计的软件工具。例如探测工具、软件监控器和跟踪器。

相对： 静态分析程序 static analyzer（2.1578）。

2.515

动态绑定 dynamic binding

在计算机程序执行期间进行的绑定。

相对： 静态绑定 static binding（2.1579）。

2.516

动态断点 dynamic breakpoint

预定义的启动事件是程序的运动时特征的断点，例如，任何 20 条源语句的执行。

参见： 代码断点 code breakpoint（2.220）、数据断点 data breakpoint（2.391）、跋断点 epilog breakpoint（2.559）、可编程的断点 programmable breakpoint（2.1249）和序断点 prolog breakpoint（2.1268）。

相对： 静态断点 static breakpoint（2.1580）。

2.517

动态缓冲 dynamic buffering

已分配给计算机程序的缓冲器，在程序执行期间，基于当前的需要而变更的缓冲技术。

相对： 简单缓冲 simple buffering（2.1451）。

2.518

动态分类 dynamic classification

其中的某一对象可更改其类的泛化的一种语义类。

相对： 静态分类 static classification（2.1581）。

2.519

动态转储 dynamic dump

在计算机程序执行期间发生的转储。

参见： 变更转储 change dump（2.196）、内存转储 memory dump（2.928）、事后转储 postmortem dump（2.1153）、选择性转储 selective dump（2.1425）和快照转储 snapshot dump（2.1466）。

相对： 静态转储 static dump（2.1582）。

2.520

动态差错 dynamic error

随输入的时间而变化的差错。

相对： 静态差错 static error（2.1583）。

2.521

动态重定位 dynamic relocation

计算机程序在它执行期间的重定位。

2.522

动态资源分配 dynamic resource allocation

赋给程序的资源在程序执行期间，基于当前的需要变更的计算机资源分配技术。

2.523

动态重构 dynamic restructuring

在程序执行期间，重构数据库、数据结构、计算机程序或系统部件集的过程。

2.524

动态存储分配 dynamic storage allocation

赋给计算机程序的存储器在程序执行期间，基于程序和其他执行的程序的当前需要变更的存储分配技术。

2.525

实体联系图 E-R diagram

entity-relationship diagram 的缩略语。

2.526

早期失效周期 early-failure period

老化周期 burn-in period

系统或部件生存期中的时间周期，在此周期中，硬件失效按递减的比率发生，如问题的检测和修复。

参见： 浴盆曲线 bathtub curve（2.129）。

相对：定失效率周期 constant-failure period(2.322)和磨损失效周期 wearout-failure period（2.1845）。

2.527

回送 echo

a) 返回一发送的信号至它的源，它常常具有延迟，指示这是反射的而不是原始的；

b) 如在 a 中的返回的信号。

2.528

工程变更建议 ECP

engineering change proposal 的缩略语。

2.529

编辑 edit

修改计算机代码、数据或文档的形式或格式。例如，插入、重安排或删除字符。

2.530

编辑程序 editor

a) 见： 文本编辑程序 text editor（2.1728）；

b) 见： 连接编辑程序 linkage editor（2.854）。

2.531

有效地址 effective address

从规定的地址上执行任何所要求的变址、间接寻址或其他修改而得到的地址。注：若规定的地址不要求修改，这也是有效地址。

参见： 生成的地址 generated address（2.675）、间接地址 indirect address（2.744）和相对地址 relative address（2.1331）。

2.532

有效指令 effective instruction

在规定的计算机指令中，在地址上执行任何的变址、间接寻址或其他修改得出的计算机指令。

注： 若规定的指令不要求修改，这也是有效指令。

参见： 绝对指令 absolute instruction（2.12）、直接指令 direct instruction（2.483）、立即指令 immediate instruction（2.722）和间接指令 indirect instruction（2.745）。

2.533

有效过程 effective process

实践过的、已文档化的、强制的、经培训的、已测量的和可改进的过程。

参见： 妥善定义的过程 well-defined process（2.1846）。

2.534

传出的 efferent

在软件系统中，从高级模块至子模块的数据或控制流。

相对： 传入的 afferent（2.51）。

2.535

效率 efficiency

系统或部件以最小的计算资源消耗实现其预定功能的程度。

参见： 执行效率 execution efficiency（2.578）和存储效率 storage efficiency（2.1594）。

2.536

无我程序设计 egoless programming

在对程序开发采用小组负责制而不是独立的对程序开发负责的概念的基础上进行软件开发的一种方式。其目的是防止个别的程序员与其产生的输出的关系过于密切，以免使客观的评价受到损害。

2.537

元素 element

某一模型的一种原子成分。

2.538

嵌入式计算机系统 embedded computer system

大的计算机系统的一部分并能执行大的系统的某些需求的计算机系统。例如，用于航天器或快速传输线系统中的计算机系统。

2.539

嵌入式软件 embedded software

a) 大的系统的一部分并能执行其某些需求的软件。例如，用于航天器或快速传输线系统中的软件。

b) 嵌入式计算机系统所用的软件。

2.540

模拟 emulation

a) 一个模型，它接受给定的系统相同的输入并产生相同的输出；

参见： 仿真 simulation（2.1453）。（用一个计算机系统，主要是通过硬件，模仿另一个计算机系统的全部或部分功能，使进行模仿的系统接受的数据、执行的程序和实现的结果均与被模仿的系统所接受的数据，执行的程序和实现的结果相同。）

b) 像在 a)中的开发过程或使用模型。

2.541

模拟器 emulator

一种设备、计算机程序或系统，它接受给定系统同样的输入并产生同样的输出。

参见：仿真器 simulator（2.1454）。

2.542

封装 encapsulation

将系统功能、一组数据和在这些数据上的操作隔离在一个模块中，并为该模块提供精确的规格说明的软件开发技术。

参见： 数据抽象 data abstraction（2.388）和信息隐蔽 information hiding（2.751）。

2.543

最终用户 end user

系统部署在其运行环境下，使用系统以实现其预定用途的个人或组。

2.544

最终用户代表 end user representatives

挑选出的、代表最终用户的个人或组。

2.545

工程 engineering

系统的、训练有素的、可计量的、处理的应用，以构造机器、产品、系统或过程。

2.546

工程变更 engineering change

在配置管理中，配置项或其他指定的项在正式建立它的配置标识后在配置中变更。

参见：配置控制 configuration control（2.307）和工程变更建议 engineering change proposal（2.546）。

相对： 偏离 deviation（2.473）和放弃 waiver（2.1842）。

2.547

工程变更建议 engineering change proposal(ECP)

在配置管理中，工程变更的建议并用文档描述变更建议。

参见： 配置控制 configuration control（2.307）。

2.548

工程组 engineering group

由从事工程学科的人员（包括负责人、管理人员和技术人员）组成的集合。工程学科的例子有系统工程、硬件工程、系统测试、软件工程、软件配置管理和软件质量保证。

2.549

增强的能力 enhanced capability

可由一个可信赖的过程改进方案证明的，比当前评估的能力更强的能力。

2.550

实体 entity

在计算机程序设计中，程序中能命名或指定的项。例如，数据项、程序语句或子程序。

2.551

实体属性 entity attribute

一种命名的特征或设计实体的特性。它提供关于实体行为的语句。

2.552

实体联系图 entity-relationship (E-R)diagram

实体联系图 entity-relationship map

规定实际的实体集和它们之间的逻辑联系的图。

参见： 数据结构图 data structure diagram（2.406）。

2.553

实体联系图 entity-relationship (E-R)map

见： 实体联系图 entity-relationship diagram（2.552）。

2.554

入口 entrance

见： 入口 entry point（2.557）。

2.555

入口 entry

见： 入口 entry point（2.557）。

2.556

进入动作 entry action

进入状态机的某一状态（而不管达到该状态采取何种转移）所执行的一种动作。

2.557

入口[点] entry point

入口 entrance

入口 entry

软件模块的一个点，在此点模块能开始执行。

参见： 重入点 reentry point（2.1326）。

相对： 出口 exit（2.586`）。

2.558

枚举型 enumeration type

一种离散的数据类型，它的成员的值能由程序员显式定义。例如，一种称为颜色（COLORS）的数据类型，具有可能的值：红（RED）、蓝（BLUE）和黄（YELLOW）。

相对： 字符型 character type（2.201）、整型 integer type（2.784）、逻辑型 logical type（2.878）和实型 real type（2.1314）。

2.559

跋断点 epilog breakpoint

后同步断点 postamble breakpoint

直至从给定的程序或例程退出时才启动的断点。

参见：代码断点 code breakpoint（2.220）、数据断点 data breakpoint（2.391）、动态断点 dynamic breakpoint（2.516）、可编程断点 programmable breakpoint（2.1249）和静态断点 static breakpoint（2.1580）。

与 2.1268 条 序断点（prolog breakpoint）相对。

2.560

等效故障 equivalent faults

导致相同失效方式的两个或多个故障。

2.561

出错，误差，差错 error

a) 计算的、观察的或测量的值或条件与实际的、规定的或理论上正确的值或条件的差别。例如，在计算的结果和正确的结果之间差 30m；

b) 不正确的步骤、过程或数据定义。例如，在计算机程序中的不正确的指定；

c) 不正确的结果。例如，当正确的结果是 10，而计算的结果是 12；

d) 产生不正确结果的人为动作。例如，在编程或操作的一部分上的不正确动作。

注： 当上述所有四种定义是公共使用时，一种区分赋给定义a为字差错（error），定义b为字过错（fault），定义c为字失效（failure）和定义d为字错误（mistake）。

参见： 动态差错 dynamic error（2.520）、致命差错 fatal error（2.608）、内在差错 indigenous error（2.742）、语义差错 semantic error（2.1430）、静态差错 static error（2.1583）、句法差错 syntactic error（2.1641）和瞬时差错 transient error（2.1767）。

2.562

出错分析 error analysis

a) 对观察到的软件故障进行调查的过程，调查的目的是跟踪那个故障以找出故障源；

b) 对观察到的软件故障进行调查以找出以下一些信息，例如故障原因。该故障是在开发过程中哪一个阶段发生的，预防或较早地探测出软件故障的方法；

c) 调查软件错误、失效和故障以确定定量速率和趋势的过程。

2.563

出错类别 error category

错误、故障或失效可能归并到其中的一组类别之一，当错误、故障或失效发生或发现后，可根据其原因、危急程度、效果、故障所属的生存周期阶段或其他特性而确定其类别。

2.564

出错数据 error data

出错数据通常（但不是精确地）用于：描述软件的问题、故障、失效及其变更，它们的特性，以及遇到或改正这些问题的条件。

2.565

出错模型 error model

出错预测模型 error prediction model

在软件评价中，用于估计或预报系统余下故障数、需要的测试时间和类似特性的模型。

参见： 出错预测 error prediction（2.566）。

2.566

出错预测 error prediction

关于系统或部件中预计的故障数或性质所作的定量陈述。

参见： 出错模型 error model（2.565）和差错播种 error seeding（2.569）。

2.567

出错预测模型 error prediction model

见： 出错模型 error model（2.565）。

2.568

出错恢复 error recovery

参见： 失效恢复 failure recovery（2.607）。

2.569

差错播种 error seeding

隐错播种 bug seeding

故障播种 fault seeding

故意地把已知故障加至已经有故障的计算机程序中，为了监控检测和删除的速度并估计程序中余下的故障数的过程。

参见： 内在错误 indigenous error（2.742）。

2.570

容错 error tolerance

尽管存在着错误的输入，系统或部件仍能正常操作的能力。

参见： 故障容忍 fault tolerance（2.616）和健壮性 robustness（2.1397）。

2.571

评价 evaluation

决定某产品、项目、活动或服务是否符合它的规定的准则的过程。

2.572

评价模块 evaluation module

针对特定软件质量特性或子特性的评价技术包。

注：该评价技术包概括了评价方法和技术、要评价的输入、要测量和收集的数据，以及支持规程和工具。

2.573

事件 event

对在时空上占有某一位置的重要现象的规格说明。在状态图语境中，事件是能触发某一转移的一种现象。

2.574

事件驱动的评审或活动 event-driven review/activity

根据项目中某事件的发生而进行的评审或活动。

2.575

异常 exception

引起正常程序执行挂起的事件。类型包括：寻址异常（addressing exception）、数据异常（data

exception)、操作异常(operation exception)、溢出异常(overflow exception)、保护异常(protection exception) 和下溢异常 (underflow exception)。

2.576

执行 execute

实施指令、过程或计算机程序。

2.577

执行 execution

由计算机运行计算机程序中一条或多条指令的过程。

2.578

执行效率 execution efficiency

系统或部件用最小的时间耗费执行它的指定的功能的程度。

参见： 执行时间 execution time (2.580)、存储效率 storage efficiency (2.1594)。

2.579

执行监控程序 execution monitor

见： 监控程序 monitor (2.981)。

2.580

执行时间 execution time

运行时间 run time

运行的时间 running time

执行一个计算机程序所用的消逝时间或中央处理器所用时间。

注： 处理器时间通常小于消逝时间，因为处理器可能空闲（例如，等待所需要的资源）或在程序执行期间由别的任务使用。

参见： 开销时间 overhead time (2.1084)。

2.581

执行时间理论 execution time theory

采用累计执行时间作为估计软件可靠性基础的一种理论。

2.582

执行踪迹 execution trace

代码踪迹 code trace

控制流踪迹 control-flow trace

在计算机程序执行期间执行的指令序列的记录。常取当程序执行时遇到的代码标记的清单的形式。

参见： 回顾的踪迹 retrospective trace (2.1383)、子例程踪迹 subroutine trace (2.1620)、符号踪迹 symbolic trace (2.1639) 和变量踪迹 variable trace (2.1823)。

2.583

执行程序 executive

见： 监督程序 supervisory program (2.1628)。

2.584

执行程序 executive program

见： 监督程序 supervisory program (2.1628)。

2.585

执行状态 executive state

见： 监督状态 supervisory state (2.1627)。

2.586

出口 exit

软件模块中，模块的执行能终止的点。

参见： 返回 return（2.1384）。

相对： 入口点 entry point（2.557）。

2.587

退出动作 exit action

退出状态机的某一状态（而不管退出该状态采取何种转移）所执行的一种动作。

2.588

出口例程 exit routine

当规定的事件（例如错误）发生时接收控制的例程。

2.589

可扩充性 expandability

见： 可扩展性 extendability（2.594）。

2.590

显式地址 explicit address

见： 绝对地址 absolute address（2.9）。

2.591

引出 export

在包的语境中，使某一元素在其环绕的命名空间之外可见。

参见： 可见性 visibility（2.1841）。

相对： 引入 import（2.733）。

2.592

表达式 expression

一种对某一特别类型求值的串。例如，表达式“7+5*3”求“数”类型的值。

2.593

扩展 extend

用况之间的一种关系，它表明为扩展用况所定义行为如何增广（依扩展中规定的条件而定）为基用况所定义的行为。该行为在由基用况中的扩展点所定义的部位被插入。基用况不依赖于扩展用况行为的完成。

2.594

可扩展性 extendability

可扩充性 expandability

可扩展性 extensibility

系统或部件能修订以增加它的存储或功能的能力的情况。

参见： 灵活性 flexibility（2.634）和可维护性 maintainability（2.903）。

2.595

可扩展性 extensibility

见： 可扩展性 extendability（2.594）。

2.596

外部测度 external measure

通过对系统行为的测量得出的对产品的一种间接量测，其中产品是系统的一部分。

注 1：系统包括任何相关的硬件、软件（定制的软件或现货软件）和用户。

注 2：在测试中发现的失效数量是对程序中的故障数量的外部度量，因为失效的数量是在计算机系统运行程序的过程中计算的。

注 3：外部测度可以用来评价更接近于最终设计目标的质量属性。

2.597

外部质量 external quality

产品在特定条件下使用时，满足显式或隐式要求的程度。

2.598

因素分解 factoring

a) 分解一个系统为层次模块的过程。

参见： 模块分解 modular decomposition（2.973）。

b) 从一模块中移动一函数并把它放入自己的模块的过程。

2.599

失效保险 fail safe

系统或部件在失效的事件中，自动把自己放入一种安全的操作方式。例如，当正常的操作失败时，交通灯在所有方向都转换为闪光的红灯。

参见：故障安全 fault secure（2.614）和故障容忍 fault tolerance（2.616）。

相对： 故障弱化 fail soft（2.600）。

2.600

故障弱化 fail soft

在一定的失效事件中，系统或部件继续提供部分操作能力的情况。例如，若黄灯失效，交通灯继续在红灯和绿灯之间交替。

参见：故障安全 fault secure（2.614）和故障容忍 fault tolerance（2.616）。

相对：失效保险 fail safe（2.599）。

2.601

失效 failure

系统或部件不能按规定的性能要求执行它所要求的功能。注：故障容忍在人们的动作（弄错－mistake）、它的显示（硬件或软件故障）、故障的结果（失效）和不正确（差错－error）结果的总数之间进行区分。

参见： 崩溃 crash（2.362）、异常 exception（2.575）、失效方式 failure mode（2.604）、失效率 failure rate（2.605）、硬失效 hard failure（2.689）、初期失效 incipient failure（2.735）、随机失效 random failure（2.1306）和软失效 soft failure（2.1468）。

2.602

失效类别 failure category

见： 出错类别 error category（2.563）。

2.603

失效数据 failure data

见： 出错数据 error data（2.564）。

2.604

失效方式 failure mode

失效的物理的或功能的一种表示。例如，系统的失效方式可以由慢操作、不正确的输出或执行的完全终止作为特征。

2.605

失效率 failure rate

失效数与给定测量单位的比率。例如，每单位时间的失效次数、若干次事务处理中的失效次数，若干次计算机运行中的失效次数。

2.606

失效比 failure ratio

见： 失效率 failure rate（2.605）。

2.607

失效恢复 failure recovery

系统失效后又回到可靠的运行状态。

2.608

致命差错 fatal error

导致系统或部件完全丧失功能的差错。

2.609

故障，缺陷 fault

a) 硬件设备或部件中的缺陷。例如，短路或断线。

b) 在计算机程序中不正确的步骤、过程或数据定义。注：此定义最初由容错系统使用。在通常用法中，术语“差错（error）”和“隐错（bug）”表示同样含义。

参见： 数据敏感故障 data - sensitive fault（2.403）、程序敏感故障 program-sensitive fault（2.1242）、等效故障 equivalent fault（2.560）、故障屏蔽 fault masking（2.613）和间歇故障 intermittent fault（2.803）。

2.610

故障类别 fault category

见： 出错类别 error category（2.563）。

2.611

故障字典 fault dictionary

系统或部件中的故障清单和已经指定进行检测的测试清单。

2.612

故障插入 fault insertion

见： 故障播种 fault seeding（2.615）。

2.613

故障屏蔽 fault masking

一种故障防止检测另一种故障的条件。

2.614

故障安全 fault secure

用于说明系统或部件，在其中失效不会从预描述的故障集中产生。

参见： 故障容忍 fault tolerance（2.616）、失效保险 fail safe（2.599）和故障弱化 fail soft（2.600）。

2.615

故障播种 fault seeding

隐错播种 bug seeding

为了估计程序中的固有故障数，有意地在计算机程序已有的故障上添加已知数目的故障的过程。

2.616

故障容忍 fault tolerance

a) 尽管存在硬件或软件故障，系统或部件仍可继续正常运行的内在能力；

参见：容错 error tolerance(2.570)、失效保险 fail safe(2.599)、故障弱化 fail soft(2.600)、故障安全 fault secure（2.614）和健壮性 robustness（2.1397）。

b) 在正常操作被损害之前，系统或部件能经得起的故障数；

c) 关于错误、故障和失效以及在故障存在的情况下允许系统继续正常操作的方法的研究。

参见： 恢复 recovery（2.1319）、冗余 redundancy（2.1323）和重启动 restart（2.1379）。

2.617

故障容忍 fault tolerant

系统或部件，尽管存在故障仍能继续正常操作。

2.618

功能性配置审核 FCA

functional configuration audit 的缩略语。

2.619

可行性 feasibility

对于系统或部件的需求、设计或计划在存在的约束下仍能实现的程度。

2.620

可行性研究 feasibility study

辨别和分析某一问题及其可能解决方法，以确定其生命力、成本和效益的一种研究。

2.621

特征 feature

封装在某一类目（例如接口、类或数据类型）之内的一种性质，如操作或属性。

2.622

读取 fetch

从存储器中定位和装入计算机指令或数据。

参见： 移动 move（2.983）和存储 store（2.1595）。

2.623

第五代语言 fifth generation language(5GL)

一种计算机语言，它合并基于知识的系统、专家系统、推论引擎和自然语言处理的概念。

相对： 汇编语言 assembly language（2.86）、第四代语言 fourth generation language（2.654）、高级语言 high order language（2.702）和机器语言 machine language（2.890）。

2.624

文字式常数 figurative constant

为了在编程语言中保留的特殊的常数的数据名。例如，数据名 THREE，可以为表示值 3 保留。

参见： 字面值 literal（2.862）。

2.625

文件，文卷 file

作为一个单位来看待的一组相关的记录。例如，在采购控制中，文件能是由一组发货单记录组成。

参见： 逻辑文件 logical file（2.875）。

2.626

终[结]状态 final state

一种表明环绕的组合状态或整个状态已经完成的特种状态。

2.627

发现 findings

对在调查范围内标识出最重要的事情、问题或时机而进行的评估、评价、审核或评审的结论。

2.628

有限状态机 finite state machine

由有限个状态及这些状态之间转换（可能伴随有动作）构成的计算模型。

2.629

激发 fire

执行某一状态转移。

参见： 转移 transition（2.1769）。

2.630

固件 firmware

硬件设备和驻留在此设备上的作为只读软件的计算机指令和数据的组合。

2.631

第一代语言 first generation language(1GL)

见： 机器语言 machine language（2.890）。

2.632

一线软件经理 first-line software manager

对由软件工程师和其他有关人员组成的单位（例如一个部门或项目组）的人员配备和活动负有直接管理责任（包括提供技术指导及人事和工资管理等职能）的负责人。

2.633

标志 flag

设置为预描述的状态的变量，它常常基于处理的结果或规定的条件的发生置为“真（true）”或“假（false）”。

参见： 指示器 indicator（2.741）和信号量 semaphore（2.1433）。

2.634

灵活性 flexibility

适应性 adaptability

系统或部件能轻而易举地加以修改以便能在不是设计规定的应用程序或环境中使用的特性。

参见： 可扩展性 extendability（2.594）和可维护性 maintainability（2.903）。

2.635

流图 flow diagram

见： 流程图 flowchart（2.637）。

2.636

控制流 flow of control

执行某一算法时所完成的操作序列。

见： 控制流 control flow（2.338）。

2.637

流程图 flowchart

流图 flow diagram

控制流程图，在其中，适当注释的几何图用于表示操作、数据或设备，箭头用于指示从一个到另一个的顺序流（见图 10）。

参见： 框图 block diagram（2.145）、盒图 box diagram（2.156）、泡图 bubble chart（2.161）、图 graph（2.684）、输入处理输出图 input-process-output chart（2.763）和结构图 structure chart（2.1605）。

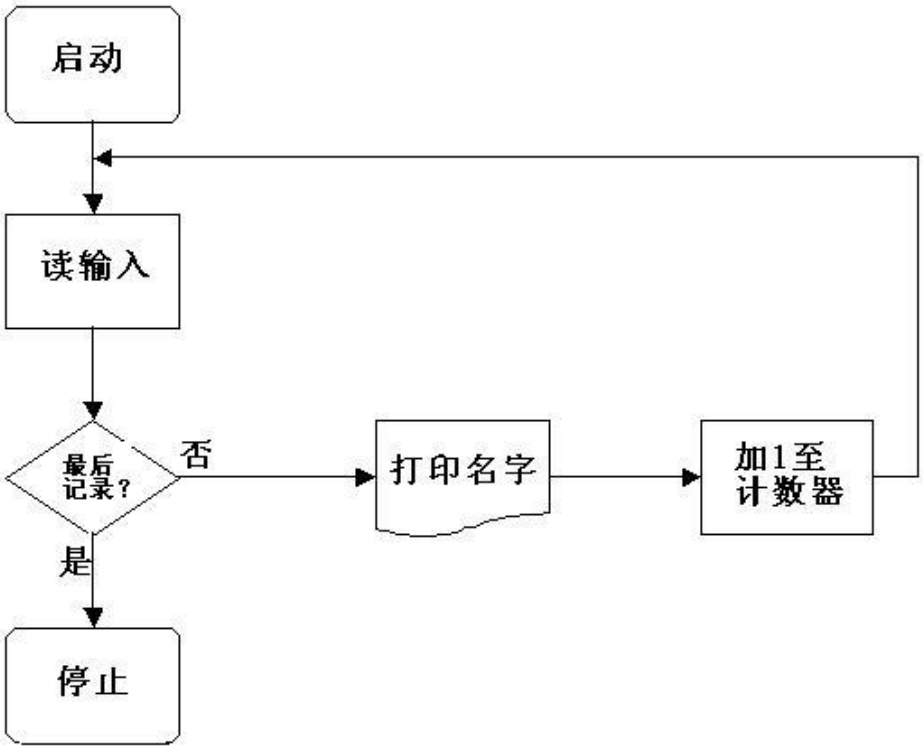

图10　流程图

2.638

流程图生成程序　flowcharter

一种软件工具，它接受程序的设计或代码表示作为输入并产生程序的流程图作为输出。

2.639

焦点类　focus class

对进行支持的一个或多个辅助类定义核心逻辑或控制流的一种衍型类。在典型情况下，焦点类与一个或多个辅助类一起使用，在设计阶段，对规定构件的核心业务逻辑或控制流特别有用。

参见： 辅助类 auxiliary class（2.114）。

2.640

控制聚焦　focus of control

在时序图中，展现如下时间段的一种符号：在这一时间段，某一对象正直接地或通过下级过程完成一个动作。

2.641

前台　foreground

在作业调度中的一种计算环境，它使高优先级的进程或要求用户交互的进程在此环境中执行。

参见： 前台处理 foreground processing（2.642）。

相对： 后台 background（2.118）。

2.642

前台处理　foreground processing

当较低优先级进程等待计算机资源的可用或要求用户交互的进程执行时，高优先级进程的执行。

相对： 后台处理 background processing（2.119）。

2.643

形式、适宜和功能 form,fit,and function

在配置管理中，包含作为实体的项的物理的和功能的特性，但并不包含构成此项的各元素的任何特性的配置。

参见： 配置标识 conguration identification（2.309）。

2.644

形式语言 formal language

一种语言，它的规则在使用之前显式规定。例子包括编程语言和数学语言。

相对： 自然语言 natural language（2.1013）。

2.645

形参 formal parameter

虚参数 dummy parameter

软件模块中使用的变量。用来表示调用模块传送至模块的数据或程序元素。

相对： 实参 actual parameter（2.39）和自变量 argument（2.74）。

2.646

正式合格评审 formal qualification review

一种测试、审查或分析过程，在此过程中，构成系统的一组配置项被验证是否满足特定的合同的性能要求。

相对： 代码评审 code review（2.224）、设计评审 design review（2.452）、需求评审 requirement review（2.1366）和测试就绪评审 test readiness review（2.1717）。

2.647

正式评审 formal review

向最终用户、顾客或其他有兴趣的各方介绍产品，以供其评论和审定的正式会议。正式评审也可以评审项目的管理活动和技术活动以及项目的进展情况。

2.648

形式规格说明，正式规格说明 formal specification

a) 根据已建立的标准书写并获准的规格说明。

b) 用形式语言写的规格说明，常用在正确性证明中。

2.649

正式测试 formal testing

按照已经由客户、用户或指定的管理级评审和批准的测试计划和过程进行测试活动并报告结果。

相对： 非正式测试 informal testing（2.749）。

2.650

前向（正向）恢复 forward recovery

a) 由用以文件的按时间顺序的变更记录的数据，更新较早的版本以重构文件至给定的状态。

b) 一种恢复类型，在此种类型中，系统、程序、数据库或其他系统资源被恢复为新的、不是以前所有的、能执行所要求的功能的状态。

相对： 后向恢复 backward recovery（2.123）。

2.651

喷泉模型 fountain model

一种软件开发模型，该模型体现了软件创建所固有的迭代和无“间隙”的特征。它主要用于描述面向对象开发过程。

2.652

四地址指令 four-address instruction

包含四个地址字段的计算机指令。例如，一条指令，加 A、B、C 单元的内容，并把结果放至单元 D。

相对： 一地址指令 one-address instruction（2.1047）、二地址指令 two-address instruction（2.1777）、三地址指令 three-address instruction（2.1733）和零地址指令 zero-address instruction（2.1859）。

2.653

四加一地址指令 four-plus-one address instruction

包含五个地址字段的指令，其中，第五个包含下一条要执行的指令的地址。例如，一条指令加 A、B、C 单元的内容，把结果放至 D 单元，然后执行在 E 单元中的指令。

相对： 一加一地址指令 one-plus-one address instruction（2.1050）、两加一地址指令 two-plus-one address instruction（2.1780）和三加一地址指令 three-plus-one address instruction（2.1734）。

2.654

第四代语言 fourth generationlanguage(4GL)

设计用于改进高级（第三代）语言达到的生产率并常常使计算机的功能能用于非程序员的计算机语言。特性典型地包括一集成的数据库管理系统、查询语言、报告生成器和屏幕定义设施。附加的特性可能包括图形生成器、决策支持功能、财政模块、表格能力和静态分析功能。

相对： 机器语言 machine language（2.890）、汇编语言 assembly language（2.86）、高级语言 high order language（2.702）和第五代语言 fifth generation language（2.623）。

2.655

正式合格评审 FQR

formal qualification review 的缩略语。

2.656

框架 framework

包含为系统的全部或部分规定某一可重用体系结构的各模型元素的一种衍型包。在典型情况下。框架包括类、模式或模板。当框架对某一应用领域专化时，往往称之为应用框架。

2.657

功能，函数，职能 function

a) 系统或部件的定义的目标或特征动作。例如，一系统有库存控制作为它的主要功能。

参见： 功能需求 functional requirement（2.667）、功能规格说明 functional specification（2.668）和功能测试 functional testing（2.669）。

b) 执行特定动作的软件模块，由出现在表达式中的名字引用，可以接收输入值并返回一单值。

参见：子例程 subroutine（2.1619 条）。

c) 为实现预定的目的或目标，专门安排的或适宜于个人或工具所充当的角色来完成的一组相关动作。

2.658

功能字段 function field

见： 操作字段 operation field（2.1060）。

2.659

功能基线 functional baseline

在配置管理中，一配置项的初始批准的技术文档。

相对：分配的基线 allocated baseline（2.57）、开发配置 developmental configuration（2.470）和产品基线 product baseline（2.1213）。

2.660

功能分析 functional analysis

对实在的或计划好的系统的功能所进行的一种系统性调研。

2.661

功能内聚度 functional cohesion

一类内聚度，其中，由软件模块执行的所有任务都有助于单个功能的性能。

相对： 一致的内聚度 coincidental cohesion（2.229）、通信内聚度 communicational cohesion（2.246）、逻辑内聚度 logical cohesion（2.874）、过程内聚度 procedural cohesion（2.1179）和顺序内聚度 sequential cohesion（2.1439）。

2.662

功能配置审核 functional configuration audit(FCA)

一种审核，它指导验证：配置项的开发已经满意完成、配置项已经达到在功能的或分配的配置标识中规定的性能和功能特征并且它的操作的和支持的文档已满意地完成。

参见： 配置管理 configuration management（2.313）和物理配置审核 physical configuration audit（2.1139）。

2.663

功能配置标识 functional configuration identification

在配置管理中，配置项的当前批准的技术文档。它描述所有必须的功能特征，显示达到规定的功能特征所要求的测试，与相关的配置项需要的接口特征，配置项的关键功能特征和它的低级关键配置项（若有）和设计约束。

参见： 功能基线 functional baseline（2.659）。

相对： 分配的配置标识 allocated configuration identification（2.58）和产品配置标识 product configuration identification（2.1215）。

2.664

功能分解 functional decomposition

设计系统的一种方法，它把系统分成若干部分，使其直接与系统功能和子功能对应。

参见： 层次分解 hierarchical decomposition（2.696）和逐步细化 stepwise refinement（2.1587）。

2.665

功能设计 functional design

定义数据处理系统各部件的功能及相互之间接口的过程。

参见： 体系结构设计 architectural design（2.72）和概要设计 preliminary design（2.1163）。

2.666

函数语言 functional language

用于表示程序为一系列函数和函数调用的编程语言。例子包括 LISP。

2.667

功能需求 functional requirement

规定系统或系统部件必须能够执行的功能的需求。

相对： 设计需求 design requirement（2.451）、实现需求 implementation requirement（2.730）、接口需求 interface requirement（2.798）、性能需求 performance requirement（2.1133）和物理需求 physical requirement（2.1140）。

2.668

功能规格说明 functional specification

规定系统或部件必须执行的功能的文档。常常是需求规格说明（requirement specification）的一部分。

2.669

功能测试 functional testing

黑盒测试 black-box testing

a) 忽略系统或部件的内部机制只集中于响应所选择的输入和执行条件产生的输出的一种测试。

相对： 结构测试 structural testing（2.1604）。

b) 有助于评价系统或部件与规定的功能需求遵循性的测试。

参见： 性能测试 performance testing（2.1135）。

2.670

功能单元 functional unit

能实现某一特定目标的硬件、软件或两者兼而有之的实体。

2.671

无用单元收集 garbage collection

在计算机资源管理中与内存紧缩 memory compaction（2.927）同义。

2.672

通用性 generality

系统或部件执行功能的范围的宽广的程度。

参见： 可重用性 reusability（2.1387）。

2.673

可泛化元素 generalizable element

一种可参与泛化关系的模型元素。

参见： 泛化 generalization（2.674）。

2.674

泛化 generalization

较一般元素与较特殊元素之间的一种分类学关系。较特殊元素与较一般元素完全一致，并包含添加信息。较特殊元素的实例可用于允许较一般元素使用的场合。

参见： 继承 inheritance（2.752）。

2.675

生成的地址 generated address

综合地址 synthetic address

在计算机程序执行期间已经计算的地址。

参见： 绝对地址 absolute address（2.9）、有效地址 effective address（2.531）、相对地址 relative address（2.1331）和间接地址 indirect address（2.744）。

2.676

通用程序单元 generic program unit

以通用方式定义的软件模块，它要求用规定的数据或指令或这两者进行替换，以便在计算机程序中使用。

参见： 例示 instantiation（2.768）。

2.677

白盒 glass box

白盒 white box

a) 一个系统或部件，它的实现内容是已知的。

相对：黑盒 black box（2.141）。

b) 关于把系统或部件按 a 中那样对待的方法。

2.678

白盒测试 glass-box testing

见： 结构测试 structural testing（2.1604）。

2.679

全局紧缩 global compaction

在微程序设计中的一种压缩，其中的微操作可能移出它们发生的单入口、单出口的顺序块的边界。

相对： 局部紧缩 local compaction（2.869）。

2.680

全局数据 global data

公共数据 common data

可以由两个或多个非嵌套的计算机程序模块不是显式地作为参数在模块之间传送而能访问的数据。

相对： 局部数据 local data（2.870）。

2.681

全局变量 global variable

可以由两个或多个非嵌套的计算机程序模块不是显式地作为参数在模块之间传送而能访问的数据。

相对： 局部变量 local variable（2.871）。

2.682

目标 goals

对关键过程域中关键实践的一种概括，可用于确定组织或项目是否已经有效地实施该关键过程域。目标表示每个关键过程域的范围、边界和意图。

2.683

“转至”语句 go to

一种引起转移的计算机程序语句。

参见： 分支 branch（2.157）。

相对： 调用 call（2.171）、（分）情况语句 case（2.186）和 “若-则-否则”语句 if-then-else（2.718）。

2.684

图 graph

a) 一种图形，它在与一个或多个其他变量的比较中，表示一个变量的变化。例如，浴盆曲线（见图 11）。

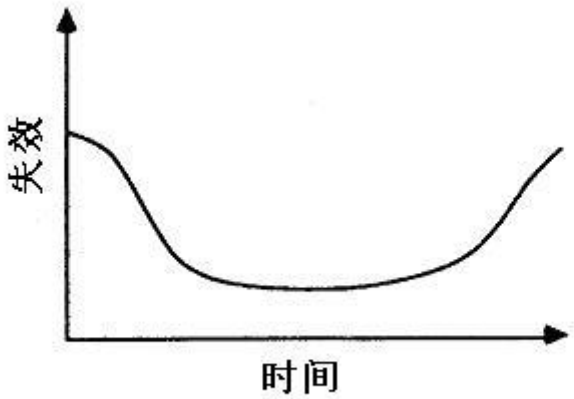

图11 图（graph）a

b) 由一组有限的节点和称为边缘（edges）或弧形（arcs）的节点间连接的图形或其他表示（见图 12）。

参见： 框图 block diagram（2.145）、盒图 box diagram（2.156）、泡图 bubble chart（2.161）、

定向图 directed graph（2.485）、输入处理输出图 input-process-output chart（2.763）和结构图 structure chart（2.1605）。

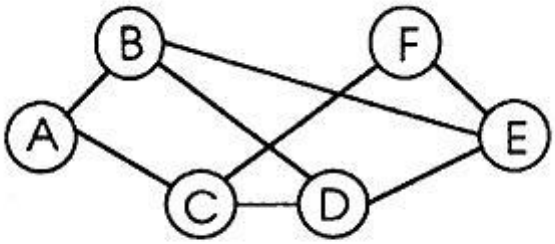

图12　图（graph）b

2.685

Grosch 定律　Grosch’s law

H.R.J.Grosch 的公式表述：计算机的能力与计算机的成本的平方成正比。

参见： 计算机性能评价 computer performance evaluation（2.275）。

2.686

组　groups

负责一组任务或活动的部门、经理和个人的集合。组的规模不等，可以是一个指派的兼职人员，可以是由不同部门指派的若干兼职人员，也可以是若干个专职人员。

2.687

守卫条件　guard condition

为使一个相关的转移能够激发而必须得到满足的一种条件。

2.688

暂停　halt

a) 多数作停止 stop（2.1590）的同义词。

b) 少数情况下也作暂停 pause（2.1125）的同义词。

2.689

硬失效　hard failure

导致系统完全关闭的失效。

相对： 软失效 soft failure（2.1468）。

2.690

硬件　hardware

用于处理、存储或传送计算机程序或数据的物理设备。

相对： 软件 software（2.1469）。

2.691

硬件配置项　hardware configuration item(HWCI)

为配置管理指定的且在配置管理过程中作为单个实体对待的硬件的集合。

参见： 配置项 configuration item（2.311）。

相对： 计算机软件配置项 computer software configuration item（2.290）。

2.692

硬件设计语言 hardware design language(HDL)

一种具有特殊的结构和有时有验证协议的语言，它用于开发、分析和说明硬件设计。

参见： 程序设计语言 program design language（2.1231）。

2.693

硬件监控器 hardware monitor

a) 测量或记录计算机系统的规定的事件或特征的设备、例如，记录各种电子事件的发生数或测量这些事件之间的时间的设备。

b) 记录或分析在计算机程序执行期间的硬件事件的软件工具。

参见： 监控器 monitor（2.981）和软件监控器 software monitor（2.1509）。

2.694

硬件设计语言 HDL

harfware design language 的缩略语。

见： 设计语言 design language（2.447）。

2.695

标题，报头，前导文件（头文件） header

a) 放在计算机程序或例程开始处的注释块。

b) 放在文件或消息开始处的标识或控制信息。

相对： 尾部 trailer（2.1755）。

2.696

层次[结构]分解 hierarchical decomposition

设计系统（模块分解）的一种方法。这种方法通过一系列自顶向下逐步求精的办法把系统分成部件的层次结构。

参见：功能分解 functional decomposition（2.664）和逐步细化 stepwise refinement（2.1587）。

2.697

层次的输入-处理-输出 hierarchical input-process-output(HIPO)

见： 输入-处理-输出 input-process-output（2.762）。

2.698

层次建模 hierarchical modeling

用于评价计算机性能的技术，在此技术中，计算机系统用它的子系统的层次结构表示，子系统被分析以确定它们的性能，且其结果用于评价整个系统的性能。

2.699

层次[结构] hierarchy

一种结构，其中，部件排列为它们的下属构成的级、每个部件有零个、一个或多个下属，直至无部件有多于一个的下属部件。

参见： 层次结构分解 hierarchical decomposition（2.696）和层次模型 hierarchical modeling（2.698）。

2.700

层次[结构]图 hierarchy chart

见： 结构图 structure chart（2.1605）。

2.701

高级语言 high level language

见：高级语言 high order language（2.702）。

2.702

高级语言 high order language

高级语言 high level language

较高级语言 higher order language

第三代语言 third generation language

一种程序设计语言,它对程序在其上运行的计算机的知识要求较少、能转换成若干不同的机器语言、允许符号命名操作和地址、提供设计以易于数据结构和程序逻辑表示的特性、并通常对每个程序语言转换为若干机器指定。例子包括：Ada、COBOL、FORTRAN、ALGOL、PASCAL。

相对： 汇编语言 assembly language（2.86）、第五代语言 fifth generation language（2.623）、第四代语言 forth generation language（2.654）和机器语言 machine language（2.890）。

2.703

较高级语言 higher order language

见： 高级语言 high order language（2.702）。

2.704

高级语言 HLL

high level language 的缩略语。

见： 高级语言 high order language（2.702）。

2.705

人机接口（人机界面） HMI

human-machine interface 的缩略语。

见： 用户接口 user interface（2.1812）。

2.706

高级语言 HOL

high order language 的缩略语。

2.707

同构冗余 homogeneous redundancy

在容错系统中，同一函数用相同的方法的实现，例如，用两个相同的处理器。

相对： 多样性 diversity（2.493）。

2.708

横向微指令 horizontal microinstruction

规定实现给定的某一机器语言指令所需的一组同时操作的一种微指令。注：横向微指令相对是较长的，常常是 64 位或更多，所以称为是“横向”，因为规定的同时操作组是写在一行中，而不是在页中顺序往下列。

相对： 对角微指令 diagonal microinstruction（2.477）和纵向微指令 vertical microinstruction（2.1832）。

2.709

宿主计算机 host computer

用于开发软件的计算机。

参见： 目标计算机 target computer（2.1675）。

2.710

[宿]主机 host machine

a) 程序或文件所装入的计算机。

b) 用以开发供另一台计算机用的软件的计算机。

相对： 目标机 target machine（2.1677 ）(a)。

c) 用以模仿另一台计算机的计算机。

相对： 目标机 target machine（2.1677 ）(b)。

d) 在计算机网络中，为该网络的用户提供处理能力的计算机。

2.711

内务操作 housekeeping operation

开销操作 overhead operation

一种计算机操作，它建立或重新建立一组初始条件以实现计算机程序的执行。例如，初始化存储区域、清除标志、回绕磁带、打开和关闭文件。

2.712

人机界面 human-machine interface(HMI)

见： 用户界面 user interface（2.1812）。

2.713

硬件配置项 HWCI

hardware configuration item 的缩略语。

2.714

混合耦合 hybrid coupling

一类耦合，在其中，数据项值的范围的不同子集，假定能用于不同的软件模块中不同的和不相关的目的。

相对：公共环境耦合 common-environment coupling(2.243)、内容耦合 content couping(2.327)、控制耦合 control coupling（2.336）、数据耦合 data couping（2.393）和相依耦合 pathological coupling（2.1123）。

2.715

标识符 identifier

计算机程序中对象的名字、地址和标号。

2.716

空闲的 idle

用于说明在操作的和在服务中但未在用的系统或部件。

参见： 忙 busy（2.168）、关机 down（2.502）和开机 up（2.1797）。

2.717

空闲时间 idle time

待命时间 standby time

系统或部件在操作和服务中但未在用的时间间隔。

参见： 忙时间 busy time(2.169)、关机时间 down time(2.503)、设置时间 set-up time(2.1449)和开机时间 up time（2.1798）。

2.718

“若-则-否则”语句 if-then-else

一种单入口单出口，定义了条件的两路分支。若规定的条件满足则要执行的处理；若不满足，如何处理是任选的；在两种情况下都返回控制至立即跟随整个结构的语句（见图 13）。

参见： 二元选择构造 dyadic selective construct（2.510）和一元选择构造 monadic selective construct（2.980）。

相对： （分）情况语句 case（2.186）、跳移 jump（2.830）和“转至”语句 go to（2.683）。

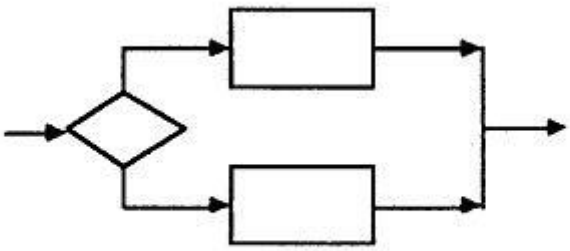

图13　if-then-else 构造

2.719

立即地址　immediate address

见：　立即数据 immediate data（2.721）。

2.720

立即控制　immediate control

一种微程序设计技术，其中，微指令中一个字段的含义，取决于微指令中另一字段的值。

参见：2.1779 条　二级编码（two-level encoding）。

相对：　残留控制 residual control（2.1374）。

2.721

立即数据　immediate data

包含在计算机指令地址字段中的数据。

参见：　立即指令 immediate instruction（2.722）。

相对：　直接地址 direct address（2.481）、间接地址 indirect address（2.744）和　n 级地址 n-level address（2.1007）。

2.722

立即指令　immediate instruction

地址字段包含操作数的值而不是它的地址的计算机指令。

参见：　绝对指令 absolute instruction（2.12）、有效指令 effective instruction（2.532）和立即数据 immediate data（2.721）。

相对：直接指令 direct instruction（2.483）和间接指令 indirect instruction（2.745）。

2.723

强制性构造　imperative construct

一种不包含分支或迭代的单步或多步序列。

2.724

强制性语句　imperative statement

见：　指令 instruction（2.770）。

2.725

不完全排错　imperfect debugging

在可靠性模拟中，纠正或清除已经发现故障的意图并非总是成功的一种假定。

2.726

实现　implementation

a) 把设计转换为硬件部件、软件部件或两者的过程。

参见： 编码 coding（2.227）。

b) a 中过程的结果。

2.727

实现类 implementation class

一种如下衍型类：为其中实例可不具有多个类的某一程序设计语言（如 C++，Smalltalk,Java）中的某一类规定实现。当实现类提供的具有如同为该类型的运算所规定的同一行为的某一类型来定义的全部操作时，即称该实现类实施了此类型。

参见： 类型 type（2.1781）。

2.728

实现继承 implementation inheritance

对较一般元素的实现的继承。

相对： 接口继承 interface inheritance（2.797）。

2.729

实现阶段 implementation phase

软件生存周期中的一段时间，在此期间根据设计文件制造软件产品并排除其中的隐错。

参见:安装检验阶段 installation and check-out phase(2.765)和测试阶段 test phase(2.1713)。

2.730

实现需求 implementation requirement

规定或强制系统或系统部件的编码或结果的要求。

相对： 设计要求 design requirement（2.451）、功能需求 functional requirement（2.667）、接口要求 interface requirement（2.798）、性能需求 performance requirement（2.1133）和物理需求 physical requirement（2.1140）。

2.731

隐含寻址 implied addressing

一种寻址方法，在其中，计算机指令的操作字段，隐含着操作数的地址。例如，若计算机只有一个累加器,涉及累加器的指令不需要描述它的地址信息。类型包括向前一个寻址(one-ahead addressing)、重复寻址（repetitive addressing)。

2.732

隐含的要求 implied needs

当实体用在特定条件下时，尚未说明但又是实际需要的要求。

注：隐含的要求是未形成文档的真实要求。

2.733

引入 import

它表明在包的语境中，一个指定的包（包括递归地嵌入其内的包）可以引用另外一些包中的类的一种依赖。

相对： 引出 export（2.591）。

2.734

待查事件 incident

见： 软件测试待查事件 software test incident（2.1536）。

2.735

初期的失效 incipient failure

一种即将发生的失效。

2.736

包括 include

在基用况到包含用况的一种如下关系，它表明基用况的行为如何纳入包含用况的行为。行为被包含在基用况中所定义的部位。基用况依赖于包含用况行为的完成，但不依赖于其结构（即属性或操作）。

参见： 扩展 extend（2.593）。

2.737

增量编译程序 incremental compiler

会话式编译程序 conversational compiler

联机编译程序 on-line compiler

一种编译程序，它在源语句输入或扫描时，尽可能多地完成每个源语句的转换。典型地用于联机计算机程序的开发和检验。

2.738

增量开发 incremental development

一种软件开发技术，在这种技术中，需求定义、设计、实现和测试的发生是以一种重叠的、反复的方式（不是顺序的）进行，导致整个软件产品的递增的完成。

参见： 以数据结构为中心的设计 data structure-centered design（2.405）、输入-处理-输出 input-process-output（2.762）、模块分解 modular decomposition（2.973）、面向对象的设计 object-oriented design（2.1037）、快速原型 rapid prototyping（2.1307）、螺旋模型 spiral model（2.1553）、逐步细化 stepwise refinement（2.1587）、结构设计 structured design（2.1607）、事务分析 transaction analysis（2.1760）和转换分析 transform analysis（2.1764）。

相对： 瀑布模型 water-fall model（2.1844）。

2.739

独立验证和确认 independent verification and validation(IV&V)

验证和确认过程由在技术上、管理上、财务上独立于开发组织的组织执行。

2.740

索引地址 indexed address

一个地址，它必须加至一个变址寄存器的内容以得到要访问的存储单元的地址。

参见：偏移量 offset（2.1043）、相对地址 relative address（2.1331）和自相关地址 self-relative address（2.1429）。

2.741

指示器，指示符 indicator

一个设备或变量，它能加以设置，以描述基于处理的结果或规定的条件的发生的状态。例如， 标志 flag（2.633 条）或信号量 semaphore（2.1433）。

2.742

内在差错 indigenous error

计算机程序中存在的一种差错，它不是作为差错播种过程的一部分而插入的。

2.743

内在故障 indigenous fault

计算机程序中存在的一种故障。这种故障不是作为故障播种过程的一部分而插入的。

2.744

间接地址 indirect address

多级地址 multilevel address

标识另一个地址的存储单元的地址。指定的存储单元可以包含所希望的操作数或另一个间接地址的地址、地址链最后引导至操作数。

参见： 间接指令 indirect instruction（2.744）和 n－级地址 n-level address（2.1007）。

相对： 直接地址 direct address（2.481）和立即数据 immediate data（2.721）。

2.745

间接指令　indirect instruction

包含操作数的间接地址的计算机指令。

参见： 绝对指令 absolute instruction（2.12）和有效指令 effective instruction（2.532）。

相对： 直接指令 direct instruction（2.483）和立即指令 immediate instruction（2.722）。

2.746

间接测度　indirect measure

从一个或一个以上的其他属性的测量得出的一种对属性的测量。

注：对计算机系统属性（例如对用户输入的响应时间）的外部测量就是对软件属性的一种间接测量，因这种测量要受计算环境的属性和软件属性的影响。

2.747

归纳断言法　inductive assertion method

一种正确性证明技术，在采用时要写出描述程序输入、输出和中间条件的断言，开发当输入断言满足时，使输出断言得到满足的一组定理，并且用归纳法证明这些定理。

2.748

早期死亡率　infant mortality

在系统或部件的早期失效周期期间发生的失效。

2.749

非正式测试　informal testing

按照尚未由客户、用户或指定的管理级评审和批准的测试计划和过程进行的测试行为。

相对： 正式测试 formal testing（2.649）。

2.750

信息分析　information analysis

对实在的或计划好的系统中的信息及其流程所进行的一种系统调研。

2.751

信息隐蔽　information hiding

一种软件开发技术，其中，每一模块的接口尽可能少泄漏模块的内部工作并防止其他模块使用不在模块的接口规格说明中的关于模块的信息。

参见： 封装 encapsulation（2.542）。

2.752

继承　inheritance

一种机制，通过它特殊的元素能够合并较一般的元素的结构和行为。

参见： 泛化 generalization（2.674）。

2.753

继承的差错　inherited error

在顺序过程中，从上一步带来的一种差错。

2.754

初始程序装入　initial program load

见： 引导程序 bootstrap（2.151）。

2.755

初始程序装入程序　initial program loader

一个引导装入程序，它用于装入操作系统中为装入操作系统的其余部分必需的部分。

2.756

初始化　initialize

给变量、寄存器、或其他存储单元设置一起始值。

参见： 清除 clear（2.213）和复位 reset（2.1372）。

2.757

初[始]状态 initial state

表明向组合状态的系统设定状态作单次转移的源状态的一种特种状态。

2.758

内联码 inline code

一种计算机指令序列，它与逻辑地领先和跟随它的指令是物理邻接的。

2.759

输入 input

a) 从外部源接收的数据。

b) 从外部源接收数据涉及的设备、进程或通道。

c) 从外部源接收数据。

相对： 输出 output（2.1080）。

2.760

输入断言 input assertion

一种逻辑表达式，它规定：为了有效，程序的输入必须满足的一个或多个条件。

参见： 归纳断言方法 inductive assertion method（2.747）。

相对： 循环断言 loop assertion（2.880）和输出断言 output assertion（2.1081）。

2.761

输入-输出耦合 input-output coupling

见： 数据耦合 data coupling（2.393）。

2.762

输入-处理-输出 input-process-output

一种软件设计技术，它由标识在每一要执行的处理中包括的步骤并标识每一步的输入和输出组成。

注：称为分层的输入-处理-输出的细化在通用和详细级上标识步骤、输入和输出。

参见： 以数据结构为中心的设计 data structure-centered design（2.405）、输入-处理-输出图 input-process-output chart（2.762）、模块分解 modular decomposition（2.973）、面向对象的设计 object-oriented design（2.1037）、快速原型 rapid prototyping（2.1307）、螺旋模型 spiral model（2.1553）、逐步细化 stepwise refinement（2.1587）、结构设计 structured design（2.1607）、事务分析 transaction analysis（2.1760）和转换分析 transform analysis（2.1764）。

2.763

输入-处理-输出图 input-process-output(IPO) chart

软件系统或模块图，它由在左边列输入的方框、在中间列处理步骤的方框和在右边列输出的方框，并由连接输入至处理步骤及处理步骤至输出的箭头组成（见图 14）。

参见： 框图 block diagram（2.145）、盒图 box diagram（2.156）、泡图 bubble chart（2.161）、流程图 flowchart（2.637）、图 graph（2.684）和结构图 structure chart（2.1605）。

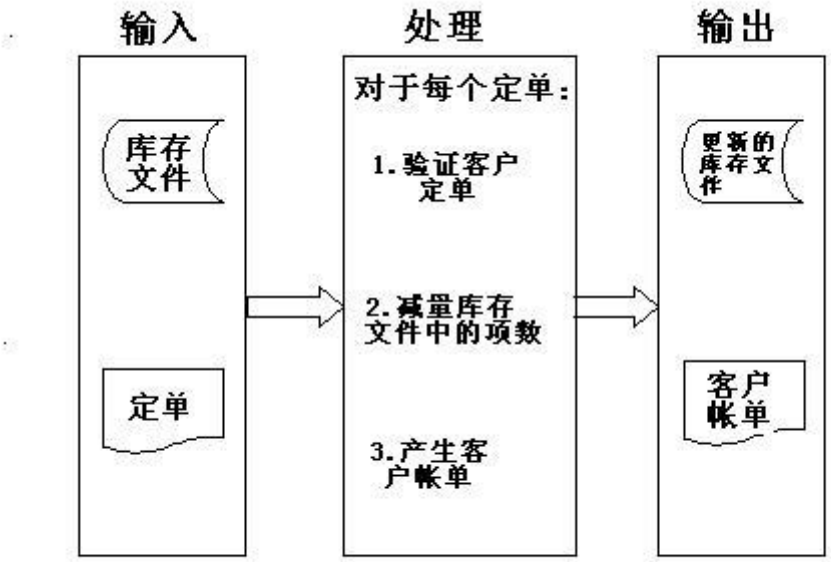

图14　输入-处理-输出图

2.764

审查　inspection

一种静态分析技术，它依靠对开发产品的可视的检查，以检测错误、对开发标准的违反和其他问题。类型包括：代码审查、设计审查。

2.765

安装检验阶段　installation and check-out phase

软件生存周期中的一段时间，期间软件产品被结合到工作环境中，并在该环境中加以测试，以保证它能按照要求进行工作。

2.766

安装手册　installation manual

提供为安装系统或部件、设置初始参数和为了操作使用准备系统或部件所必需的信息的文档。

参见： 诊断手册 diagnostic manual（2.476）、操作员手册 operator manual（2.1068）、程序员手册 programmer manual（2.1250）、支持手册 support manual（2.1631）和用户手册 user manual（2.1813）。

2.767

实例　instance

一个实体，它具有唯一的标识，应用于该实体的一组操作和存储操作效果的状态。

参见：对象 object（2.1030）。

2.768

例示　instantiation

替换特定的数据、指令或两者至通用的程序单元中使它在计算机程序中可用的过程。

2.769

制度化　institutionalization

支持方法、实践和规程的基础设施及组织文化的建立，以使这些方法、实践和规程成为组织经营的方式，即使在最初规定它们的人员离去之后仍然是组织的经营方式。

2.770

指令 instruction

a) 程序设计中的一个语句，即规定要求计算机执行的操作和相关的操作数的值或地址；例如把 A 传送至 B。

参见： 指令格式 instruction format（2.773）、指令集 instruction set（2.777）。

b) 接近于程序设计语言中的任一可执行语句。

2.771

指令计数器 instruction counter

程序计数器 program counter

指示下一条要执行的计算机指令的单元的寄存器。

2.772

指令周期 instruction cycle

从内存取一计算机指令并执行它的过程。

参见： 指令时间 instruction time（2.779）。

2.773

指令格式 instruction format

在计算机指令中对字段数的安排。

参见： 地址字段 address field（2.45）、地址格式 address format（2.46）、操作字段 operation field（2.1060）。

2.774

指令长[度] instruction length

存储一条计算机指令所需的字、字节或位的个数。

参见： 指令格式 instruction format（2.773）。

2.775

指令修改符 instruction modifier

用于替换计算机指令的字或字的一部分。

2.776

指令系统 instruction repertoire

见： 指令集 instruction set（2.777）。

2.777

指令集（指令系统） instruction set

指令系统 instruction repertoire

由给定的计算机能识别的或由给定的编程语言提供的指令的完全集。

2.778

指令集体系结构 instruction set architecture

用指令集表征的抽象机。

2.779

指令时间 instruction time

计算机从内存取一条指令并予执行的时间。

参见： 指令周期 instruction cycle（2.772）。

2.780

指令踪迹 instruction trace

参见： 踪迹 trace（2.1751）。

2.781

插装 instrument

在软件或系统测试中，将某些器件或指令安装或插入至硬件或软件中，以监控系统或部件的操作。

2.782

插装[工具] instrumentation

安装或插入硬件或软件中以监控系统或部件的操作的设备或指令。

参见： 程序探测 program instrumentation（2.1235）。

2.783

插装工具 instrumentation tool

一种软件工具，它在被测程序中的适当位置上产生并插入，起计数器或其他探头作用的语句，以提供有关程序执行情况的统计数字，如程序中的代码被执行到的覆盖程度。

2.784

整型 integer type

一种数据类型，它的成员能假定只是整数值且只能由整数算术操作，例如，加、减和乘进行操作。

相对： 字符型 character type（2.201）、枚举型 enumeration type（2.558）、逻辑型 logical type（2.878）和实型 real type（2.1314）。

2.785

集成编程支持环境 integrated programming support environment(IPSE)

见： 编程支持环境 programming support environment（2.1253）。

2.786

集成软件管理 integrated software management

基于组织的标准软件过程和相关过程资产，将软件工程和管理活动统一并集成为一个协调定义的软件过程。

2.787

集成 integration

把软件、硬件部件或两者合成为一个完整的系统的过程。

2.788

集成测试 integration testing

把软件部件、硬件部件或两者组合起来进行的测试，并测试以评价它们之间的交互。

参见： 部件测试 components testing（2.264）、 接口测试 interface testing（2.800）、系统测试 system testing（2.1669）和单元测试 unit testing（2.1792）。

2.789

完整性 integrity

系统或部件防止未授权访问或修改计算机程序或数据的程度。

2.790

交互[作用] interaction

对执行某一特定任务的实例之间如何发送激励的一种规格说明。交互在某一协作的语境中加以定义。

参见： 协作 collaboration（2.230）。

2.791

交互图 interaction diagram

一个一般性术语，用于几种强调对象交互的图。包括协作图与时序图。

2.792

交互式 interactive

用于说明每一个用户输入，引起系统响应或动作的系统或操作方式。

相对： 会话的 conversational（2.348）、联机的 on-line（2.1045）和实时的 real time（2.1313）。

2.793

交互语言 interactive language

一种非过程的语言，其中的程序作为在用户与计算机系统交互对话的结果而建立。系统提供问题、格式等等，以帮助用户表达得到的结果。

参见： 声明语言 declarative language（2.417）和基于规则的语言 rule-based language（2.1403）。

2.794

交互系统 interactive system

指这样一个系统。在这种系统中，每一个用户的输入均能得到该系统的响应。

2.795

界面，接口 interface

a) 一个共享的边界。信息跨越边界传送。

b) 连接两个或多个其他部件，为了相互间传送信息的硬件或软件部件。

c) 连接两个或多个部件，为了在相互间传送信息。

d) 作为如 b 中连接的或被连接的部件。

2.796

接口控制 interface control

a) 在配置管理中，(1)标识由一个或多个组织提供的两个或多个配置项的接口的所有功能的和物理的特征和(2)保证对这些特征的建议的变更在实现前评价和批准的过程。

b) 在配置管理中，管理的和技术的过程和文档需要在由不同的开发者提供的配置项之间或配置项之内标识功能的和物理的特征并解决关于规定的接口的问题。

参见： 配置控制 configuration control（2.307）。

2.797

接口继承 interface inheritance

对较为一般的元素的接口的继承。不包括对实现的继承。

相对： 实现继承 implemantation inheritance（2.728）。

2.798

接口需求 interface requirement

规定系统或系统部件必须与之交互的外部项或提出由这样的交互引起的在格式、时间关系或其他因素方面提出的限制的需求。

相对： 设计需求 （2.451 条 design requirement）、功能需求 functional requirement（2.667）、实现需求 implementation requirement（2.730）、性能需求 performance requirement（2.1133）和物理需求 physical requirement（2.1140）。

2.799

接口规格说明 interface specification

规定已存在的或计划的系统或部件的接口需求的规格说明。

2.800

接口测试 interface testing

引导以评价系统或部件能否相互正确地传送数据与控制信息的测试。

参见： 部件测试 component testing（2.264）、集成测试 integration testing（2.788）、系统测试 system testing（2.1669）和单元测试 unit testing（2.1792）。

2.801

交错 interleave

用一个或多个其他序列的元素替换一个序列的元素，以使每个序列保持它的标识、例如，变更两个不同任务的执行步骤以达到两个任务的并发操作。

2.802

软件中间产品 intermediate software product

软件开发过程中，用作软件开发过程另一阶段的输入的产品。

注：在某些情况下，中间产品也可以是最终产品。

2.803

间歇故障 intermittent fault

在部件中临时的或未预料到的故障。

参见： 随机失效 random failure（2.1306）、瞬时错误 transient error（2.1767）。

2.804

内部测度 internal measure

对产品本身的一种直接的或间接的度量。

注：代码行数、复杂度度量、在走查和 Fog 索引中发现的故障数都是对产品本身进行的内部度量。

2.805

内部质量 internal quality

产品属性的总和，它决定了产品在特定条件下使用时，满足明确和隐含要求的能力。

注1： 在GB/T XXXXX中，与术语“外部质量”对应的“内部质量”基本上与GB/T 6583-1994中 “质量” 的含义相同。

注 2： 术语“属性”与术语“特性” 的含义相同，而术语“特性”在 GB/T 16260 中有更具体的含义。

2.806

内部转移 internal transition

表明不更改某一对象的状态而对事件作用响应的一种转移。

2.807

互操作能力，互操作性 interoperability

a) 两个或多个系统或部件交换信息并相互使用已交换的信息的能力。

相对： 兼容性 compatibility（2.249）。

b) 两个或两个以上系统可互相操作的能力。

2.808

解释 interpret

逐条翻译并立即执行计算机程序的每一源语言语句。

相对： 汇编 assemble（2.79）和编译 compile（2.252）。

2.809

解释程序（解释器） interpreter

翻译和执行每一条语句或在翻译和执行下一条语句前构造计算机程序的计算机程序。

相对： 汇编程序 assembler（2.82）和编译程序 compiler（2.254）。

2.810

解释代码 interpretive code

按可由解释程序识别并处理的形式表达的计算机指令和数据定义。

相对： 汇编代码 assembly code（2.85）、编译代码 compiler code（2.255）和机器代码 machine code（2.887）。

2.811

中断 interrupt

中断 interruption

a) 进程暂停以处理进程以外的事件。

参见：中断潜伏期 interrupt latency（2.812）、中断屏蔽 interrupt mask（2.813）、中断优先级 interrupt priority（2.814）、中断服务例程 interrupt service routine（2.816）和优先级

中断 priority interrupt（2.1172）。

b） 引起进程暂停。

c） 十分接近于中断请求。

2.812

中断潜伏期 interrupt latency

在计算机系统接受到中断请求至请求的处理之间的延迟。

参见： 中断优先级 interrupt priority（2.814）。

2.813

中断屏蔽 interrupt mask

一种屏蔽，它用于由保持或抑制代表中断请求的位以启用或禁止中断。

2.814

中断优先级 interrupt priority

赋予给定的中断请求的重要性，它确定请求是否引起当前进程的暂停且若有若干个未处理的中断请求，应先处理哪一个。

2.815

中断请求 interrupt request

使当前执行的进程暂停以允许另一个进程执行的信号或其他输入请求。

2.816

中断服务例程 interrupt service routine

保存关键寄存器的内容、执行由中断请求要求的处理、恢复寄存器的内容和重启动被中断的处理以响应中断请求的例程。

2.817

中断 interruption

见： 中断 interrupt（2.811）。

2.818

不变式 invariant

计算机程序的特定段或在一规定点始终为真（true）的断言。

2.819

输入-处理-输出图 IPO chart

input-process-output chart 的缩略语。

2.820

集成编程支持环境 IPSE

integrated programming support environment 的缩略语。

2.821

迭代 iteration

a） 重复执行步骤的序列的过程。

参见： 循环 loop（2.879）和递归 recursion（2.1320）。

b） 在 a 中的步骤序列的单个执行。

2.822

迭代构造 iterative construct

见： 循环 loop（2.879）。

2.823

独立验证和确认 IV&V

independent verification and validation 的缩略语。

2.824

作业控制语言　JCL

job control language 的缩略语。

2.825

作业　job

由计算机实现的用户定义的工作单元。例如，计算机程序的编译、装入和执行。

参见： 作业控制语言 job control language（2.826）、作业步 job step（2.828）和作业流 job stream（2.829）。

2.826

作业控制语言　job control language(JCL)

用于标识作业的序列、描述它们对操作系统的要求并控制它们执行的语言。

2.827

作业功能　job function

一组工程的进程，为了工作组织、赋值或评价目的作为一个单元标识。例如，设计、测试和配置管理。

2.828

作业步　job step

由作业控制语句显式标识的作业的用户定义部分。一个作业由一个或多个作业步组成。

2.829

作业流　job stream

运行流　run stream

程序或作业设置序列，它使计算机能逐个自动处理而不需操作员的介入。

2.830

跳转　jump

传送　transfer

a) 从隐含的或声明的正在执行的计算机程序语句顺序离开。

b) 引起像 a)那样离开的程序语句。

参见： 分支 branch（2.157）和“转至”语句 go to（2.683）。

相对： （分）情况语句 case（2.186）和“若-则-否则”语句 if-then-else（2.718）。

c) a 中描述的离开。

参见： 条件跳转 conditional jump（2.304）和无条件跳转 unconditional jump（2.1785）。

2.831

内核　kernel

核心　nucleus

常驻控制程序　resident control program

a) 所有时候都保持在主内存中的操作系统部分。

b) 浓缩系统的基本功能的软件模块。

参见： 安全内核 security kernel（2.1421）。

2.832

关键字　key

数据集合中的一个或多个字符，它含有有关该集合的信息，包括其标识。

2.833

关键实践　key practices

对关键过程域的有效实施和制度化贡献最大的基础设施和活动。

2.834

关键过程域 key process area （KPA）

一组相关的活动，当这些活动共同完成时，对建立过程能力至关重要的一组目标就能实现。

2.835

千次运算每秒 KOPS

每秒千次运算，即 kilo-operations per second 的缩略语。计算机处理速度的一种度量。

参见： 百万次浮点运算每秒 MFLOPS（2.943 条）和百万次指令运算每秒 MIPS（2.958）。

2.836

标号 label

a） 赋予计算机程序语句的名字或标识符，它允许其他语句引用此语句。

b） 在数据集内或附加至数据集的一个或多个字符，它标识或描述此数据。

2.837

语言 language

a） 由用传统的符号、声音、手势或标记及允许的表达的构造规则的通信概念的一种分类手段。

b） 具有语法和句法，由一组表达式、约定和用于传输信息的相关规则构成的通信手段。

参见： 计算机语言 computer language（2.273）。

2.838

语言处理程序 language processor

a） 一种计算机程序，它执行的功能，诸如处理指定程序设计语言所需的翻译、解释功能和其他任务。例如 FORTRAN 处理程序、COBOL 处理程序。

b） 一种软件工具，它完成这样一些功能，诸如处理指定的语言（如需求规格说明语言、设计语言或程序设计语言）所需的翻译、解释或其他任务。

2.839

语言标准 language standard

一种描述语言特征的标准，此语言用于描述需求规格说明、设计或测试数据。

2.840

潜伏期 latency

在指令控制单元发出调用数据的瞬间与数据启动传送的瞬间之间的时间间隔。

2.841

侧向压缩 lateral compression

在软件设计中，一种去模块化形式，它把两个或多个顺序执行的模块组合为单个模块。

相对： 向下压缩 downward compression（2.505）和向上压缩 upward compression（2.1800）。

2.842

层 layer

在同一抽象级对类目或包进行的组织。层表示对体系结构的横向片，分区则表示纵向片。

2.843

前导判定 leading decision

在循环体之前执行的循环控制。

参见： “当”语句 WHILE（2.1847）。

相对： 尾随判定 trailing decision（2.1756）。

2.844

计算机软件的法律保护 legal protection of computer software

以法律手段对计算机软件的知识产权提供保护和支持计算机软件的安全运行而提供的法律保护。

2.845

等级，层 level

a) 一个项在某一层次排列中下属的级数。

b) 层次结构中的级。若一项目没有从属项则属最低级，若没有比它高的项则为最高级。

2.846

文档等级 level of documentation

指明文档的范围、内容、格式以及质量。文档等级可根据项目成本、预期用途、作用范围及其他因素进行选择。

2.847

库管理员 librarian

参见： 软件库管理员 software librarian（2.1504）。

2.848

库 library

参见： 软件库 software library（2.1505）和系统库 system library（2.1657）。

2.849

许可证发放标准 licensing standard

描述由官员或合法的授权机构对个人或组织授权允许做或拥有某特定的事情的授权特征的标准。

2.850

生存周期 life cycle

见： 软件生存周期 software life cycle（2.1506）和系统生存周期 system life cycle（2.1658）。

2.851

生存周期模型 life-cycle model

一个框架，它含有从需求定义到使用终止，跨越整个生存期的系统开发、操作和维护中所需实施的过程、活动和任务。

2.852

连接，链接 link

a) 从两个或多个独立翻译的目标模块或装入模块，它由解决它们之间的交叉引用建立一装入模块。

参见： 连接编辑程序 linkage editor（2.854）。

b) 计算机程序的一部分，它通常是单指令或地址，在程序的分别的模块之间传送控制和参数。

同义词： 连接 linkage（2.853）。

c) 提供如在 b 中的连接。

2.853

连接 linkage

见： 连接 link（2.852）。

2.854

连接编辑程序 linkage editor

连接程序 linker

一个计算机程序,它从两个或多个独立翻译的目标模块或装入模块由解决它们之间的交叉引用而建立单个装入模块。也可能作为装入程序的一部分，需要把一些元素重新定位。

参见：连接装入程序 linking loader（2.857）。

2.855

连接表 linked list

见： 链接表 chained list（2.193）。

2.856

连接程序 linker

见： 连接编辑程序 linkage editor（2.854）。

2.857

连接装入程序 linking loader

一计算机程序，它读入一个或多个目标模块至主内存中准备执行，由解决分别的模块之间的交叉引用，建立一单个装入模块。并在有些情况下，调整地址以反映已经装入的代码的存储单元。

参见： 绝对装入程序 absolute loader（2.13）、重定位装入程序 relocating loader（2.1347）和连接编辑程序 linkage editor（2.854）。

2.858

[列]表，清单 list

a) 每个有相同数据定义的数据项集合。

b) 为打印或显示一组数据项。

2.859

列表处理 list processing

一种用表的形式来处理数据的方法。通常使用链接表，这样就能变更项的逻辑顺序而无需变更它们的物理位置。

2.860

表处理语言 list processing language

一种编程语言，其目的是促进以表的形式表示的数据的处理。例子是 LISP 和 IPL。

参见：代数语言 algebraic language（2.53）、算法语言 algorithmic language（2.56）和逻辑编程语言 logic programming language（2.873 ）。

2.861

列表 listing

数据项、程序语句或其他信息的有序的显示或打印输出。

2.862

字面值 literal

在源程序中，一项值的显式表示、例如，在指令：if x=0 then print ”FAIL”中的字 FAIL。

参见： 立即数据 immediate data（2.721）和数字形式常数 figurative constant（2.624）。

2.863

装入 load

a) 读机器代码至主内存准备执行，在有些情况下，执行地址调整和模块连接。

参见： 装入程序 loader（2.868）。

b) 复制计算机指令或数据从外部存储器至内部存储器，或从内部存储器至寄存器。

参见： 读取指令 fetch（2.622）和移动 move（2.983）。

相对：存储 store（2.1595）。

2.864

装入并执行 load-and-go

一种操作技术，它在计算机程序的装入和执行阶段之间没有停顿。

2.865

装入映像表 load map

计算机生成的表,它标识驻留在内存中的计算机程序或驻留在内存中的数据的全部或指定部分的位置或大小。

2.866

装入模块 load module

适合于装入到主存中去等待执行的计算机程序或子程序。它通常是连接编辑程序的输出。

参见： 目标模块 object module（2.1036）。

2.867

装入的起点 loaded origin

计算机程序在程序装入至主内存时的初始的存储单元的地址。

参见： 偏移量 offset（2.1043）和起始地址 starting address（2.1576）。

相对： 汇编的起点 assembled origin（2.81）。

2.868

装入程序 loader

a) 一种计算机程序。它把机器代码读入到主存中准备执行，在某些情况下，调整地址和连接模块。类型包括：绝对装入程序（absolute loader）、连接装入程序（linking loader）和重定位装入程序（relocating loader）。

参见： 引导程序 bootstrap（2.151）和连接编辑程序 linkage editor（2.854）。

b) 读程序或数据至主内存中的任何程序。

2.869

局部紧缩 local compaction

在微程序编程语言中，微操作未超出它们发生的单入口、单出口顺序块的边界之外的压缩。

相对： 全局紧缩 global compaction（2.679）。

2.870

局部数据 local data

只能由计算机程序中的一个模块或一组嵌套的模块访问的数据。

相对： 全局数据 global data（2.680）。

2.871

局部变量 local variable

只能由计算机程序中的一个模块或一组嵌套的模块访问的变量。

相对： 全局变量 global variable（2.681）。

2.872

锁定 lockout

一种计算机资源分配技术，其中，对共享的资源（特别是数据）加以保护，一次只允许一个设备或进程访问。

参见： 死锁 deadlock（2.4105）和信号量 semaphore（2.1433）。

2.873

逻辑编程语言 logic programming language

用于根据控制结构和受限的断言运算表达程序的编程语言。例如，PROLOG。

参见： 代数语言 algebraic language（2.53）、算法语言 algorithmic language（2.56）和表处理语言 list processing language（2.860）。

2.874

逻辑内聚度 logical cohesion

一类内聚度，表现为软件模块能处理逻辑上类似的功能。例如处理不同类型的输入数据。

相对： 一致内聚度 coincidentional cohesion（2.229）、通信内聚度 communicational cohesion（2.246）、功能内聚度 functional cohesion（2.661）、过程内聚度 procedural cohesion（2.1179）和顺序内聚度 sequential cohesion（2.1439）。

2.875

逻辑文件 logical file

与物理环境无关的文件。同一逻辑文件的各部分可以放在不同的物理文件中、几个逻辑文件或几个逻辑文件的各部分可以放在一个物理文件中。

2.876

逻辑记录 logical recod

与物理环境无关的记录。同一逻辑记录的各部分可以放在不同的物理记录中，几个逻辑记录或几个逻辑记录的各部分可以放在一个物理记录中。

2.877

逻辑踪迹 logical trace

只记录分支或转移指令的执行踪迹。

参见： 执行踪迹 execution trace（2.582）、回顾踪迹 retrospective trace（2.1383）、子例程踪迹 subroutine trace（2.1620）、符号踪迹 symbolic trace（2.1639）和变量踪迹 variable trace（2.1823）。

2.878

逻辑型 logical type

一种数据类型，它的成员只能假定有逻辑值（通常为 TRUE 和 FAUSE）且只能由逻辑运算符，例如，AND、OR 和 NOT 对其进行运算。

相对： 字符型 character type（2.201）、枚举型 enumeration type（2.558）、整型 integer type（2.784）和实型 real type（2.1314）。

2.879

循环 loop

迭代构造 iterative construct

a) 一计算机程序语句序列，它重复地执行直至满足给定的条件或当一给定的条件为真。

参见： 循环体 loop body（2.881）、循环控制 loop control（2.882）、“直至”语句 UNTIL（2.1795）和“当”语句 WHILE（2.1847）。

b) 像在 a 中那样执行一计算机程序语句序列。

2.880

循环断言 loop assertion

一种逻辑表达式，它规定每次一具体的点在程序循环中执行时必须满足的一个或多个条件。

参见： 归根断言方法 inductive assertion method（2.747）。

相对： 输入断言 input assertion（2.760）和输出断言 output assertion（2.1081）。

2.881

循环体 loop body

循环的一部分，它实现循环的主要目的。

相对： 循环控制 loop control（2.882）。

2.882

循环控制 loop control

决定是否退出循环的循环部分。

参见： 先导判定 leading decision（2.843）和尾随判定 trailing decision（2.1756）。

相对： 循环体 loop body（2.881）。

2.883

循环控制变量 loop control variable

用于决定是否从循环退出的程序变量。

2.884

回送测试 loopback testing

测试的信号或数据从测试设备输入至系统或部件，且结果返回至测试设备为了测量和比较的测试。

2.885

低级语言 low level language

见： 汇编语言 assembly language（2.86）。

2.886

机器地址 machine address

见： 绝对地址 absolute address（2.9）。

2.887

机器代码 machine code

以计算机处理单元能识别的形式表示的计算机指令和数据定义。

相对： 汇编代码 assembly code（2.85）、编译代码 compiler code（2.255）和解释代码 interpretive code（2.810）。

2.888

依赖于机器的 machine dependent

用于说明软件取决于具体类型计算机独特的特性，因而只能在此类型计算机上执行。

相对： 独立于机器的 machine independent（2.889）。

2.889

独立于机器的 machine independent

用于说明不依赖于具体类型计算机独特的特性，因而能在多于一种类型计算机上执行的软件。

参见： 可移植性 portability（2.1149）。

相对： 依赖于机器的 machine dependent（2.888）。

2.890

机器语言 machine language

第一代语言 first generation language

面向机器的语言 machine-oriented language

能由计算机的处理单元识别的语言。这样的语言通常由 0 和 1 的模式组成，没有操作的符号名或地址。

相对： 汇编语言 assembly language（2.86）、第五代语言 fifth generation language（2.623）、第四代语言 forth generation language（2.654）、高级语言 high order language（2.702）和符号语言 symbolic language（2.1638）。

2.891

面向机器的语言 machine-oriented language

见： 机器语言 machine language（2.890）。

2.892

机器可读的 machine readable

以能自动输入至计算机的格式表示的数据。例如，在软盘上编码的数据。

2.893

宏 macro

宏定义 macro definition

在软件工程中，一个预先定义好的计算机指令序列。它在汇编或编译期间把该指令序列插入到程序中每一处出现相应宏指令的地方。

参见： 宏指令 macroinstruction（2.899）、宏生成器 macrogenerator（2.898）和开式子例程 open subroutine（1052）。

2.894

宏定义 macro definition

见： 宏 macro（2.893）。

2.895

宏生成程序 macro generating program

见： 宏生成器 macrogenerator（2.898）。

2.896

宏库 macro library

可由宏生存器使用的宏的集合。

参见： 系统库 system library（2.1657）。

2.897

宏汇编程序 macroassembler

包括或执行宏生成器功能的汇编程序。

2.898

宏生成器 macrogenerator

宏生成程序 macro generating program

一个例程，常常是汇编程序或编译程序的一部分，在源程序中，它用宏指令表示的预定义指令序列，替换每条宏指令。

2.899

宏指令 macroinstruction

源语言中的一条指令，它将用同一源语言书写的预先定义的指令序列在汇编或编译期间代替。宏指令也可以为将要代替它的指令中的参数指定其值。

参见： 宏 macro（2.893）和宏生成器 macrogenerator（2.898）。

2.900

宏处理程序 macroprocessor

在某些汇编程序和编辑程序中的例程或例程集，它用以支持宏的定义和使用。

2.901

宏编程 macroprogramming

编写用宏和宏指令构成的计算机程序。

2.902

主程序 main program

由计算机的操作系统调用的软件部件，它通常调用其他软件部件。

参见： 例程 routine（2.1402）和子程序 subprogram（2.1618）。

2.903

可维护性 maintainability

a) 软件系统或部件能修改以排除故障、改进性能或其他属性或适应变更了的环境的容易程度。

参见： 可扩展性 extendability（2.594）和灵活性 flexibility（2.634）。

b) 硬件系统或部件能从执行它所要求的功能的状态得到或重新存放回去的容易程度。

2.904

维护方 maintainer

执行维护活动的一个机构。

2.905

维护 maintenance

软件维护 software maintenance

a) 在交付以后，修改软件系统或部件以排除故障、改进性能或其他属性或适应变更了的环境的过程。

参见： 适应性维护 adaptive maintenance（2.43）、纠正性维护 corrective maintenance（2.354）和完善性维护 perfective maintenance（2.1130）。

b) 得到硬件系统或部件或重存至能执行所要求的功能的状态的过程。

参见： 预防性维护 preventive maintenance（2 1168）。

2.906

维护手册 maintenance manual

见： 支持手册 support manual（2.1631）。

2.907

维护阶段 maintenance phase

见： 运行和维护阶段 operation and maintenance phase（2.1057）。

2.908

维护计划 maintenance plan

维护软件产品所使用的说明管理方法和技术途径的文档。典型的维护计划内容包括：工具、资源、设施及日程。

2.909

人机界面、人机接口 man-machine interface(MMI)

见： 用户界面 user interface（2.1812）。

2.910

受到管理和控制 managed and controlled

标识和定义软件工作产品的过程，这些软件工作产品不是基线的一部分，因此不一定要置于配置管理之下，但为使项目有纪律地开展又必须受控。“受到管理和控制”是指在给定时间（过去或现在）使用的工作产品的版本均是可查知（即，版本控制），而且以受控的方式进行更改（即，更改控制）。

2.911

经理 manager

一种角色，对在其职责范围内执行任务或活动的人员提供技术上和行政上的管理和控制。经理的传统职能包括对职责范围内的工作进行策划、提供资源、进行组织，并进行指导和控制。

2.912

制造 manufacture

在软件工程中，复制软件至磁盘、芯片或其他设备以发布至客户或用户的过程。

2.913

制造阶段 manufactureing phase

软件生存期的一个时间阶段，期间复制适应操作环境的规定集的软件的基本版本并发布至客户。

2.914

映像程序 map program

一种软件工具，多半是编译程序或汇编程序的组成部分，它生成装入映像。

2.915

屏蔽 mask

用来与一未知的数据项逻辑地组合的位或字符的模式以得到或抑制数据项的某些部分。例如，位串”00000011”当与一八位数据项逻辑与时，得到数据项的最低两位而在所有其他位的位置为 0 的结果。

参见： 中断屏蔽 interrupt mask（2.813）。

2.916

主库 master library

包括软件和文档从它的工作副本复制的主副本的软件库，为了发布和使用。

相对： 产品库 product library（2.1217）、软件开发库 software development library（2.1487）、软件储存库 software repository（2.1529）和系统库 system library（2.1657）。

2.917

主状态 master state

见： 监督状态 supervisor state（2.1627）。

2.918

成熟度等级 maturity level

在朝着实现成熟软件过程的目标前进过程中，组织的不断进化但又相对稳定的软件过程的一种妥善定义的状态。能力成熟度模型中的五个成熟度等级是：

a) 初始级：软件过程处于无序甚至混乱的状态。几乎没有什么经过定义的过程，成功完全依赖于有经验的软件经理及高水平的软件队伍。

b) 可重复级：已建立基本的项目管理过程，对成本、进度和功能特性进行跟踪。已建立必要的过程纪律，使类似项目能重复以前的成功。

c) 已定义级：软件过程在管理活动和工程活动两方面均已文档化、标准化，并集成为组织的标准软件过程。每个项目均采用组织的标准软件过程的经剪裁并得到批准的子集。

d) 定量管理级：已采集关于软件产品和过程质量的详细测量数据。无论软件过程还是产品均得到定量的了解和控制。

e) 优化级：利用过程实施中的定量反馈信息和新思想、新技术的先导性试验中的定量反馈信息，使组织能持续进行过程改进。

2.919

成熟度提问单 maturity questionnaire

一组关于软件过程的提问，抽样检查能力成熟度模型的每个关键过程域中关键实践的实施情况。它是评价组织的软件过程能力或项目的软件过程性能的一种手段。

2.920

平均失效时间间隔 mean time between failures(MTBF)

在系统或部件中，在连续的失效之间预料的或观察到的时间。

参见： 开机时间 up time（1798）。

2.921

平均修复时间 mean time to repair(MTTR)

修复系统或部件使它恢复正常工作所要求的预料的或观察到的时间。

参见： 关闭时间 down time（2.503）。

2.922

测量【动】 measure(verb)

执行一次测量(活动)。

2.923

测度【名】 measure

通过执行一次测量赋予实体属性的数或类别。

2.924

测量 measurement

使用一种度量，把标度值（可以是数或类别）赋予实体的某个属性。

注：使用类别时，测量可以是定性的。如软件产品的一些重要属性，例如源程序语言（ADA，C，COBOL 等）就是定性的类别。

2.925

测量标准 measurement standard

描述评价过程或产品的特征的标准。

2.926

内存容量 memory capacity

在给定的计算机内存中能保持的项的最大数，它通常用字或字节测量。

参见： 通道容量 channel capacity（2.198）和存储容量 storage capacity（2.1593）。

2.927

内存紧缩 memory compaction

废物收集 garbage collection

a) 一种存储分配技术，其中，所分配的存储区域的内容，全移至存储空间的开始而余下的存储块组合为一个大块。

b) 所有未分配的存储器的连续的块组合成单个块的存储分配技术。

2.928

内存转储 memory dump

计算机的内部存储器的部分或全部的内容的显示，通常以二进制、八进制或十六进制形式。

参见： 变更转储 change dump（2.196）、动态转储 dynamic dump（2.519）、事后转储 postmortem dump（2.1153）、有选择性转储 selective dump（2.1425）、快照转储 snapshot dump（2.1466）和静态转储 static dump（2.1582）。

2.929

内存映像 memory map

显示程序和数据存放在计算机的内存的何处的图。

2.930

选单旁路 menu bypass

在选单驱动（menu-driven）系统中，允许高级用户以命令驱动方式执行各种功能而不从选单中选择选项的特征。

2.931

选单驱动的 menu-driven

用于说明系统或操作模式是用户通过选单指挥系统的。

参见： 选单旁路 menu bypass（2.930）。

相对： 命令驱动的 command-driven（2.233）。

2.932

消息 message

用于输送信息的字符的一种有序序列。(以其活动会随之发现，而对信息从一个实例到另一个实例的传送的一种规格说明。消息可规定信号的激活起或对操作的调用。)

2.933

元编译程序 metacompiler

见： 编译程序的生成程序 compiler generator（2.257）。

2.934

元语言 metalanguage

用于规定语言的某些方面或所有方面的语言。例如，Backus-Naur 形式。

参见： 分层的语言 stratufied language（2.1598）和非分层的语言 unstratified language（2.1794）。

2.935

元类 metaclass

一种其实例为类的类。其典型用途是构造元模型。

2.936

元元模型 meta-metamodel

一种用于定义元模型的模型。元元模型与元模型之间的关系，类同于元模型与模型之间的关系。

2.937

元模型 metamodel

一种定义的模型，它用于表达某一模型的语言。

2.938

元对象 metaobject

一个一般性术语，做为一种元建模语言中所有元实体（例如元类型、元类、元属性和元关联）的总称。

2.939

方法 method

对某一操作的实现。方法规定了与某一操作关联的算法或过程。

2.940

方法标准 method standard

描述用在产品工程中或执行服务的顺序进程或过程的特征的标准

2.941

方法论 methodology

规定产品开发的集成综合工程途径的方法、规程和标准的集合。

2.942

计量，度量[体制] metric

a) 系统、部件或进程占有给定的属性的程度的数量的测量。

参见： 质量度量 quality metric（2.1300）。

b) 定义的测量方法和测量标度。

注 1： 度量[体制]可以是内部的或外部的，可以是直接的或间接的。

注 2： 度量[体制]包括把定性数据进行分类的方法。

2.943

百万次浮点运算每秒 MFLOPS

millions of fLoating point operations per second 的缩略语。计算机处理速度的一种测量。

参见： 百万条指令每秒 MIPS（2.958）。

2.944

微体系结构 microarchitecture

微字定义、数据流、定时限制和优先限制，所有这些确定了给定的微程序计算机的特色。

2.945

微码 microcode

微指令、组成部分、全部或一组微程序的集合。

2.946

微码汇编程序 microcode assembler

把微程序从符号形式翻译为二进制形式的计算机程序。

2.947

微指令 microinstruction

规定执行一机器语言指令需要的一个或多个基本操作的微程序中的指令。类型包括：对角微指令（diagonal microinstruction）、横向微指令（horizontal microinstruction）、纵向微指令（vertical microinstruction）。

参见：微码 microcode(2.945)、微操作 microoperation(2.948)和微程序 microprogram(2.949)。

2.948

微操作 microoperation

在微程序编程中，执行一机器语言指令所需要的基本操作之一。

参见： 微指令 microinstruction（2.947）。

2.949

微程序 microprogram

规定执行一机器语言指令所需要的基本操作，称为微指令的指令序列。通常它被保存在专用存储器中，并且是由计算机指令寄存器中的计算机指令来启动其执行，微程序常常用于代替硬接线逻辑。

参见： 固件 firmware（2.630）。

2.950

可编微程序的计算机 microprogrammable computer

微程序能由用户创建或替换的微程序计算机。

2.951

微程序的计算机 microprogrammed computer

机器语言指令由微程序而不是由硬件连线逻辑实现的计算机。

注： 微程序计算机，可以是也可以不是微型计算机、它们的概念不相关，尽管术语是类似的。

参见： 微体系结构 microarchitecture（2.944）和可编微程序的计算机 microprogrammable computer（2.950）。

2.952

微程序设计 microprogramming

设计和实现计算机的控制逻辑的过程,它是通过标识为实现每条机器语言指令所需的基本操作并把这些操作表示为在称为控制存储的特殊存储器中的指令序列来实现的。这种方法是实现每条机器语言指令的所需的硬件连线控制信号的一种替换方法。技术包括：位操作、压缩、剩余控制、单级编码和两级编码。

参见： 微码 microcode（2.945）、微指令 microinstruction（2.947）和微程序 microprogram（2.949）。

2.953

微字 microword

在微程序计算机的控制存储中一可寻址的元素。

2.954

中间件 middleware

一种类型的软件模块，它处在系统软件和应用软件之间，依赖系统软件的支持，又为应用软件提供支持，以方便应用软件的开发。

2.955

迁移 migration

软件或数据从一个场所（目录或存储单元）至另一场所的传送。

2.956

里程碑 milestone

项目有关人员或管理人员负责的在预定时间将发生的事件，它用来标志工作进度。例如，正式的复审、规格说明的颁布、产品的交付。

2.957

最小延迟编程 minimum delay programming

一种编程技术，它选择计算机指令和数据的存储单元，以使访问时间最小。

2.958

百万指令每秒 MIPS

million instruction per second 的缩略语。计算机处理速度的一种测量。

参见： 千运算每秒 （2.835）和百万浮点运算每秒 MFLOPS（2.943）。

2.959

错误 mistake

产生不正确结果的人为动作。注：容错系统，在人为动作（mistake）、它的显示（硬件和软件故障）、故障的结果（failure）和结果是不正确的（error）总数之间加以区分。

2.960

混合方式 mixed mode

混合类型 mixed type

包含两个或多个不同数据类型的表达式。例如，Y := X+N，其中，Y 和 X 是浮点变量，而 N 是整数变量。

2.961

混合类型 mixed type

见： 混合方式 mixed mode（2.960）。

2.962

人机界面 MMI

man-machine interface 的缩略语。

见： 用户界面 user interface（2.1812）。

2.963

助忆符[号] mnemonic symbol

为便于人们记忆而选用的一种符号。例："multiply"的缩略语是"mul"。

2.964

模型 model

现实世界中过程、设备或概念的一种表示。

参见： 分析模型 analytical model（2.63）、可用性模型 availability model（2.116）、排错模型 debugging model（2.414）、出错模型 error model（2.565）、可靠性模型 reliability model（2.1339）、仿真 simulation（2.1453）和统计测试模型 statistical test model（2.1584）。

2.965

模型方面 model aspect

一种侧重于元模型特别质量的建模维。例如，结构模型方面侧重于元模型的结构质量。

2.966

模型驱动体系结构 model driven architecture (MDA)

它是解决基于不同中间件系统的集成问题。它将不同的系统都结构化成平台独立模型（PIM-platform independent model）和平台特定模型（PSM-platform specific model），并针对不同实现技术与平台制订多个映射规则，然后通过这些映射规则及辅助工具将 PIM 转换成 PSM，再将 PSM 不断求精直至形成最后代码。

2.967

模型详化 model elaboration

从已公布的模型生成某一存储库类型的过程。包括接口和实现按如下方式的生成：使各存储库能基于并符合于已详化的模型而加以实例化和装入。

2.968

模型元素 model element

从被建模的系统抽象出来的元素。

相对： 视图元素 view element（2.1834）。

[MOF]在 MOF 规格说明中，将模型元素当作是对象。

2.969

模型库 model library

含有供其他包重用的模型元素的一种衍型包。模型库类似于某些程序设计语言中的类库。

2.970

建模时[的] modeling-time

用于说明在软件开发过程的建模阶段所出现的某一事物。包括分析阶段与设计阶段。用法注解：在讨论对象系统时，区分建模与运行时间的有关问题，常常是很重要的。

参见： 分析阶段 analysis phase（2.62）和设计阶段 design phase （2.450）。

相对： 运行时间 run time（2.1407）。

2.971

修改 modification

a) 对软件进行的更改。

b) 更改软件的过程。

2.972

模块的 modular

由分离的各部分构成的。

参见： 模块分解 modular decomposition（2.973）和模块化程序设计 modular programming（2.974）。

2.973

模块分解 modular decomposition

模块化 modularization

把系统分成若干模块（模块编程元素）以便于设计和开发的过程。

参见： 内聚度 cohesion（2.228）、耦合 coupling（2.358 条）、去模块化 demodularization（2.432）、因素分解 factoring（2.598）、功能分解 functional decomposition（2.664）、层次结构分解 hierarchical decomposition（2.696）和封装 packaging（2.1091）。

2.974

模块化程序设计 modular programming

把软件作为模块的集合进行开发的软件开发技术。

参见： 以数据结构为中心的设计 data structure-centered design（2.405）、输入-处理-输出 input-processoutput（2.762）、模块分解 modular decomposition（2.973）、面向对象的设计 object-oriented（2.1037）、快速原型 rapid prototyping（2.1307）、逐步细化 stepwise refinement（2.1587）、结构冲突 structure clash（2.1606）、结构设计 structured design（2.1607）、事务分析 transaction analysis（2.1760）和转换分析 transform analysis（2.1764）。

2.975

模块性 modularity

计算机程序由离散部件组成的特性，以便使一个模块的变更对其他模块的影响最小。

参见： 内聚度 cohesion（2.228）和耦合 coupling（2.358）。

2.976

模块化 modularization

见： 模块分解 modular decomposition（2.973）。

2.977

模块 module

a) 离散的程序单位。且对于编译、与其他单位相结合和装入来说是可标识的。例如，汇编程序、编译程序、连接编辑程序或执行的例行程序的输入或输出。

b) 程序中一个能逻辑地分开的部分。注：术语“模块（module）”、“部件（component）”和“单位（unit）”常常是可互换的或以不同方法定义为另一个的子元素，取决于上下文。这些术语之间的关系尚未标准化。

2.978

模块强度 module strength

见： 内聚度 cohesion（2.228）。

2.979

模块测试 module testing

见： 部件测试 component testing（2.264）。

2.980

一元选择构造 monadic selective construct

一种 if-then-else 结构，在此种结构中，处理只由分支的一个输出规定，另一个输出导致跳过此处理。

相对： 二元选择构造 dyadic selective construct（2.510）。

2.981

监控程序，监控器 monitor

执行监控程序 execution monitor

一软件工具或硬件设备，它与系统或部件并发工作，监督、记录、分析或验证系统或部件的操作。

参见： 硬件监控器 hardware monitor（2.692）、软件监控程序 software monitor（2.1509）。

2.982

监控 monitoring

由需方或第三方对供方活动状况及其成果的检查。

2.983

移动 move

a) 从源读数据，变更源单元的内容并可以与源不同的物理格式写同一数据至别处。例如，从一个文件传送数据至另一个文件。

相对： 复制 copy（2.351）。

b) 有时，作为复制 copy（2.351）的同义词。

参见： 读取 fetch（2.622）、装入 load（2.863）和存储 store（2.1595）。

2.984

平均失效间隔时间 MTBF

mean time between failures 的缩略语。

2.985

平均修复时间 MTTR

mean time to repair 的缩略语。

2.986

多地址指令 multiaddress instruction

多地址指令 multiple-address instruction

包含多于一个地址字段的指令。

相对： 单地址指令 one-address instruction（2.1047）。

2.987

多级地址 multilevel address

见： 间接地址 indirect address（2.744）。

2.988

多级安全 multilevel security

一种操作方式，当至少有某些用户对系统中包括的全部数据既不清楚也不需要知道时，它允许处于各种安全级上的数据并行地在计算机系统中存储和处理。

2.989

多级存储 multilevel storage

见： 虚拟存储 virtual storage（2.1840）。

2.990

多地址指令 multiple-address instruction

见： 多地址指令 multiaddress instruction（2.986）。

2.991

多重分类 multiple classification

其中对象直接属于的类目可多于一个的泛化的一种语义变化。

参见：静态分类 static classification（2.1581）和动态分类 dynamic classification（2.518）。

2.992

多排它选择构造 multiple exclusive selective construct

见： （分）情况语句 case（2.186）。

2.993

多包含选择构造 multiple inclusive selective construct

case 结构的一种特殊情况，在此情况下，控制表达式的两个或多个值导致相同的处理。例如，值 1 和 2 引起一分支，3 和 4 引起另一分支，等等。

2.994

多[重]继承 multiple inheritance

泛化的语意变种，其中类型可具有多于一个超类型。

相对： 单一继承 single inheritance（2.1458）。

2.995

多重性势域 multiplicity

对某一集合可取的势的范围的一种规格说明。多重性可对关联内的角色、组合中的各部分、重复次数或其他目的来给出规格说明。在本质上，多重性是非负整数的一个（可能无限）子集。

相对： 势 cardinality（2.184）。

2.996

多重处理 multiprocessing

一种操作模式，在其中，两个或多个进程由分别的处理单元并行地执行。它们访问共同的主存储器（通常）。

参见： 多任务 multitasking（2.998）和分时 time sharing（2.1629）。

相对： 多道程序设计 multiprogramming（2.997）。

2.997

多道程序设计 multiprogramming

一种操作方式，它可以使单处理机交替地执行两个或多个计算机程序。

参见： 多任务 multitasking（2.998）和分时 time sharing（2.1740）。

相对： 多重处理 multiprocessing（2.954）。

2.998

多任务 multitasking

一种操作方式。其中，两个或多个任务以交替方式执行。

参见： 多重处理 multiprocessing（2.996）、多道程序设计 multiprogramming（2.997）和分时 time sharing（2.1740）。

2.999

多值的 multi-valued

带有经定义的多重性的一种模型元素，其多重性类型即上层属性设置为大于 1 的数。在任何时刻，术语多值都与属性、参数等所取的值的数目无关。

相对： 单值 single value（2.1463）。

2.1000

变异 mutation

见： 程序变异 program mutation（2.1239）。

2.1001

变异测试 mutation testing

一种测试技术，其中的两个或多个程序变异用相同的测试用例执行，以评价测试用例检测变异中的差别的能力。

2.1002

N地址指令 n-address instruction

由n个地址字段组成的指令，其中，n可以是任一非负整数。

参见： 一地址指令 one-address instruction（2.1047）、二地址指令 two-address instruction（2.1777）、等等。

相对： n加1地址指令 n-plus-one address instruction（2.1008）。

2.1003

N-进制的，N元的 N - ary

a) 由n个不同可能的值或状态的挑选、选取或条件所表征。

b) 具有基数n的固定基数数制系统。

2.1004

N元关联 n-ary association

三个或更多类之间的一种关联。此关联中的任何实例都是取自有关各类的一个n元组值。

相对： 二元关联 binary association（2.137）。

2.1005

名[称] name

一种用于标识某一模型元素的字符串。

2.1006

命名空间 namespace

模型的一部分，可在其中定义并使用名称。命名空间中的每一名称意义唯一。

参见： 名称 name（2.1005）。

2.1007

N级地址 n-level address

一种间接地址，它规定n个存储单元的链的第一个，它的前n-1包含链中下一个单元的地址，最后，包含所希望的操作数。例如，两级地址。

相对： 直接地址 direct address（2.481）和立即数据 immediate data（2.721）。

2.1008

N加1地址指令 n-plus-one address instruction

包含n+1个地址字段的指令，其中最后一个包含下一个要执行的指令的地址。

参见： 1加1地址指令 one-plus-one address instruction（2.1050）、 2加1地址指令 two-plus-one address instruction（2.1780）等等。

相对： n地址指令 n-address instruction（2.1002）。

2.1009

超微码 nanocode

超微指令（nanoinstruction）的集合。

2.1010

超微指令　nanoinstruction

在微程序的两级实现中，规定实现微指令所需要的一个或多个基本操作的指令。

2.1011

超微存储　nanostore

在微程序的两级实现中，存储超微指令的二级控制存储器。

2.1012

Nassi-Shneiderman 图　Nassi-Shneiderman chart

见： 盒图 box diagram（2.156）。

2.1013

自然语言　natural language

一种语言，其规则根据当前的习惯用法而不是在使用前预它建立。例如英语、汉语、法语以及斯瓦希利语。

相对： 形式语言 formal language（2.644）。

2.1014

嵌套　nest

a) 把某一类的一个或多个结构合并到同一类结构中去。例如，把一个循环〈被嵌套的循环〉嵌套到另一个循环（嵌套的循环）中去、把一个子例程（被嵌套的子例程）嵌套到另一个子例程（嵌套的子例程）中去。

b) 把子例行程序或数据放在处于另一不同层次级别上的另一个子例行程序或数据中，使得该子例行程序可作为递归子例行程序被执行，或该数据能被递归地存取。

2.1015

网[络]　network

a) 一组互连或相关的结点。

b) 涉及约束性或面向问题的修饰词、资料、文件以及人力资源的组合，通过设计把这些组合起来以实现某些目标。例如：社会科学网络、科学信息网络。

2.1016

网[络]图　network chart

一种有向图，它在项目控制中用以描述和调度各事件、活动及其联系。

2.1017

网络规划　network planning

使用网络图来规划、调度和控制某一项目的一种技术。

2.1018

空操作　no-op

no-operation 的缩略语。

2.1019

空操作　no-operation

空操作　donothing operation

一种计算机操作，它的执行，除了使指令计数器前进至下一条指令外，没有任何其他影响。用于在程序中保留空间，若重复执行，为等待一给定的事件。常常缩略语为 no-op。

2.1020

结点，节点　node

a) 网络或图中任何分支的端点，或属于两个或多个分支的公共结点。

b) 在树形结构中，下属数据项由之发源的一点。

c) 在网络中，一个或多个功能部件在此互连传输线的点。

d) 图中的点、圆或任何其他几何图形，用以表示状态、事件或其他感兴趣项。

参见： 图 graph（2.684）b)。

2.1021

命名原则标准 nomenclature standard

描述系统、一组名字、指示或符号的特征的标准。

2.1022

非交付项 non-deliverable item

不需要按合同交付的、但在软件的开发中可能用到的硬件和软件。

2.1023

非技术性要求 non-technical requirements

影响并规定软件项目管理活动的协议、条件和（或）合同条款。

2.1024

非破坏性读 nondestructive read

一种读操作，它不删除在所访问单元中的数据。

相对： 破坏性读 destructive read（2.461）。

2.1025

非过程语言 nonprocedural language

一种语言，其中，没有说明为完成用户状态，在给定的序列中计算机必须执行的特定指令。

参见： 声明语言 declarative language（2.417）和基于规则的语言 rule-based language（2.1403）。

相对： 过程语言 procedural language（2.1180 条）。

2.1026

修订通知 NOR

在配置管理中，notice of revision 的缩略语。

注：为一种逻辑操作。

2.1027

记法标准 notation standard

描述在声明中正式接口的特征的标准。

2.1028

修订通知 notice of revision(NOR)

在配置管理中用的一种格式，用以对图、表进行修订提出建议，以及在批准后通知用户图或表已经或将相应地修订。

参见： 配置控制 configuration control（2.307）、工程变更 engineering change（2.546）和规格说明变更通知 specification change notice（2.1549）。

2.1029

核心 nucleus

见： 内核 kernel（2.831）a)。

2.1030

目标，对象 object

a) 关于汇编或编译过程的输出。

参见： 目标代码 object code（2.1031）、目标模块 object module（2.1036）和目标程序 object program（2.1039）。

b) 一程序常数或变量。

c) 数据和对该数据进行操作的服务的封装体。

参见： 面向对象的设计 object-oriented design（2.1037）。

2.1031

目标代码 object code

由汇编程序或编译程序输出的格式表示的计算机指令和数据定义。目标程序由目标代码构成。

相对： 源代码 source code（2.1541）。

2.1032

对象图 object diagram

一种由在某一时刻的各对象及其关系所组成的图。对象图为可视类图或协作图的特例。

参见： 类图 class diagram（2.212）和协作图 collaboration diagram（2.231）。

2.1033

对象流状态 object flow state

在某一活动图中的一种状态，它表示一个对象从一个状态中的动作输出到另一状态中的动作输入的传递的状态。

2.1034

目标语言 object language

见： 目标语言 target language（2.1676）。

2.1035

对象生存线 object lifeline

在时序图中，一条表示对象经历时期的线。

参见： 时序图 seqence diagram（2.1437）。

2.1036

目标模块 object module

是汇编程序或编译程序的输出的计算机程序或子程序。

参见： 装入模块 load module（2.866）和目标程序 object program（2.1039）。

2.1037

面向对象的设计 object-oriented design

系统或部件用对象和它们之间连接的术语表示的软件开发技术。

参见： 以数据结构为中心的设计 data structure-centered design（2.405）、输入-处理-输出 input-process-output（2.762）、模块分解 modular decomposition（2.973）、快速原型 rapid prototyping（2.1307）、逐步细化 stepwise refinement（2.1587）、结构设计 structured design（2.1607）、事务分析 transaction analysis（2.1760）和转换分析 transform analysis（2.1764）。

2.1038

面向对象的语言 object-oriented language

一种程序设计语言，它允许用户用对象和这些对象之间的消息的术语表示程序。例子包括 Smalltalk 和 LOGO。

2.1039

目标程序 object program

目标程序 target program

已完成编译或汇编已准备好装入到计算机中的程序。

参见： 目标模块 object module（2.1036）。

相对： 源程序 source program（2.1544）。

2.1040

客观证据 objective evidence

基于可验证的观测、测量或测试，关于过程元素的存在和实施或者项或服务的特性的定量或定性的事实陈述、记录或信息

2.1041

职务标准 occupational title standard

描述通常的职业和工作的特征的标准。

2.1042

脱机的，离线的 offline

用于说明未在计算机的中央处理单元直接控制下的设备或进程。

相对： 在线的 online（2.1045）b)。

2.1043

偏移量 offset

a) 在装入的初始地址与计算机程序的汇编的初始地址之间的差。与重定位因数 relocation factor（2.1349）同义。

b) 必须加至相对地址以确定要访问的存储单元的地址的数。此数可能是在 a 中定义的差或在程序中定义的另一个数。

参见： 基地址 base address（2.124）、索引的（变址的）地址 indexed address（2.740）、相对地址 relative address（2.1331）和自相关地址 self-relative address（2.1429）。

2.1044

现货产品 off-the-shelf product

由供方、需方或第三方提供的、已经开发出来的、可得到、可使用的、现成的或需要加以修改的产品。

2.1045

联机的，在线的 online

a) 关于输入数据直接从初始点进入计算机或输出数据直接发送至所用的点的系统或部件。例如，航线订票系统。

参见： 会话的 conversational（2.348）、交互的 interactive（2.792）和实时的 real time（2.1313）。

相对：批处理的 batch（2.128）。

b) 在计算机中央处理单元直接控制下的设备或进程。

相对： 离线的 offline（2.1042）。

2.1046

联机编译程序 online compiler

见： 增量编译程序 incremental compiler（2.737）。

2.1047

一地址指令 one-address instruction

单地址指令 single-address instruction

单操作数指令 single-operand instruction

包含一个地址字段的计算机指令。例如，装入 A 单元内容的指令。

相对： 多地址指令 multiaddress instruction（2.986）、二地址指令 two-address instruction（2.1777）、三地址指令 three-address instruction(2.1733)、四地址指令 four-address instruction（2.652）和零地址指令 zero-address instruction（2.1859）。

2.1048

向前一个寻址 one-ahead addressing

一种隐含寻址方法，其中，计算机指令的操作数在上一条执行的指令所用操作数的存储单元的下一个存储单元中。

相对： 重复寻址 repetitive addressing（2.1354）。

2.1049

一级地址 one-level address

见： 直接地址 direct address（2.481）。

2.1050

一加一地址指令 one-plus-one address instruction

包含两个地址字段的计算机指令，其第二个包含下一条要执行的指令的地址。例如，一条指令加载单元 A 的内容，然后执行在单元 B 的指令。

相对： 二加一地址指令 two-plus-one address instruction（2.1780）、三加一地址指令 three-plus-one address instruction（2.1734）和四加一地址指令 four-plus-one address instruction（2.653）。

2.1051

操作码 opcode

见： 操作码 operation code（2.1058）。

2.1052

开式子例程 open subroutine

直接插入子例程 direct insert subroutine

在调用时复制至计算机程序的调用处的子例程。

参见： 内联码 inline code（2.758）和宏 macro（2.893）。

相对： 闭式子例程 closed subroutine（2.216）。

2.1053

开放系统 open system

一种按标准建立起来的计算机系统。开放系统的主要特征是：

兼容性 compatibility（2.249）；

可移植性 portability（2.1149）；

互操作性 interoperability（2.807）和

可扩缩性 scalability（2.1411）。

开放系统可以使用户摆脱特定厂商的控制，并有利于提高软件厂商的效益。

2.1054

操作数 operand

要在其上执行操作的变量、常数或函数。例如，在表达式 A=B+3 中，B 和 3 是操作数。

2.1055

操作系统 operating system

控制各种程序的执行并可提供资源分配、调度、输入输出控制以及数据管理等服务的软件。虽然绝大多数操作系统是软件，部分用固件或硬件实现也是可能的。

2.1056

操作，运行，运算 operation

a) 在计算机算术操作中，由一个操作符在一个或多个操作数上规定的动作。例如，在表达式 A=B+3 中，加 B 至 3 以得到 A 的过程。

b) 在程序设计中，定义由计算机系统执行的动作、例如，相加、比较、分支等。注：与算术操作不同，操作可能不包含操作符或操作数、例如暂停（halt）操作。

c) 在目标环境中执行目标函数的计算机系统的运行过程。

d) 在面向对象中，操作是对对象行为的指正。

2.1057

运行和维护阶段 operation and maintenance phase

软件生存周期中的一个阶段，在此阶段软件产品在规定的运行环境中进行使用、监控，以得到满意的性能，必要时要进行修改，以改正问题或对变化了的需求作出响应。

2.1058

操作码 operation code

操作码 opcode

规定计算机操作的字符或字符组。例如，代码 BNZ 指定“若非零分支”。

2.1059

操作异常 operation exception

当程序遇到无效操作码引起的异常。

参见： 寻址异常 addressing exception（2.50）、数据异常 data exception（2.395）、溢出异常 overflow exception（2.1082）、保护异常 protect exception（2.1274）和下溢异常 underflow exception（2.1786）。

2.1060

操作字段 operation field

功能字段 function field

操作部分 operation part

规定要执行的操作的计算机指令字段。

相对： 地址字段 address field（2.45）。

2.1061

操作部分 operation part

见： 操作字段 opeeration field（2.1060）。

2.1062

可运行的 operational

用于说明：

a) 在它的目标系统中准备好使用的系统或部件；

b) 在目标环境中安装的系统或部件；

c) 系统或部件准备使用的环境。

2.1063

运行可靠性 operational reliability

在实际使用环境中，系统或软件子系统的可靠性。运行可靠性可能与规定环境或测试环境中的可靠性有很大的不同。

2.1064

运行软件 operational software

预定在交付给顾客，并部署在预定环境下的系统中使用和运行的软件。

2.1065

运行测试 operational testing

在它的目标环境中，评价系统或部件的测试行为。

参见： 验收测试 acceptance testing（2.19）和合格性测试 qualification testing（2.1291）。

相对： 开发测试 development testing（2.468）。

2.1066

运算符，操作员 operator

a) 在操作中表示要执行的动作的算术或逻辑符号。例如，在表达式 A=B+3 中，+是一操作符，表示相加。

b) 操作计算机系统的人。

2.1067

操作符字段 operator field

见： 操作字段 operation field（2.1060）。

2.1068

操作员手册 operator manual

提供启动和操作系统或部件必需的信息的文档。典型地描述准备、操作、监控和恢复过程。

注： 当在操作计算机系统（安装磁带等等）的人与为了一定的目的使用计算机系统的人之间有区分时，操作员手册和用户手册有区别。

参见： 诊断手册 diagnostic manual（2.476）、安装手册 installation manual（2.766）、程序员手册 programmer manual（2.1250）、支持手册 support manual（2.1631）和用户手册 user manual（2.1813）。

2.1069

时机研究 opportunity study

一种研究，它考查某一问题，并确定其是否需在考虑中的某一时期来解决。

2.1070

次序抵触（冲突） order clash

在软件设计中，结构冲突的一种类型，在这种相撞中，程序必须处理两种或多种已经以不同次序排序的数据集。

参见： 以数据结构为中心的设计 data structure-centered design（2.405）。

2.1071

组织 organization

公司或其他实体中的单位，在这个单位中许多项目作为一个整体加以管理。一个组织内的所有项目有一个共同的顶层经理，并有共同的方针。

2.1072

组织过程 organizational process

为构成自始至终的一个完整过程，由机构指定并以项目为例说明的一套软件工程和管理过程。

2.1073

组织单位 organizational unit

将经受评估的组织部分。

注 1：一个组织单位使用一个或一组且具有协调的环境的过程并在一组协调经营目标内运营。

注 2：组织单位是大型组织的一个典型部分、尽管在一个小的组织中，组织单位可能是指整个组织。

例如：一个组织单位可以是：

— 一个特定的项目或一组（相关的）项目、
— 组织内的一个关注某个特定的生存周期阶段（或几个阶段）单位诸如采购、开发、维护或支持、
— 组织内部负责某个特定产品或产品类的所有方面的某个部分。

2.1074

组织的测量大纲 organization's measurement program

阐述组织的测量需求的一组相关元素，它包括全组织范围的测量的定义、收集和分析组织的测量数据的方法和实践、以及组织的测量目标。

2.1075

组织的软件过程资产 organization's software process assets

由组织维护的，供项目在制定、剪裁、维护和实施其软件过程时使用的一组实体的集合。组织的软件过程资产一般包括：

a) 组织的标准软件过程；
b) 被批准使用的软件生存周期的描述；
c) 剪裁组织的标准软件过程的指南和准则；
d) 组织的软件过程数据库；
e) 软件过程有关文档库；

f) 组织认为在进行过程定义和维护活动方面有用的任何实体。

2.1076

组织的软件过程数据库 organization's software process database

为收集有关软件及其生成的软件过程工作产品的数据(特别是与组织的标准软件过程有关的数据),并使其可供使用而建立的数据库。该数据库包含(或者引用)实际的测量数据和为理解测量数据及评估其合理性和适用性所需要的相关信息。过程和工作产品的数据的例子包括:软件规模、工作量和成本的估计数据;软件规模、工作量和成本的实际数据;生产率数据;同行评审的覆盖率和效率的数据;以及在软件代码中所发现的缺陷的数目及严重性。

2.1077

组织的标准软件过程 organization's standard software process (OSSP)

一组基本过程的操作性的定义,这些基本过程用来指导在组织的所有项目中建立一个共同的软件过程。它描述各个软件项目定义的软件过程预期要包括的基本软件过程元素,以及这些软件过程元素间的关系(例如,排序和接口)。

2.1078

定向培训 orientation

对一个主题的一个概论或引论性培训,以便对负责实施该主题的人员进行一般的管理和交往。

2.1079

起点 origin

赋给计算机程序的在主内存中初始存储单元的地址。

参见: 汇编的起点 assembled origin (2.81)和装入的起点 loaded origin (2.867)。

相对: 启动地址 starting address (2.1576)。

2.1080

输出 output

a) 传送至外部目标的数据。

b) 在把数据传送至外部目标中包含的设备、进程或通道。

c) 为传送数据至外部目标。

d) 接近输出数据。

相对: 输入 input (2.759)。

2.1081

输出断言 output assertion

一个逻辑表达式,它规定为了保证程序正确,程序输出必须满足的一个或多个条件。

参见:2.747 条 归纳断言方法(inductive assertion method)。

相对: 输入断言 input assertion (2.760)和循环断言 loop assertion (2.880)。

2.1082

溢出异常 overflow exception

当算术运算的结果超出了指定的存储单元能接受它的大小时发生的异常。

参见: 寻址异常 addressing exception (2.50)、数据异常 data exception (2.395)、操作异常 operation exception (2.1059)、保护异常 protection exception (2.1274)和下溢异常 underflow exception (2.1786)。

2.1083

开销操作 overhead operation

见: 内务操作 housekeeping operation (2.711)。

2.1084

开销时间 overhead time

计算机系统执行并不直接贡献给任一用户任务的进行的任务所花费的时间的总数。例如，为了编排的目的安排计算机资源所花费的时间。

2.1085

覆盖 overlay

a) 一种存储分配技术，在此技术中，计算机程序段从辅助存储器装入至主内存，当需要时，覆写当前不在使用的其他段。

b) 一个计算机程序段，它在辅助存储器中维护，当需要时装入主内存，覆写当前未在使用的其他段。

c) 用覆写程序的其他段的方式从辅助存储器装入计算机程序段至主内存。

2.1086

覆盖监督程序 overlay supervisor

控制覆盖的次序和位置的例程。

2.1087

重载 overload

赋予一操作符、标识符或文字多于一种含义，取决于在程序执行期间任一给定时间与其相关的数据类型。

2.1088

紧缩 pack

用数据和媒体已知的特征以允许数据恢复的方式把数据用压缩的方法存放在媒体中。

相对：解包 unpack（2.1793）。

2.1089

包 package

由相关的数据类型、数据对象和子程序组成的可分别编译的软件部件。

参见： 数据抽象 data abstraction（2.388）、封装 encapsulation（2.542）和信息隐藏 information hiding（2.751）。

2.1090

包图 package diagram

包图由包和包之间的联系构成。包是一种对模型元素进行成组组织的通用机制，它把语义上相近的可能一起变更的模型元素组织在同一个包里，便于理解复杂的系统，控制系统结构各部分间的接缝。包图是维护和控制系统总体结构的重要建模工具。

2.1091

打包 packageing

在软件开发中，把模块赋予段，以便为计算机执行时作为不同的物理单元处理。

2.1092

填充 padding

a) 用具有虚拟的字符、字或记录的数据填写固定长度块的技术。

b) 用于填写固定长度数据块的虚拟字符、字或记录。

2.1093

页[面] page

a) 固定长度的数据或在存储分配中作为一个单元对待的计算机程序段。

参见： 调页 paging（2.1101）。

b) 在虚拟存储器系统中，一固定长度的数据或计算机程序段。此程序有虚拟地址并在主内存和辅助存储器之间作为一个单元传送。

c) 在视频显示终端上，一屏信息。

2.1094

页亏 page breakage

当数据或计算机程序的最后页未填满分配给它的存储器的整块时，主内存未用的部分。

参见： 调页 paging（2.1101）。

2.1095

页帧 page frame

主内存的一个块，它具有能保持一页的大小。

参见： 调页 paging（2.1101）。

2.1096

页对换 page swapping

在主内存和辅助存储器之间页的交换。

参见： 调页 paging（2.1101）。

2.1097

页表 page table

标识在存储器中页的单元并给出这些页充分的属性的表。

参见： 调页 paging（2.1101）。

2.1098

页轮流 page turning

见： 调页 paging（2.1101）（c）。

2.1099

页 0 page zero

在存储器分配的调页方法中，页序列中的第一页。

2.1100

调页程序 pager

启动和控制在主内存和辅助存储器之间页的传送的例程。

参见： 调页 paging（2.1101）。

2.1101

分页，调页 paging

块分配 block allocation

a) 一种存储分配技术，其中的程序或数据分为长度固定的称为页的块，主内存也分成同样长度称为页帧的块，页存储在页帧中，不必连续。或按逻辑顺序。

相对：邻接分配 contiguous allocation（2.329 条）。

b) 一种存储分配技术，在此技术中，程序或数据分为长度固定的称为页的块，主内存也分成同样长度称为页帧的块，当需要时，页在主内存和辅助存储器之间传送。

参见： 先行调页 anticipatory paging（2.66）、 按需调页 demand paging（2.431）和虚拟存储器 virtual storage（2.1840）。

c) 像在 b 中那样传送页。与页旋转 page turning（2.1098）同义。

参见： 页 page（2.1093）、页亏 page breakage（2.1094）、页帧 page frame（2.1095）、页对换 page swapping（2.1096）、页表 page table（2.1097）、页 0 page zero（2.1099）、调页程序 pager（2.1100）和工作集 working set（2.1853）。

2.1102

并行 parallel

a) 用于说明整体的分别部分的同时传送、发生或处理。例如，字符的位对于各个部分用分别的设施同时处理。

相对： 串行 serial（2.1442）a）。

b) 见： 并发 concurrent（2.298）。

2.1103

并行构造 parallel construct

由两个或多个能同时发生的过程组成的程序构造。

2.1104

并行运行 parallel run

为比较和可信起见，两个信息处理系统以同一应用程序和源数据进行的操作，其中一个系统是给定的，另一个是其预期替换者。

2.1105

参数 parameter

a) 一种变量，针对一特定的应用程序可给定一固定值。

参见： 适应参数 adaptation parameter（2.42）。

b) 用于在软件模块之间传递值的常数、变量或表达式。

参见： 自变量 argument（2.74）和形参 formal parameter（2.645）。

2.1106

参数化元素 parameterized element

模板 template

对带有一个或多个未绑定参数的类的描述符。

2.1107

父 parent

在泛化联系中，对另一元素即子的泛化。

参见： 子类 subclass（2.1613）、子类型 subtype（2.1623）。

相对： 子 child（2.207）。

2.1108

Pareto 分析 Pareto analysis

采用缺陷原因从最重要到最不重要的排序方法对缺陷进行分析。Pareto 分析的根据是以 19 世纪经济学家 Vilfredo Pareto 命名的原理，即大多数后果是由于相对少的原因引起的，也就是 80%后果是由可能原因中的 20%所造成的。

2.1109

语法分析 parse

由分解语言单元为更多的基本的子单元并建立它们之间的关系以确定语言单元的句法结构。例如，把块分解为语句，把语言分解为表达式，把表达式分解为操作符和操作数。

2.1110

语法分析程序 parser

对计算机程序或其他文本进行语法分析的软件工具，常常作为汇编、编译、解释或分析的第一步。

2.1111

部分正确性 partial correctness

在正确性证明中，指出程序的输出断言是它的输入断言和处理步骤的合乎逻辑的结果。

相对： 完全正确性 total correctness（2.1750）。

2.1112

参与 participate

模型元素连接到一个关系或对具体化的关系。例如，类参与关联，施动者参与用况。

2.1113

分区 partition

a) （用于活动图）在活动图中，组织动作职现的部分。

参见： 泳道 swimlane（2.1634）。

b) （用于体系结构）一种在同一抽象级或跨分层体系结构的各层的有关类目或包的集合。分区表示跨全系结构的纵断片，层则表示横断片。

相对： 层 layer（2.842）。

2.1114

划分 partitioning

分解；由整体到各组成部分的分割。

2.1115

遍 pass

数据集处理过程中的单个周期，通常执行整个过程的一部分。例如，汇编程序通过源程序一遍、排序程序通过数据集一遍。

2.1116

通过／失败准则 pass/fail criteria

用于确定一软件项或一软件特性是否通过测试或失败的决定准则。

参见： 测试准则 test criteria（2.1699）。

2.1117

修补，补丁 patch

a) 直接对目标程序所做的修改，而不从源程序重新汇编或编译。

b) 对源程序所做的修改，作为最后固定或事后思考。

c) 对源程序或目标程序的任何修改。

d) 执行如在 a)、b)、c)中的修改。

2.1118

路径 path

a) 在软件工程中，可能在计算机程序的执行中执行的指令序列。

b) 在文件访问中，规定文件的存储单元的目录和子目录名的层次序列。

2.1119

路径分析 path analysis

对程序进行的如下分析：以标识通过该程序的所有可能的路径，检测不完全的路径，或发现不处在任何路径上的程序部分。

2.1120

路径条件 path condition

为了要执行具体的程序路径所必须满足的一组条件。

2.1121

路径表达式 path expression

一个逻辑表达式。它说明为了执行具体的程序路径所必须满足的输入条件。

2.1122

路径测试 path testing

设计以执行通过计算机程序的全部或所选择的路径的测试。

相对： 分支测试 branch testing（2.158）和语句测试 statement testing（2.1575）。

2.1123

相依耦合 pathological coupling

一类耦合，在其中，一个软件模块影响或依赖于另一个的内部实现。

相对：公共环境耦合 common-environment coupling(2.243)、内容耦合 content couping(2.327)、控制耦合 control coupling（2.336）、数据耦合 data couping（2.393）和混合耦合 hybrid coupling

（2.714）。

2.1124

模式敏感故障 pattern-sensitive fault

见： 数据敏感故障 data-sensitive fault（2.388）。

2.1125

暂停 pause

挂起计算机程序的执行。与： 暂停 halt（2.688）（b）同义。

相对： 停止 stop（2.1590）。

2.1126

物理配置审核 PCA

physical configuration audit 的缩略语。

2.1127

程序设计语言 PDL

program design language 的缩略语。

2.1128

概要设计评审 PDR

preliminary design review 的缩略语。

2.1129

同行评审 peer review

由软件工作产品生产者的同行遵循已定义的规程对工作产品进行的评审，以标识工作产品存在的缺陷和需要的改进。

2.1130

完善性维护 perfective maintenance

执行以改善计算机程序的性能、可维护性或其他属性的软件维护。

相对：适应性维护 adaptive maintenance(2.43)和纠正性维护 corrective maintenance(2.354)。

2.1131

性能 performance

系统或部件在给定的约束，例如，速度、精度或存储器使用条件下实现指定的功能的程度。

2.1132

性能评价 performance evaluation

为确定运行目标达到了何种有效程度而对系统或系统部件的技术评价。

2.1133

性能需求 performance requirement

在功能需求上加上条件的需求。例如，在规定的速度、精度或存储器使用条件下执行指定函数的需求。

相对： 设计需求 design requirement（2.451）、功能需求 functional requirement（2.667）、实现需求 implementation requirement（2.730）、接口需求 interface requirement（2.798）和物理需求 physical requirement（2.1140）。

2.1134

性能规格说明 performance specification

规定系统或部件必须具有的性能特征的文档。这些特征典型地包括：速度、精度和存储器使用。常常是需求规格说明书的一部分。

2.1135

性能测试 performance testing

评价系统或部件与规定的性能需求的依从性的测试行为。

参见： 功能测试 functional testing（2.669）。

2.1136

定期评审或活动 periodic review/activity

在规定的固定时间间隔进行的评审或活动。

参见： 事件驱动的评审或活动 event-driven review/activity（2.574）。

2.1137

持久对象 persistent object

一种在创建它的进程或线程不复存在时依然存在的对象。

2.1138

Petri 网 Petri net

信息流的一种抽象的、形式的模型，其中指出一系统的静态和动态性质。petri 网通常表示成图。图中有两类用弧彼此相连的结点（称为位置和转移）和指示其动态性能的标记〈称为记号〉。

参见： 状态图 state diagram（2.1570）。

2.1139

物理配置审核 physical configuration audit

验证已建立的某个配置项遵循定义它的技术文档的审核行为。

参见： 功能配置审核 functional configuration audit（2.662）。

2.1140

物理需求 physical requirement

规定系统或系统部件必须具有的物理特征的一种需求，例如，材料、形状、尺寸、重量。

相对： 设计需求 design requirement（2.451）、功能需求 functional requirement（2.667）、实现需求 implementation requirement（2.730）、接口需求 interface requirement（2.798）和性能需求 performance requirement（2.1133）。

2.1141

物理系统 physical system

a) 某一模型的材料。

b) 连接起来的各物理单元的一种如下汇集：能包括软件、硬件和人员，组织起来去达到特定目的。物理系统能由一个或多个模型，可能以不同的观点加以描述。

相对：系统 system（2.1645）。

2.1142

试点项目 pilot project

一种项目，它用于在实际的但有限的运行操作下，对信息处理系统的初级版本进行测试，之后，将用于对该系统的最后版本进行测试。

2.1143

管道 pipeline

一种软件或硬件设计技术，其中，一个进程的输出用作第二个的输入，第二个的输出用作第三个的输入，等等，常常在单个周期时间内同时性。

2.1144

计划标准 plan standard

描述在规定的资源内实现定义的目标或工作的方案的特征的标准。

2.1145

重演 playback

见： 可逆执行 reversible execution（2.1391）。

2.1146

指针 pointer

规定另一数据项位置的一种数据项、例如，一数据项，它规定下一个要处理的记录的地址。

2.1147

方针 policy

一种指导原则，一般由高层管理者制定，并由组织或项目用来影响和确定决策。

2.1148

端口到端口时间 port-to-port time

在激活至输入接口的应用程序和输出接口出现响应之间消逝的时间。

参见：响应时间 response time（2.1377）、思考时间 think time（2.1729）和往返时间 turnaround time（2.1775）。

2.1149

可移植性 portability

可运输性 transportability

系统或部件能从一种硬件或软件环境转换至另外一种环境的特性。

参见： 机器独立 machine independent（2.889）。

2.1150

后[置]条件 postcondition

一种在某一操作完成之时必须为真的约束。

2.1151

后测试的迭代 post-tested iteration

见： “直至”语句 UNTIL（2.1795）。

2.1152

后同步码断点 postamble breakpoint

见： 跋断点 epilog breakpoint（2.559）。

2.1153

事后转储 postmortem dump

在计算机程序异常终止后产生的转储。

参见： 变更转储 change dump（2.196）、动态转输 dynamic dump（2.519）、内存转储 memory dump（2.928）、选择的转储 selective dump（2.1425）、快照转储 snapshot dump（2.1466）、静态的转储 static dump（2.1582）。

2.1154

后处理器 postprocessor

在主要过程完成以后执行某些最后的处理步骤的计算机程序。例如，为了输出，重新格式数据的例程。

相对： 预处理器 preprocessor（2.1165）。

2.1155

开发后评审 post-development review

在系统运转使用已经达到稳定状态之后，对该系统效果的一种评论。

2.1156

实践，惯例 practices

用于描述对于软件开发过程的约束的统一的方法的需求。

参见： 约定 conventions（2.347）和标准 standards（2.1562）。

2.1157

编译指示 pragma

见： 伪指令 pseudo instruction（2.1280）。

2.1158

预测试迭代 pre-tested iteration

见： “当”语句 WHILE（2.1847）。

2.1159

前同步码断点 preamble breakpoint

见： 序断点 prolog breakpoint（2.1268）。

2.1160

精度 precision

对于说明的量的精确或差异的程度。例如，2 位十进制数字对 5 位十进制数字。

相对： 准确 accuracy（2.22）。

2.1161

预编译程序 precompiler

一种计算机程序，它对源代码（其中有一部分可能是编译程序无法接受的）进行预处理，以产生与之等效但可为编译程序接受的代码。例如，把结构化 FORTRAN 转换为 GB 3507 FORTRAN 的预处理程序。

参见： 预处理器 preprocessor（2.1165）。

2.1162

前[置]条件 precondition

一种当某一操作被启用时必须为真的约束。

2.1163

概要设计 preliminary design

a) 对于系统或部件分析各种设计方案和定义软件体系结构、部件、接口和提出时间和规模方面的估计的过程。

参见： 详细设计 detailed design（2.462）、功能设计 function design（2.665）。

b) a)中过程的结果。

2.1164

概要设计评审 preliminary design review(PDR)

a) 一种评审行为，它评价进展、技术合理和对于一个或多个配置项所选择的设计方法的风险解决方案、确定每一个设计与配置项的需求的兼容性、评价与选择的制造方法和过程相关的技术风险的定义和评定的程度、建立配置项之间和配置项与设备、设施、软件和人的其他项之间的物理的和功能的接口的存在和兼容性，若可能，评价主要的操作和支持文档。

参见： 关键设计评审 critical design review（2.365）和系统设计评审 system design review（2.1650）。

b) 任何的硬件或软件部件如 a 中的评审。

2.1165

预处理程序，预处理器 preprocessor

在主要过程之前，执行某些处理步骤的计算机程序或例程。例如，预编译程序或为了处理重新格式代码和数据的例程。

相对： 后处理器 postprocessor（2.1154）。

2.1166

预存储 prestore

在程序或例程进入之前，存储计算机程序或例程要求的数据。

2.1167

模样打印 prettyprinting

使用缩排、空行和其他可视的线索以显示程序的逻辑结构。

2.1168

预防性维护 preventive maintenance

在问题发生之前，为防止问题发生所进行的维护。

2.1169

主承制方 prime contractor

管理一个或多个产品的设计、开发和（或）制造的子合同的个人、合伙组织、公司或协会。

2.1170

原语类型 primitive type

见： 原子类型 atomic type（2.103）。

2.1171

优先权，优先级 priority

赋予项的重要性的等级。

2.1172

优先级中断 priority interrupt

允许比当前正在执行的进程有更高优先级的进程执行的一种中断。

2.1173

私有类型 private type

一种数据类型，它的结构和可能的值被定义但不透露给类型的用户。

参见： 信息隐藏 information hiding（2.751）。

2.1174

特权指令 privileged instruction

只有管理程序能执行的计算机指令。

2.1175

特权状态 privileged state

见： 监督状态 supervisor state（2.1627）。

2.1176

问题描述 problem description

对问题的一种陈述，它可包括用于对其求解的数据、方法、步骤和算法的描述。

2.1177

面向问题的语言 problem-oriented language

设计为了解决给定类问题的编程语言。例子是表处理语言（list processing language）、信息检索语言（information retrieval language）和仿真语言（simulation language）。

2.1178

问题状态 problem state

从属的状态 slave state

用户状态 user state

在计算机系统的操作中，非管理程序能执行的状态。

相对： 监督状态 supervisor state（2.1627）。

2.1179

过程内聚度 procedural cohesion

一类内聚度，其中，由软件模块执行的任务完全有助于给定的程序过程，例如一个重复或决定过程。

相对： 一致内聚度 coincidental cohesion（2.229）、通信内聚度 communicational cohesion（2.246）、功能内聚度 functional cohesion（2.661）、逻辑内聚度 logical cohesion（2.874）和顺序内聚度 sequential cohesion（2.1439）。

2.1180

过程语言 procedural language

面向过程的语言 procedure-oriented language

用户说明计算机必须以给定的顺序执行的特定的指令集的编程语言。所有广泛使用的编程语言都是这种类型的语言。

参见： 代数语言 algebraic language（2.53）、算法语言 algorithmic language（2.56）、表处理语言 list processing language（2.860）和逻辑编程语言 logic programming language（2.873）。

相对： 非过程语言 nonprocedural language（2.1025）。

2.1181

过程，规程 procedure

a) 计算机程序的一个部分，它被命名并实现一个特定的动作。

b) 为执行给定的任务而采取的动作的经过。

c) 如像 b)中采取动作的经过所写的描述。例如，说明的测试过程。

2.1182

面向过程的语言 procedure-oriented language

见： 过程语言 procedural language（2.1180）。

2.1183

过程，进程，处理【动】 process

a) 为给定目的所执行的步骤序列、例如，软件开发过程。

b) 由操作系统调度程序管理的一可执行单元。

参见： 任务 task（2.1679）和作业 job（2.825）。

c) 对数据进行操作。

2.1184

过程评估 process assessment

对组织的软件过程对照与参考模型相一致的模型的原则性的评价。

2.1185

过程属性 process attribute

适用于任何过程的可测量的过程能力特性。

2.1186

过程属性评级 process attribute rating

对于被评估过程的过程属性达到已定义能力级别的判断。

2.1187

过程能力 process capability

某过程达到所要求目标的能力。

2.1188

过程能力基线 process capability baseline

对典型环境下遵循一个特定过程通常所能达到的预期结果范围的一种文档化的特征描述。一般在组织层上建立过程能力基线。

2.1189

过程能力确定 process capability determination

在一个组织内部，对选定的软件过程对照某个目标能力所进行系统性的评估和分析，其目的是识别出所使用过程的强项、弱项及相关风险以满足某个特定要求。

2.1190

过程能力确定发起方 process capability determination sponsor

发起过程能力确定的组织，组织的某个部分或人。

2.1191

过程能力等级 process capability level

表示所执行的过程能力增强六点有序（过程能力）度量的某个点、每个等级均建立在其下级的基础之上。

2.1192

过程能力评级 process capability rating

一种对某个已评估过程的过程属性评定所达到的过程能力等级的表示。

2.1193

过程范畴 process category

涉及活动的方面的一组过程。

注：过程范畴涉及活动的 5 种一般方面：顾客－供方、工程、支持、管理和组织。

2.1194

过程环境 process context

对过程属性评定的判定、理解和可比性有影响的，并在评估输入中文档化了的一组因素。

2.1195

过程描述 process description

过程主要成分的操作性定义。即以全面的、精确的、可验证的方式规定过程的要求、设计、行为或其他特征的文档。它也可能包括用以确定这些条款是否已得到满足的规程。在任务层、项目层或组织层上均可能有过程描述。

2.1196

过程开发 process development

定义和描述过程的行动。它可包括策划、构造、设计、实施和确认。

2.1197

过程维 process dimension

过程的集合，它包括过程参考模型的功能方面和过程能力。

注：将过程划分为相关活动的类别。

2.1198

过程改进 process improvement

改进组织过程采取的行动，使过程满足组织经营的需要和更有效地实现组织的经营目标。

2.1199

过程改进行动 process improvement action

某项计划的和执行的行动以改进所有或部分软件过程。

注：（一项）过程改进行动能有助于实现一个以上的过程目标。

2.1200

过程改进纲要 process improvement programme

与特定改进目标实现相关的所有战略、方针、目标、职责和活动。

注：一项过程改进计划可以横跨一个以上的过程改进的完整的周期。

2.1201

过程改进项目 process improvement project

任何为实现某个特定改进所形成的过程改进计划的子集合。

2.1202

过程管理 process management

为开发一产品或执行服务要执行的工作的方向、控制和协调。例子是质量保证。

2.1203

过程测量 process measurement

为测量过程及其生成的产品而采用的一组定义、方法和活动，其目的是为了描述和了解该过程。

2.1204

过程结果 process outcome

过程成功地实施的可观测的结果。

注：主要过程结果列表形成了参考模型中每个过程的部分描述。

2.1205

过程性能 process performance

执行某个过程实现其目标的程度。

2.1206

过程性能基线 process performance baseline

对执行一个过程所达到实际结果的一种文档化的特征描述。它用作将实际过程性能与期望的过程能力进行比较的基准。一般在项目层上建立过程性能基线，尽管最初的过程性能基线通常由组织的过程能力基线导出。

参见： 过程能力基线 process capability baseline（2.1188）。

2.1207

过程轮廓 process profile

某个已评估过程的过程属性鉴定的集合。

2.1208

过程目的 process purpose

执行过程的高层次的可测量目标和有效地实施过程的可能结果。

2.1209

过程标准 process standard

处理在制造或完成产品中所用的动作或操作系列的标准。

2.1210

过程剪裁 process tailoring

通过对过程的过程元素的细节或其他不完备的规范进行提炼、调整和（或）补充的方法来生成过程描述的一种活动。在过程剪裁期间通常说明项目的特定业务需要。

2.1211

产品 product

要交付给用户的一套完整的计算机程序、过程以及有关的文档和数据。

2.1212

产品分析 product analysis

用手工或自动的方法计算产品以确定产品是否有一定的特征的过程。

2.1213

产品基线 product baseline

在配置管理中，在它的生存周期的生产、操作、维护和后勤支持期间，定义一配置项的初始的经批准的技术文档（对于软件，包括源代码清单）。

2.1214

产品认证 product certification

见： 认证 certification（2.192）。

2.1215

产品配置标识 product configuration identification

在它的生存周期的生产、操作、维护和后勤支持期间，定义一配置项的当前批准的或有条件批准的技术文档。它描述配置项的所有必需的物理的或形式的，适合的以及功能特征，所选择的功能特征为产品验收测试指定。

参见： 产品基线 product baseline（2.1213）。

相对： 分配的配置标识 allocated configuration identification（2.58）和功能配置标识 functional configuration identification（2.663）。

2.1216

产品工程 product engineering

定义、设计和构造或装配一产品的技术过程。

2.1217

产品库 product library

一种软件库，其中含有已被批准供当前运行使用的软件。

2.1218

产品管理 product management

在产品的开发周期期间，定义、协调和控制产品的特征。例子是配置管理（configuration management）。

2.1219

产品规格说明 product specification

a) 规定系统或部件必须实现的产品的设计文档。注：对于软件，此文档描述软件构造时的版本。

参见： 设计描述 design description（2.442）。

b) 描述为潜在的客户和用户考虑的计划的或已存在的产品特征的文档。

2.1220

产品标准 product standard

一种标准，它处理在标记或完成产品中所用的动作或操作序列。

2.1221

产品支持 product support

提供信息、帮助和培训，以安装并使软件在其目标环境中可操作并向用户发布改进的功能。

2.1222

产品库 production library

一软件库，包装为当前操作的用户使用的已批准的软件。

相对： 主库 master library（2.916）、软件开发库 software development library（2.1487）、软件储藏库 software repository（2.1529）和系统库 system library（2.1657）。

2.1223

专业标准 professional standard

标识专业为学科并与其他专业区分的标准。

2.1224

外廓，剖面 profile

a) 一种如下的衍型的包：包含有为特定领域或目的已采用扩展机制加以客户化的模型元素，例如衍型、加标定义和约束。外廓还可规定所依赖的模型库和所扩展的元模型子集。

b) 沿时间轴，计划或预测结果与实际结果的一种比较。一般用图表示。

2.1225

程序，编程【动】 program

a) 参见： 计算机程序 computer program（2.276）。

b) 写计算机程序。

2.1226

程序体系结构 program architecture

计算机程序部件之间的结构和关系。程序体系结构也可以包括它和程序运行环境之间的程序界面。

2.1227

程序块 program block

在面向问题的语言中，计算机程序的子部分，它用于把语句分组、划分例行程序、规定存储分配、确定标号的可使用性、或者为其他一些目的而将计算机程序的分段。

2.1228

程序正确性 program correctness

参见： 正确性 correctness（2.355）。

2.1229

程序计数器 program counter

见： 指令计数器 instruction counter（2.771）。

2.1230

程序定义语言 program definition language

见： 程序设计语言 program design language（2.1231）。

2.1231

程序设计语言 program design language(PDL)

具有特殊结构的规格说明语言，它有时带有验证协议，用于开发、分析和说明程序设计。

参见： 设计语言 design language（2.447）、硬件设计语言 hardware design language（2.692）和伪代码 pseudo code（2.1279）。

2.1232

程序扩展 program extension

对现存软件进行增强以扩大程序能力的范围。

2.1233

程序流程图 program flowchart

见： 流程图 flowchart（2.637）。

2.1234

程序指令 program instruction

在源程序中的计算机指令。注：程序指令与从汇编、编译或其他解释过程产生的计算机指令是有区别的。

2.1235

程序插装 program instrumentation

a) 插入到计算机程序中的探头。如指令或断言，以利于执行监控、正确性证明、资源监控或其他活动。

b) 准备探头并把它插入到计算机程序中去的过程。

2.1236

程序库 program library

计算机程序的有组织的集合。

参见： 软件库 software library（2.1505）和系统库 system library（2.1657）。

2.1237

程序列表 program listing

计算机程序的的源代码，有时是目标代码的打印输出或其他人们可读的显示。

2.1238

系统维护手册 program maintenance manual

提供维护某一程序所必须的信息的一种文件。

2.1239

程序变异 program mutation

a) 从目标版本有目的地变更的计算机程序，它用以估计程序测试用例检测变更的能力。

参见： 变异测试 mutation testing（2.1001）。

b) 如a)中那样创建变更的程序的过程。

2.1240

程序网络图 program network chart

显示两个或多个计算机程序之间关系的图。

2.1241

程序保护 program protection

为预防对计算机程序的任何非授权存取或修改而施行的内部或外部控制。

2.1242

程序敏感故障 program-sensitive fault

当执行的程序步的某些具体序列失败引起的故障。

相对： 数据敏感故障 data-sensitive fault（2.403）。

2.1243

程序规格说明 program specification

a) 计算机程序的任何规格说明。

参见：设计规格说明 design specification(2.453条)、功能规格说明 functional specification（2.668）、性能需求 performance requirement（2.1134）和需求规格说明 requirement specification（2.1367）。

b) 与设计规格说明 design specification（2.453）同义。

2.1244

程序状态字 program status word(PSW)

a) 包含规定计算机程序当前状态的信息的计算机字。此信息可能包含出错指示、下一条要执行的指令的地址、当前允许的中断等等。

b) 包含如a)中这样的程序状态字的特殊目的的寄存器。

2.1245

程序结构图 program structure diagram

见： 结构图 structure chart（2.1605）。

2.1246

程序支持库 program support library

见： 软件开发库 software development library（2.1487）。

2.1247

程序综合 program synthesis

把程序规格说明变换为实现那个规格说明的程序的软件工具的使用。

2.1248

程序确认 program validation

计算机程序确认 computer program validation

参见： 确认 validation（2.1819）。

2.1249

可编程的断点 programmable breakpoint

一种断点，当它启动时，自动引用以前规定的调试过程。

参见：代码断点 code breakpoint(2.220)、数据断点 data breakpoint(2.915)、动态断点 dynamic breakpoint（2.516）、跋断点 epilog breakpoint（2.559）、序断点 prolog breakpoint（2.1268）和静态断点 static breakpoint（2.1580）。

2.1250

程序员手册 programmer manual

提供为开发或修改给定的计算机系统的软件所必需的信息的文档。典型地描述的是计算机系统的设备配置、操作特征、编程特性、输入/输出特性和编译或汇编特性。

参见： 诊断手册 diagnostic manual（2.476）、安装手册 installation manual（2.766）、操作员手册 operator manual（2.1068）、支持手册 support manual（2.1631）和用户手册 user manual（2.1813）。

2.1251

编程，程序设计 programming

a) 程序的编写工作。

b) 用源程序语言或某种代码为程序编码之前的一部分工作作设计。

2.1252

编程语言，程序设计语言 programming language

用于表达计算机程序的语言。

参见： 汇编语言 assembly language（2.86）、高级语言 high order language（2.702）和机器语言 machine language（2.890 条）。

相对： 查询语言 query language（2.1303）和规格说明语言 specification language（2.1550）。

2.1253

程序设计支持环境 programming support environment

通过单一命令语言提供在整个软件生存周期中的程序支持能力的软件工具的集成的集合。此环境典型地包括为规定、设计、编辑、编译、装入、测试、配置管理和项目管理中所用的工具。有时称集成编程支持环境（integrated programming support environment）。

参见： 脚手架 scaffolding（2.1409）。

2.1254

编程系统 programming system

编程语言和为了在给定的计算机系统中使用这些语言所必需的支持软件（编辑程序、编译程序、连接程序等）的集合。

2.1255

项目 project

需要协同工作的一组任务，其目的在于开发和（或）维护一个具体的产品。产品可以包括硬件、软件或其他成分。一般项目有自己的经费、成本核算和交付进度。

2.1256

项目控制 project control

按项目计划对项目的进展、指导、质量和资源利用进行监控的各有关活动。

2.1257

项目定义的软件过程 project's defined software process (PDSP)

项目所采用的软件过程的操作性定义。项目定义的软件过程是一个妥善定义的软件过程，它用软件标准、规程、工具和方法来描述，并且通过剪裁组织的标准软件过程以适合项目的具体特征。

参见： 组织的标准软件过程 organization's standard software process（2.1077）。

2.1258

项目文件 project file

项目簿 project notebook

中心储藏处的与项目相关的资料。内容典型地包括：备忘录、计划、技术报告和相关项。

2.1259

项目库 project library

见： 软件开发库 software development library（2.1487）。

2.1260

项目管理 project management

与项目规划和项目控制有关的各种活动。

2.1261

项目经理 project manager

对整个项目的业务工作负全面责任的一种角色；或对建立软件系统或硬件/软件系统的项目进行指导、控制、管理和调整的人员。项目经理是对顾客最终负责的人员。

2.1262

项目簿 project notebook

见： 项目文件 project file（2.1258）。

2.1263

项目计划 project plan

描述项目所采取的技术和管理方法的文档。此计划典型地描述要做的工作、所需要的资源、使用的方法、采用的过程、要满足的进度表和项目组织方法。例如，软件开发计划。

2.1264

项目进度（表） project schedule

见： 项目计划 project plan（2.1263）。

2.1265

项目软件经理 project software manager

对项目中的全部软件活动负有全面责任的一种角色。项目软件经理控制项目的全部软件资源，项目经理按照软件承诺与之打交道。

2.1266

项目规格说明 project specification

某一项目的目标、要求、范围以及与其他项目的关系的一种规格说明。

2.1267

投影 projection

从集合到其子集的一种映射。

2.1268

序断点 prolog breakpoint

前同步码断点 preamble breakpoint

直至进入程序或例程启动的断点。

参见：代码断点 code breakpoint(2.220)、数据断点 data breakpoint(2.391)、动态断点 dynamic breakpoint（2.516）、可编程断点 programmble breakpoint（2.1249）、静态断点 static breakpoint（2.1580）。

2.1269

提示 prompt

a) 由计算机系统显示的，请求从系统的用户输入的符号或消息。

b) 显示如 a 中的符号或消息。

2.1270

正确性证明 proof of correctness

a) 用于数学上证明计算机程序满足它规定的需求的正式的技术。

参见： 断言 assertion（2.87）、正式规格说明 formal specification（2.648）、归纳断言方法 inductive assertion method（2.747）、部分正确性 partial correctness（2.1111）和完全正确性 total correctness（2.1750）。

b) 应用在 a 中的技术导致的证明。

2.1271

性质 property

表征元素某一特性的一种命名值。性质有语义作用。某些性质在 UML 预定义、另一些可由用户定义。

参见： 加标值 tagged value（2.1673）。

2.1272

建议能力 proposed capability

由组织提出的，为满足特定要求而使用的过程能力。

注：对于核心过程能力确定而言，建议能力是指组织当前已评估的能力，而对于扩展的过程能力确定而言，建议能力是增强的能力或构造的能力之一。

2.1273

保护 protection

限制对计算机系统进行全部或部分存取或使用的一种安排。

2.1274

保护异常 protection exception

当程序企图写至存储器中受保护的区域时引起的异常。

参见： 寻址异常 addressing exception（2.50）、数据异常 data exception（2.395）、操作异常 operation exception（2.1059）、溢出异常 overflow exception（2.1082）和下溢异常 underflow exception（2.1786）。

2.1275

协议 protocol

支配系统内进程、设备和其他部件交互的一组约定。

2.1276

原型 prototype

系统的最初的类型、形式或例子，它用作为系统的以后阶段或最后的完整的版本的模型。

2.1277

原型法 prototyping

一种硬件和软件开发技术，其中，硬件或软件的部分或整体的最初版本被开发，允许用户反馈、确定可行性、研究定时或开发过程支持的其他问题。

参见： 快速原型法 rapid prototyping（2.1307）。

2.1278

见习评估师 provisional assessor

具有技能和能力的，在合格的评估师指导和监控下执行评估的个人。

2.1279

伪码 pseudo code(pseudocode)

用于表达计算机程序设计的编程语言结构和自然语言的组合。例如：

```
IF the data arrives faster than expected（若 数据到达比预料的快,）
    THEN reject every third input.（则在每当两个输入后 拒绝第三个输入。）
    ELSE process all data received)（否则 处理接收到的所有数据。）
ENDIF
```

2.1280

伪指令 pseudo instruction

编译指示 pragma

伪操作 pseudo-op

伪操作 pseudo operation

为汇编程序或编译程序提供信息或方向而不转换为目标语言指令的源语言指令。例如，规定所希望的源代码列表格式的指令。

2.1281

伪操作 pseudo operation

见： 伪指令 pseudo instruction（2.1280）。

2.1282

伪操作 pseudo-op

见： 伪指令 pseudo instruction（2.1280）。

2.1283

伪状态 pseudo-state

在状态机中，一种虽有状态形式却无状态行为的顶点。伪状态包括初始顶点和历史顶点。

2.1284

程序状态字 PSW

program status word 的缩略语。

2.1285

已公布模型 published model

一种已予“冻结”，且对存储库实例化与支持地其他模型定义成为可用的模型。“已冻结”模型的模型元素不能变更。

2.1286

下推式存储器 pushdown storage

按后进先出〈LIFO〉的方法处理数据的存储器。在这种方法中下一个要取的项是仍在存储器中的最后存入的那个项。

参见： 栈 stack（2.1560）。

2.1287

质量保证 QA

quality assurance 的缩略语。

2.1288

质量控制 QC

quality control 的缩略语。

2.1289

鉴定 qualification

一个正式的过程，通过这个过程确定系统或部件是否符合它的规格说明，是否可在目标环境中适合于操作使用。

2.1290

鉴定需求 qualification requirement

准则或一组条件，当一个产品符合这些准则或条件时，就确定它符合规格说明并可以在其目标环境中使用。

2.1291

合格性测试 qualification testing

确定系统或部件是否适合于操作使用的测试行为。

参见： 验收测试 acceptance testing（2.19）、开发测试 development testing（2.468）和运行测试 operational testing（2.1065）。

2.1292

限定符 qualifier

一种关联属性或属性元组，其值将跨某一关联与一个对象有关的对象集加以划分。

2.1293

质量 quality

a) 系统、部件或过程满足规定的需求的程度。

b) 系统、部件或过程满足客户或用户需要或期望的程度。

2.1294

质量保证 quality assurance(QA)

a) 为使某项目或产品遵循已建立的技术需求提供足够的置信度，而必须采取的有计划的和有系统的全部动作的模式。

b) 设计以估算产品开发或制造过程的一组活动。

相对： 质量控制 quality control（2.1296）。

2.1295

质量属性 quality attribute

质量因素 quality factor

影响一个项的质量的特性或特征。

注： 在质量属性的层次中，高级属性可能称为质量因素，低级属性，称为质量属性。

2.1296

质量控制 quality control(QC)

注： 直至此时，在软件工程中，此术语无标准化的含义。候选的定义是：

注： a) 设计以估算产品开发或制造过程的一组活动。

b) 相对： 质量保证 quality assurance（2.1294）b）。

c) 验证自己的工作或协作者工作的过程。

d) 与质量保证 quality assurance（2.1294）同义。

2.1297

质量评价 quality evaluation

对实体能满足特定需求的程度的系统检测。

注：当按照合同为某个特定用户开发产品时，其需求是正式规定的、当产品是为非特定用户开发时，如消费软件，其需求由开发组织来规定、当用户为比较和选择的目的评价产品时，需求可以是更一般的。

2.1298

品质因数，质量因数 quality factor

见： 质量属性 quality attribute（2.1295）。

注： 在质量属性的层次中，高级属性可能称为质量因数，低级属性，称为质量属性。

2.1299

使用质量 quality in use

特定用户使用产品满足其要求的程度，以达到在特定应用环境中的有效性、生产率、安全性和满意度等特定目标。

注：这种使用质量的定义类似于 ISO 9241-11 中可用性的定义。在 GB/T XXXXX 中术语“可用性”用来指在 ISO/IEC 9126 中描述的软件质量特征。

2.1300

质量度量 quality metric

a) 一种项具有给定的质量属性的程度的定量度量。

b) 一种函数，它的输入是软件数据，输出是能解释该软件具有给定的质量属性的程度的单一个数值。

2.1301

质量模型 quality model

一组特性及特性之间的关系，它提供规定质量需求和评价质量的基础。

2.1302

定量控制 quantitative control

适于分析软件过程、标识软件过程性能变化的特殊原因和使软件过程性能处于妥善定义的范围内的任何一种定量技术或基于统计的技术。

2.1303

查询语言 query language

用于访问存储在数据库中的信息的语言。

相对： 编程语言 programming language（2.1252）和规格说明语言 specification language（2.1550）。

2.1304

队列 queue

一种把项附加至它的最后位置而从它的第一个位置检索的队列。

2.1305

静寂 quiescing

由拒绝工作的新请求使设备或系统暂停的过程。

2.1306

随机失效 random failure

一种失效，它的发生是随意性的，在统计情况下是不可预测的。

参见： 间歇故障 intermittent fault（2.803）和瞬时错误 transient error（2.1767）。

2.1307

快速原型法 rapid prototyping

一类原型法，其重点是放在过程早期就开发出原型，使反馈和分析提前以支持开发过程。

参见： 以数据结构为中心的设计 data structure-centered design（2.405）、增量开发 incremental development(2.738)、输入-处理-输出 input-process-output(2.762)、模块分解 modular decomposition(2.973)、面向对象的设计 object-oriented design(2.1037)、螺旋模型 spiral model（2.1553）、逐步细化 stepwise refinement（2.1587）、结构设计 structured design（2.1607）、事务分析 transaction analysis（2.1760）和转换分析 transform analysis（2.1764）。

相对： 瀑布模型 waterfall model（2.1844）。

2.1308

评级 rating

把测量值映射到相应的评定等级的活动，用于确定与软件某一质量特性相关的等级。

2.1309

评定等级 rating level

在有序标尺上的某个刻度，用于分类某一测量的标度。

注 1： 评定等级能使软件按照明确或隐含的要求进行分类（评定）（见 10.2）。

注 2： 相应的评定等级与质量的不同角度有关，如“用户”、“管理者”或“开发者”的角度。

2.1310

读 read

从存储设备或数据媒体访问数据。

参见： 破坏性读 destructive read（2.461）和非破坏性读 nondestructive read（2.1024）。

相对： 写 write（2.1858）。

2.1311

实地址 real address

虚拟存储系统的主存储器部分中的存储单元的地址。

相对： 虚拟地址 virtual address（2.1836）。

2.1312

实存储器 real storage

虚拟存储系统的主存储器部分。

相对： 虚拟存储器 virtual storage（2.1836）。

2.1313

实时的 real time

用于说明在外部过程发生的实际时间期间执行计算操作的系统或方式，目的是计算结果能以及时的方式用于控制、监控或响应外部过程。

参见： 会话的 conversational（2.348）、交互的 interactive（2.792）、中断 interrupt（2.811）和联机的 on-line（2.1045）。

相对： 批处理的 batch（2.128）。

2.1314

实型 real type

一种数据类型，它的成员能假定值为实数且能由实数算术运算，例如，加、减、乘、除和开方进行运算。

相对： 字符型 character type（2.201）、枚举型 enumeration type（2.558）、整型 integer type（2.784）和逻辑型 logical type（2.878）。

2.1315

接收 receive

对于由发送方实例传递来的激励的处理。

参见： 发送方 sender（2.1435）和接收方 receiver（2.1316）。

2.1316

接收方 receiver

处理由发送方对象传递来的激励的对象。

参见： 发送方 sender（2.1435）。

2.1317

接受 reception

由类目准备对信号的接收做出反应的一处声明。

2.1318

记录 record

作为一个单元来处理的有关数据项的集合。例如，在采购控制中，每个发票的数据能构成一记录。

参见：逻辑记录 logical record（2.876）。

2.1319

恢复 recovery

系统、程序、数据库或其他系统资源恢复至一种能执行所要求的功能的状况。

参见： 后向恢复 backward recovery（2.123）、检查点 checkpoint（2.204）和前向恢复 forward recovery（2.650）。

2.1320

递归 recursion

a) 软件模块调用它自己的过程。

参见： 同时递归 simultaneous recursion（2.1456）。

b) 以它自己的方式定义或生成一进程或数据结构的过程。

2.1321

递归的 recursive

a) 关于调用它自己的软件模块。

b) 关于以自己的方式定义或生成的进程或数据结构。

2.1322

递归例程 recursive routine

一种例程。它可以作为自己的子例程来使用，它直接调用它自己或被它所调用的另一子例程所调用。使用递归例程通常要求在某处（例如下推表中）保持其尚未完成的状态的记录。

2.1323

冗余 redundancy

在容错系统中，系统中存在辅助的部件执行像其他元素同样的功能以防止失败或从失败恢复。

参见： 活动冗余 active redundancy（2.34）、多样性 diversity（2.493）、同构冗余 homogeneous redundancy（2.707）和备用冗余 standby redundancy（2.1565）。

2.1324

可重入的 reenterable

见： 重入的 reentrant（2.1325）。

2.1325

重入的 reentrant

可重入的 reenterable

软件模块当作为另一个进程的一部分执行时仍可作为进程的一部分进入，仍可得到所希望的结果。

2.1326

重入点 reentry point

在软件模块中，模块跟随调用其他模块可重入的地方。

2.1327

基准 reference

a. 对模型元素的一种指称、

b. 在类目之中，便于对其他类目导航的一种命名位槽。与指针 pointer（2.1146）同义。

2.1328

精化，细分 refinement

表示对在某种程序中已作出详细规定的某一事物作更完备规格说明的一种联系。例如，设计类是对分析类的精化。

2.1329

回归测试 regression testing

系统或部件选择的重新测试，用以验证修改未引起不希望的有害效果，或证明修改后的系统或系统部件仍满足规定的需求。

2.1330

联系，关系 relationship

对模型元素之间的一种语义连接。联系的例子有关联和泛化。

2.1331

相对地址 relative address

一个地址，它必须由加一偏移量调整以确定要访问的存储单元的地址。

参见： 基地址 base address（2.124）、索引地址 indexed address（2.740）和自相关地址 self-relative address（2.1429）。

相对： 绝对地址 absolute address（2.9）。

2.1332

相对装入程序 relative loader

见： 重定位装入程序 relocating loader（2.1347）。

2.1333

发布 release

一项配置管理行为，它说明某配置项的一个特定版本已准备好用于特定的目的（例如发布测试产品）。

2.1334

可靠性 reliability

在规定时间间隔内和规定条件下，系统或部件执行所要求功能的能力。

参见： 可用性 availability（2.115）和平均失效时间间隔 MTBF（2.984）。

2.1335

可靠性评估 reliability assessment

确定现有系统或系统部件可靠性所达到的水平的过程。

2.1336

可靠性数据 reliability data

在软件生存周期中在选择的点上评价软件可靠性所需要的信息。例如可靠性模型中使用的错误数据和时间数据，程序属性（如复杂性），程序设计特性（如使用的开发技术及程序员的经验）。

2.1337

可靠性评价 reliability evaluation

见： 可靠性评估 reliability assessment（2.1335）。

2.1338

可靠性增长 reliability growth

从纠正软件故障而得到的软件可靠性的改进。

2.1339

可靠性模型 reliability model

预测、估计或评估可靠性所使用的模型。

参见： 可靠性评估 reliability assessment（2.1335）。

2.1340

可靠性值 reliability numeric

在规定时间间隔内和规定条件下，一配置项将实现所要求功能的概率。

2.1341

可重定位的 relocatable

能装入主内存任何部分的代码。起始地址由装入程序确定，然后，调整代码中的地址，以反映代码已经装入的存储单元。

参见： 可重定位装入程序 relocating loader（2.1347）。

2.1342

重定位地址 relocating address

当计算机程序包含被装入至内存的地址时，由装入程序调整的地址。

相对： 绝对地址 absolute address（2.9）。

2.1343

重定位[代]码 relocating code

代码包含由装入程序调整以反映代码装入的存储单元的地址。

相对： 绝对代码 absolute code（2.11）。

2.1344

可重定位机器[代]码 relocatable machine code

要求在计算机执行之前把相对地址翻译为绝对地址的机器语言代码。

相对： 绝对机器代码 absolute machine code（2.14）。

2.1345

重定位 relocate

从主内存的一部分移动机器代码至另一部分，并调整地址使代码在它的新的单元能执行。

2.1346

重定位汇编程序 relocating assembler

产生可重定位代码的汇编程序。

相对： 绝对汇编程序 absolute assembler（2.10）。

2.1347

重定位装入程序 relocating loader

相对装入程序 relative loader

将可重定位代码装入主内存并调整代码中的地址以反映代码已经装入的存储单元。

相对： 绝对装入程序 absolute loader（2.13）。

2.1348

重定位字典 relocation dictionary

目标模块或装入模块的一部分，它标识当重定位发生时必须调整的地址。

2.1349

重定位因素 relocation factor

见： 偏移量 offset（2.1043）(a)。

2.1350

远程批[处理]入口 remote batch entry

见： 远程作业入口 remote job entry（2.1351）。

2.1351

远程作业入口 remote job entry(RJE)

远程批处理入口 remote batch entry

作业通过远程输入设备提交，此输入设备通过数据链接连接至计算机。

2.1352

会合 rendezvous

在两个平行任务之间，当一个任务已调用另一任务的入口，并且后一任务也正在为前者执行一相应的接受语句时所出现的相互作用。

2.1353

可重复性 repeatability

见： 测试可重复性 test repeatability（2.1717）。

2.1354

重复性寻址 repetitive addressing

隐含寻址的一种方法，在此方法中，从计算机指令的操作字段可得到上一条执行的指令的操作数的地址。

相对： 向前一个寻址 one-ahead addressing（2.1048）。

2.1355

重演 replay

见： 逆向执行 reversible execetion（2.1390）。

2.1356

报告标准 report standard

描述工程和管理活动结果的特征的标准。

2.1357

储藏库 repository

一种存储对象模型、接口和实现的设施。

2.1358

表示标准 representation standard

描述工程和管理产品外观特征的标准。

2.1359

招标 request for proposal

需方使用的一份文件，它用来向潜在的投标人表示它要获得某特定系统、产品或服务的意图。

2.1360

要求的培训 required training

履行某个特定角色所要求的、由组织指定的培训。

2.1361

需求，要求 requirement

a) 用户为解决某一问题或达到某个目标所需要的条件或能力。

b) 系统或系统部件必须满足或处理的条件或能力以满足合同、标准、规格说明或其他正式的强制性文件的要求。

c) 如在 a、b 中的条件或能力的说明的表示。

参见： 设计需求 design requirement（2.451）、功能需求 functional requirement（2.667）、实现需求 implementation requirement（2.730）、接口需求 interface requirement（2.798）、性能需求 performance requirement（2.1133）和物理需求 physical requirement（2.1140）。

2.1362

需求标准 requirement standard

描述需求规格说明的特征的标准。

2.1363

需求分析 requirements analysis

a) 研究用户需要以得到系统、硬件或软件需求的定义的过程。

b) 研究和重新定义系统、硬件或软件需求的过程。

2.1364

需求审查 requirements inspection

见： 审查 inspection（2.764）。

2.1365

需求阶段 requirements phase

软件生存周期中的一个阶段，在此期间对软件产品的需求（如功能和性能方面的能力）进行定义并编制出相应的文档。

2.1366

需求评审 requirement review

把系统、硬件项或软件项的需求提交给项目个人、管理者、用户、客户或其他感兴趣的部门以便进行评论或审批的过程或会议。类型包括系统需求评审、软件需求评审。

相对： 代码评审 code review（2.224）、设计评审 design review（2.452）、形式合格评审 formal qualification review（2.646）和测试准备好评审 test readiness review（2.1717）。

2.1367

需求规格说明 requirements specification

规定系统或部件的需求的文档。典型地包括功能需求、性能需求、接口需求、设计需求和开发标准。

参见： 功能规格说明 functional specification（2.668）和性能规格说明 performance specification（2.1135）。

相对： 设计描述 design description（2.424）。

2.1368

需求规格说明语言 requirements specification language

具有特殊构造和验证协议的规格说明语言，它用于开发、分析和编制硬件或软件需求文件。

参见： 设计语言 design language（2.447）。

2.1369

需求验证 requirements verification

见： 验证 verification（2.1826）。

2.1370

营救点 rescue

见： 重启动点 restart point（2.1380）。

2.1371

保留字 reserved word

编程语言中的字，它的含义由语言的规则在一定的或全部上下文中固定，不能由程序员用作他用。例子包括：IF 、THEN、WKILE 等等。

2.1372

复位 reset

把变量、寄存器或其他存储单元设置回预先描述的状态。

参见： 清除 clear（2.213）、初始化 initialize（2.756）。

2.1373

常驻控制程序 resident control program

见： 核心 kernel（2.831）a）。

2.1374

残留控制 residual control

一种微程序技术，在此技术中，微指令字段中的含义取决于在辅助寄存器中的值。

参见： 两级编码 two-level encoding（2.1779）。

2.1375

资源分配 resource allocation

见： 计算机资源分配 computer resource allocation（2.286）。

2.1376

资源管理 resource management

用于开发产品或执行服务的工具的标识、估计、分配和监控。例子是估计。

2.1377

响应时间 response time

在至交互的计算机系统的调查或命令的结束与系统的响应的开始之间消逝的时间。

参见： 端口到端口的时间 port-to-port time（2.1148）、思考时间 think time（2.1729）和往返时间 turnaround time（2.1775）。

2.1378

职责 responsibility

对某一类目的一种契约或责任。

2.1379

重启动 restart

在失效后，使用在校核点的状态和结果记录，使计算机程序恢复执行。

2.1380

重启动点 restart point

营救点 rescue point

计算机程序中，在失效后执行能重启动的点。

2.1381

退役 retirement

a) 系统或部件从它的操作环境永久删除。

b) 删除可操作的系统或部件的支持。

参见：软件生存周期 software life cycle（2.1506）、系统生存周期 system life cycle（2.1658）。

2.1382

退役阶段 retirement phase

软件生存周期中的一个阶段。在此阶段内，对软件产品的支持被终止。

2.1383

回顾踪迹 retrospective trace

在计算机程序执行期间，从历史的数据记录产生的踪迹。注：这与普通的踪迹不同，这是在程序执行期间累积地产生的。

参见： 执行踪迹 execution trace（2.582）、子例程踪迹 subroutine trace（2.1620）、符号踪迹 symbolic trace（2.1639）和变量踪迹 variable trace（2.1823）。

2.1384

返回 return

a) 从软件模块到调用它的软件模块的控制传送。

参见： 返回码 return code（2.1385）。

b) 赋一值至调用模块可访问的参数。例如，赋值 25 至由调用模块使用的参数 AGE。

参见： 返回值 return value（2.1386）。

c) 执行在 a 中传送的计算机指令或进程。

2.1385

返回码 return code

用于影响在从被调用模块返回后调用模块执行的码。

2.1386

返回值 return value

由被调用模块赋值给参数由调用模块访问。

2.1387

可重用性，可复用性 reusability

软件模块或其他工作产品能由多于一个计算机程序或软件系统使用的程度。

参见： 通用性 generality（2.672）。

2.1388

可重用的，可复用的 reusable

软件模块或其他工作产品能由多于一个计算机程序或软件系统使用。

2.1389

重用，复用 reuse

对某一已有制品的使用。

2.1390

逆向执行 reverse execution

见： 可逆的执行 reversible execution（2.1391）。

2.1391

可逆执行 reversible execution

后向执行 backward execution

回放 playback

重演 replay

逆向执行 reverse execution

一种调试技术，在此种技术中，记录程序执行的历史，于是可在用户控制下向前或向后重演。

2.1392

评审 review

把工作产品或一组工作产品提交给项目个人、管理者、用户、客户或其他感兴趣的部门为了评论或批准的过程或会议。类型包括代码评审(code review)、设计评审(design review)、形式合格评审(formal qualification review)、需求评审（requirement review）和测试就绪评审（test readiness review）。

2.1393

风险 risk

遭受损失的可能性。

2.1394

风险管理 risk management （RM）

一种问题分析的手段，它采用风险概率分析，对某情况的风险进行权衡研究，以便更精确地了解所涉及的风险。风险管理包括风险的识别、分析、优先级排序和控制。

2.1395

风险管理计划 risk management plan

描述一个项目将要执行风险管理活动的计划的集合。

2.1396

远程作业入口 RJE

remote job entry 的缩略语。

2.1397

健壮性，稳健性 robustness

在存在无效输入或急迫的环境条件下，系统或部件其功能正确的程度。

参见： 容错 error tolerance（2.570）和故障容忍 fault tolerance（2.616）。

2.1398

角色 role

参与特别语境的某一实体的命名的特定行为。角色可以是静态的（如关联端）或动态的（如协作角色）。

2.1399

转入 roll in

从辅助存储器传送数据或计算机程序至主内存。

参见： 对换 swap（2.1633）。

相对： 转出 roll out（2.1400）。

2.1400

转出 roll out

从主内存传送数据或计算机程序至辅助存储器，目的是释放主内存为别的使用。

参见： 对换 swap（2.1633）。

相对： 转入 roll in（2.1399）。

2.1401

根编译程序 root compiler

一种编译程序，其输出是与机器无关的中间表示。当它与依赖于机器的代码生成程序组合时，就构成了完整的编译程序。

2.1402

例程 routine

由其他程序或子程序调用的子程序。

注：术语“例程（routine）”、“子程序（subprogram）”和“子例程（subroutine）”在不同的编程语言中，其定义和使用是不同的，前述的定义是建议的标准。

参见： 协同例程 coroutine（2.353）和子例程 subroutine（2.1619）。

2.1403

基于规则的语言 rule-based language

一种非过程语言，它允许用户说明一组规则并表达使用这些规则的疑问和问题。

参见： 声明语言 declarative language（2.417）。

2.1404

运行 run

a) 在软件工程中，单个，通常是连续的计算机程序的执行。

参见： 运行时 run time（2.1407）。

b) 执行计算机程序。

2.1405

运行态，运行方式 run mode

当计算机正自动地执行着存储在存储单元中的指令，从而被认为是正在履行其功能时的那种状态。

2.1406

运行流 run stream

见： 作业流 job stream（2.829）。

2.1407

运行时间 run time

a) 计算机程序开始执行的瞬间。

b) 计算机程序执行的时间间隔。

见： 执行时间 execution time（2.580）。

2.1408

运行时间 running time

见： 执行时间 execution time（2.580）。

2.1409

脚手架 scaffolding

构成只支持软件开发和测试的计算机程序和数据文件，但不打算包括在最后产品中。例如，虚拟的例程或文件、测试用例生成器、软件监控器、桩程序（stubs）。

参见： 编程支持环境 programming support environment（2.1253）。

2.1410

标度 scale

具有特性定义的一组值。

注：标度类型的例子有：与一组类别对应的标称标度、与一组有序刻度对应的序数标度，与一组等距的有序刻度对应的间隔标度，以及既有等距刻度，也具有绝对零度的比例标度。使用标称标度或序数标度的度量（体制）产生定性的数据，而使用间隔标度和比例标度的度量（体制）产生定量的数据。

2.1411

可扩缩性 scaliability

是指软件系统可以在不同规模、不同档次的硬件平台上运行的能力。

2.1412

场景 scenario

描绘行为的动作的一种特定序列。场景可用于描绘用况实例的交互或执行。

参见： 交互 interaction（2.790）。

2.1413

调度程序 scheduler

一种计算机程序，它通常作为操作系统的一部分，用以调度、启动和终止作业。

2.1414

概包 schema

在 MOF 语境中，一种与作为模型元素的容器类同的包。概包对应于 MOF 包。

相对： 元模型 metamodel（2.937）和包 package（2.1089）。

2.1415

规格说明变更通知 SCN

specification change notice 的缩略语。

2.1416

软件设计描述，软件设计文档 SDD

software design decription 或 software design document 的缩略语。

2.1417

软件开发计划 SDP

software development plan 的缩略语。

2.1418

系统设计评审 SDR

system design review 的缩略语。

2.1419

第二代语言 second generation language(2GL)

见： 汇编语言 assembly language（2.86）。

2.1420

安全[性]，保密[性] security

对计算机硬件、软件进行的保护，以防止其受到意外的或蓄意的存取、使用、修改、毁坏或泄密。安全性也涉及对人体、数据、通信以及计算机安装的物理保护。

2.1421

安全内核 security kernel

与安全性有关的关键性语句的一个小的、自含的集合，作为操作系统的特权部分、为了使用一程序或访问某数据，必须满足核心所规定的全部准则。

2.1422

播种 seeding

见： 故障播种 fault seeding（2.615）。

2.1423

程序段，存储段，分段 segment

a) 子系统或子系统之一，它们构成整个系统。例如，财务系统中的帐目可付段。

b) 在存储分配中，计算机程序的一自包含部分，它能执行而不需把整个程序放在主内存中。

参见： 页 page（2.1093）。

c) 作为一个单元存储或传送的数据的集合。

d) 在路径分析中，在两个连续的分支点之间的计算机程序序列。

e) 把系统、计算机程序或数据文件分为如 a)、b)或 c)这样的段。

2.1424

选择性择一构造 selective choice construct

见： 分支 branch（2.157）。

2.1425

选择性转储 selective dump

只指定存储单元区域的转储。

参见： 变更转储 change dump（2.196）、动态转储 dynamic dump（2.519）、内存转储 memory dump（2.928）、事后转储 postmortem dump（2.1153）、快照转储 snapshot dump（2.1425）和静态转储 static dump（2.1582）。

2.1426

选择性踪迹 selective trace

一种变量，它只引用可选的变量。

参见： 执行踪迹 execution trace（2.582）、回顾踪迹 retrospective trace（2.1383）、子例程踪迹 subroutine trace（2.1620）和符号踪迹 symbolic trace（2.1639）。

2.1427

自描述性 self-descriptiveness

系统或部件包含足够的信息以解释它的目标和特性的程度。

参见： 可维护性 maintainability（2.903）、可测试性 testability（2.1726）和可使用性 usability（2.1801）。

2.1428

自含[带]文档的 self-documented

关于源代码，它包含解释它的目标、操作和其他有用信息以了解和维护此代码的注释。

2.1429

自相对地址 self-relative address

一种地址，它必须加至指令的地址以得到要访问的存储单元的地址。

参见： 基地址 base address（2.124）、索引地址 index address（2.740）、偏移量 offset（2.1043）和相对地址 relative address（2.1331）、

2.1430

语义错误 semantic error

由在给定的语言中，符号或符号组与它们的含义的关系的不了解导致的错误。

相对： 句法错误 syntactic error（2.1641）。

2.1431

语义 semantics

在给定的语言中，符号或符号组与它们的含义之间的关系。

相对： 句法 syntax（2.1642）。

2.1432

语义变化点 semantic variation point

在元模型中，语义发生变化的点。此种点为解释元模型的语义有意提供一定程度的自由。

2.1433

信号量 semaphore

用于同步并发进程的一种共享变量，它通过指示某动作是否已完成或某事件是否已发生来同步。

2.1434

发送 send

对于由发送方实例到接收方实例的激励的传递。

参见： 发送方 sender（2.1435）和接收方 receiner（2.1316）。

2.1435

发送方 sender

将激励传递到接收方对象的对象。

相对： 接收方 receiver（2.1316）。

2.1436

高层经理 senior manager

在组织足够高的层次上的一种管理角色，他主要关注组织长期的生命力，而不是短期的项目和合同的事务和压力问题。一般而言，工程的高层经理要负责多个项目。

2.1437

顺序图 sequence diagram

一种表示对象之间按时间顺序排列的交互的图。特别是，此种图展现了参与交互的对象和所交换消息的顺序。与协作图不同的是，时序图虽包括时间顺序，但不包括对象关系。时序图能以一般的形式（描述所有可能的场景），或以实例形式（描述一个实际场景）存在。时序图与协作图所表达的信息类似，但展现的方式不同。

参见： 协作图 collaboration diagram（2.231）。

2.1438

顺序的 sequential

串行的 serial

两个或多个动作或事件以以下方式发生：一个必须在下一个开始之前完成。

参见接续的 consecutive（2.319）。

2.1439

顺序内聚度 sequential cohesion

一类内聚度，在其中，一由软件模块执行的任务的输出作为由此模块执行的另一任务的输入。

相对： 一致内聚度 coincidental cohesion（2.229）、功能内聚度 functional cohesion（2.661）、逻辑内聚度 logical cohesion（2.874）和过程内聚度 procedural cohesion（2.1179）。

2.1440

顺序构造 sequential construct

见： 串行构造 serial construct（2.1443）。

2.1441

顺序进程 sequential processes

以这样一种方式执行的进程：在下一个动作开始之前本动作必须结束。

相对： 并发进程 concurrent process（2.299）。

2.1442

串行的 serial

整体的分别的部分，例如字符的位用同一设施顺序地传送、发生或处理。

见： 顺序的 sequential（2.1438）。

相对： 并行 parallel（2.1102）a）。

2.1443

串行构造 serial construct

顺序构造 sequential construct

由顺序步不包括决断或循环组成的程序结构。

2.1444

设置时间 set-up time

系统或部件正准备某种特定的操作的时间周期。

参见： 忙时间 busy time（2.169）、停机时间 down time（2.503）、空闲时间 idle time（2.717）和开机时间 up time（2.1798）。

2.1445

服务 service

与软件有关的活动、工作或义务的实施，例如软件的开发、维护和操作。

2.1446

严重性 severity

见： 关键程度 criticality（2.372）。

2.1447

核心外层（外壳）程序 shell

在用户与计算机系统或程序之间提供接口的计算机程序或例程。

2.1448

副作用 side effect

进行的处理或活动，或得到的结果，它们与程序、子程序、或操作的主要功能相比是处于第二位的。

2.1449

信号 signal

对在实例之间所递异步激励的规格说明。信号可有参数。

2.1450

特征标记 signature

某一行为特征的名称和参数。特征标记可包括一个可选的返回参数。

2.1451

简单缓冲 simple buffering

在程序执行期间，把缓冲器分配给计算机程序的缓冲技术。

相对： 动态缓冲 dynamic buffering（2.517）。

2.1452

简单性 simplicity

系统或部件的设计和实现的简单和易于了解的程度。

相对： 复杂性 complexity（2.260）。

2.1453

仿真 simulation

a) 一个模型，当提供一组控制输入时，它的行为或操作像一给定的系统。

参见： 模拟 emulation（2.540）。

b) 开发或用如 a)中模型的过程。

2.1454

仿真器 simulator

一个设备、计算机程序或系统，当提供一组控制输入时，其行为或操作像一给定的系统。

参见： 模拟器 emulator（2.541）。

2.1455

同时的 simultaneous

两个或多个事件在同一瞬间发生。

相对： 并发 concurrent（2.298）。

2.1456

同时递归 simultaneous recursion

两个软件模块相互调用的情况。

2.1457

单地址指令 single-address instruction

见： 一地址指令 one-address inctruction（2.1047）。

2.1458

单一继承 single inheritance

其中每一类型都只可有一个超类型的泛化的一种语义变化。

相对： 多重继承 multiple inheritance（2.994）。

2.1459

单级编码 single-level encoding

一种微程序技术，其中，不同的微操作在微指令的相同字段编码为不同值。

相对： 两级编码 two-level encoding（2.1779）。

2.1460

单操作数指令 single-operand instruction

见： 一地址指令 one-address instruction（2.1047）。

2.1461

单步执行 single-step execution

见： 单步操作 single-step operation（2.1462）。

2.1462

单步操作 single-step operation

单步执行 single-step execution

步进操作 step-by-step operation

一种调试技术，在其中，单计算机指令或指令的一部分执行以响应外部信号。

2.1463

单值的 single-valued

当其多重性类型即上层属性设置为 1 时，修饰带有所定义多重性的一种模型元素。在任何时刻，术语单值都与属性、参数等所取的值的数目无关，原因是单值属性（例如，带有下界为零的多重性）可以没有值。

相对： 多值的 multi-valued（2.999）。

2.1464

规模估计 sizing

对软件系统或部件所要求的源程序的行数或计算机存储的总量的估计过程。

2.1465

从状态 slave state

见： 问题状态 problem state（2.1178）。

2.1466

快照转储 snapshot dump

一个或多个规定的存储区域的内容的动态转储。

参见： 变更转储 change dump（2.196）、动态转储 dynamic dump（2.519）、内存转储 memory dump（2.928）、事后转储 postmortem dump（2.1153）、有选择性转储 selective dump（2.1425）和静态转储 static dump（2.1582）。

2.1467

软差错 soft error

见： 瞬时差错 transient error（2.1767）。

2.1468

软失效 soft failure

允许具有部分操作能力的系统继续操作的失效。

相对： 硬失效 hard failure（2.689）。

2.1469

软件 software

与计算机系统的操作有关的计算机程序、规程和可能相关的文档。

参见： 应用软件 application software（2.71）、支持软件 support software（2.1632）和系统软件 system software（2.1667）。

相对： 硬件 hardware（2.690）。

2.1470

软件体系结构 software architecture

它是对系统整体结构设计的刻画，包括全局组织与控制结构，构件间通信、同步和数据访问的协议，设计元素间的功能分配、物理分布、设计元素集成、伸缩性和性能、设计选择等。

2.1471

软件基线审核 software baseline audit

为验证基线是否与描述基线的文档相符，对软件基线库的结构、内容和设施进行的一种检查。

2.1472

软件基线库 software baseline library

存储配置项及其有关记录的仓库的内容。

2.1473

软件构造版 software build

软件系统或成分的一个可运行的版本，它包含最终的软件系统或成分将要提供的能力的一个规定子集。

2.1474

软件能力评价 software capability evaluation

为识别承制方是否有资格从事某项软件工作或监控现有软件工作所采用的软件过程的状态，由经过培训的专业人员组成的团队对软件能力所作的鉴定。

2.1475

软件特性 software characteristic

软件固有的可能是偶然的品质、质量或特性（例如，功能、性能、属性、设计限制、状态数、行数或分支数）。

2.1476

软部件 software component

软件配置项中的一个明确的部分。

注：一个软部件含有软件的多个单元、也可以含有多个较低级的软部件。

2.1477

软件配置 software configuration

软件产品在不同时期的组合。该组合随着开发工作的进展而不断变化。

2.1478

软件配置控制委员会 software configuration control board

负责评价和批准（或不批准）对配置项提出的更改，并负责保证那些已批准的更改能得到实施的一种组。

2.1479

软件配置管理 software configuration management

见： 配置管理 configuration management（2.313）。

2.1480

软件配置管理项 software configuration management item

置于软件配置管理之下的软件配置项的各种有关项目。包括各类管理文档、评审记录和文档、软件文档、源码及其可执行码、运行所需的系统软件和支持软件、以及各种有关数据等。

2.1481

软件数据库 software database

存放运行软件系统内部公共数据的数据定义及其当前值的文件。

2.1482

软件设计描述 software design description(SDD)

a) 为促进分析、计划、实现和做决定建立的软件表示。软件设计描述用作一通信的软件设计信息的媒体并可能作为系统的蓝图或模型。

b) 为促进分析、计划、实现和做决定建立的软件表示。软件系统的蓝图或模型。SDD 用作为通信的软件设计信息的原始媒体。

2.1483

软件开发周期 software development cycle

从决定开发一个软件产品开始到产品交付结束的时间周期。这个周期典型地包括需求阶段、设计阶段、实现阶段、测试阶段，有时还包括安装和验收阶段。

相对： 软件生存周期 software life cycle（2.1506）。

注：a) 上列的阶段可以覆盖或重复执行，取决于所用的软件开发方法。

b) 此术语有时用于含义为更长的时间周期，或者当软件不再由开发者增强而结束的时间周期或即整个软件生存周期（software life cycle）。

2.1484

软件开发环境 software development environment

支持软件产品开发的软件系统，简称 SDE。它由软件工具和环境集成机制构成，前者用以支持软件开发的相关过程、活动和任务，后者为工具集成和软件的开发、维护及管理提供统一的支持。

2.1485

软件开发文件 software development file(SDF)

软件开发文件夹 software development folder

软件开发簿 software development notebook

单元开发文件夹 unit development folder

关于给定的软件单元或相关的单元集的开发的材料的集合。典型内容包括需求、设计、技术报告、代码清单、测试计划、测试结果、问题报告、进度和单元的注释。

2.1486

软件开发文件夹 software development folder

见： 软件开发文件 software development file（2.1485）。

2.1487

软件开发库 software development library

项目库 project library

程序支持库 program support library

存放与软件开发工作有关的计算机可读信息和人们可读信息的软件库。

相对：主库 master library(2.916)、产品库 production library(2.1217)、软件储藏库 software repository（2.1529）和系统库 system library（2.1657）。

2.1488

软件开发方法 software development method

软件开发过程所遵循的方法和步骤。它是规则、方法和工具的集成，既支持开发，也支持以后的演化过程。

2.1489

软件开发簿 software development notebook

软件开发文件 software development file

有关给定软件模块情况材料的集合，其内容通常包括与给定软件模块有关的需求、设计、技术报告、代码列表清单、测试计划、测试结果、问题报告、进度、注释等等。

参见： 项目簿 project notebook（2.1262）。

2.1490

软件开发计划 software development plan(SDP)

计算机程序开发计划 computer program development plan

为开发某一软件产品而做的项目计划。

2.1491

软件开发过程 software development process

把用户要求转化为软件产品的过程，此过程包括：把用户要求转换为软件需求，把软件需求转化为设计，用代码来实现设计，对代码进行测试，有时包括安装和验收。注：这些活动可以覆盖或重复执行。

参见： 增量开发 incremental development（2.738）、快速原型 rapid prototyping（2.1307）、螺旋模型 spiral model（2.1553）和瀑布模型 waterfall model（2.1844）。

2.1492

软件多样性 software diversity

一种软件开发技术，其中，两个或多个功能相同的部分由不同的程序员或程序员组用同一个规格说明进行开发，目的是提供出错检测，增加可靠性、附加的文档或减少影响结果的编程或编译错误的概率。

参见： 多样性 diversity（2.493）。

2.1493

软件文档 software documentation

以人们可读的形式出现的技术数据和信息。包括计算机列表和打印输出，它们描述或规定软件设计或细节，说明软件具备的能力，或为使用软件以便从软件系统得到所期望的结果而提供的操作指令。

参见：文档 documentation（2.496）、系统文档 system documentation（2.1653）和用户文档 user documentation（2.1809）。

2.1494

软件工程 software engineering

应用计算机科学理论和技术以及工程管理原则和方法，按预算和进度，实现满足用户要求的软件产品的定义、开发、发布和维护的工程或进行研究的学科。

2.1495

软件工程经济学 software engineering economics

从经济学的观点来研究、分析如何有效的开发、发布软件产品和支持用户使用这一产品的学科。

2.1496

软件工程环境 software engineering environment

用于执行软件工程的硬件、软件和固件。典型的元素包括：计算机装备、编译程序、汇编程序、操作系统、调试程序、模拟器、仿真器、测试工具、文档工具和数据库管理系统。

2.1497

软件工程组 software engineering group (SEG)

负责项目的软件开发和维护活动（即需求分析、设计、编码和测试）的人员（包括经理和技术人员）

的集合。软件工程组不包括执行软件有关工作的组，例如软件质量保证组，软件配置管理组和软件工程过程组。

2.1498

软件工程过程组 software engineering process group (SEPG)

负责组织所用软件过程的定义、维护和改进工作的专家组。在关键实践中，这个组一般称作“负责组织的软件过程活动的组”。

2.1499

软件工程人员 software engineering staff

执行项目的软件开发和维护活动的技术人员（例如：分析员、程序员和工程师），包括软件任务组长，但不包括经理。

2.1500

软件经验数据 software experience data

与软件的开发或使用有关的数据。这在开发软件模型、可靠性预测，或软件的其他定量描述中可能是有用的。

2.1501

软件特征 software feature

a) 区分软件项的特征（例如，性能、可移植性或功能）。

b) 由需求文档规定或隐含的软件特征（例如，功能、性能、属性或设计约束）。

2.1502

软件集成 software integration

将选定的软件成分集成在一起,旨在提供最终软件系统具备的一组能力或者规定的一组子能力的过程。

2.1503

软件项 software item

源代码、目标代码、作业控制代码、控制数据或这些项的集合。

2.1504

软件库管理员 software librarian

负责建立、管理和维护软件库的人员。

2.1505

软件库 software library

软件和有关的文档说明的一个受控制的集合。目的是有助于软件开发、使用或维护。类型包括主库、产品库、软件开发库、程序库、软件储藏仓和系统库。

参见： 程序库 program library（2.1236）。

2.1506

软件生存周期 software life cycle

当软件产品从构思开始至软件不再可用结束的时间周期。软件生存周期典型地包括：需求阶段、设计阶段、实现阶段、测试阶段、安装和验收阶段、操作和维护阶段有时还包括退役阶段（见图 15）。

注： 这些阶段可以覆盖或重复执行。

相对： 软件开发周期 software development cycle（2.1483）。

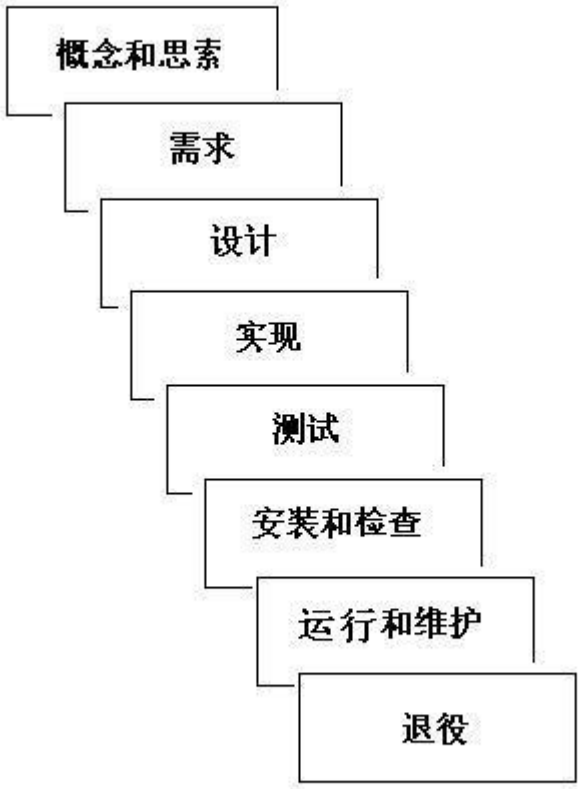

图15　软件生存周期示例

2.1507

软件维护　software maintenance

见：　维护 maintenance（2.905）。

2.1508

软件经理　software manager

在项目或组织层上对软件开发和（或）维护负直接责任的经理。

2.1509

软件监控程序　software monitor

和另一程序并行执行的软件工具，并对那个程序的执行情况提供详细的信息。

参见：　硬件监控程序 hardware monitor（2.693）和监控程序 monitor（2.981）。

2.1510

软件包　software package

一种完备并配有文档，供给有类同应用或功能要求的若干用户的程序集。

2.1511

软件计划　software plans

描述软件开发和（或）软件维护活动将如何进行的各类正式计划或非正式计划。例如软件开发计划、

软件质量保证计划、软件配置管理计划、软件测试计划、风险管理计划和过程改进计划。

2.1512

软件过程 software process

由组织或项目使用的，用以计划、管理、执行、监控、控制和改进其软件相关活动的过程或过程的集合。

2.1513

软件过程评估 software process assessment

为了确定组织当前软件过程的状态、组织面临的急需处理的软件过程的有关问题，并获得组织对软件过程改进的支持，由一个经过培训的软件专业人员组成的团队对软件过程所作的评价。

2.1514

软件过程描述 software process description

在项目定义的软件过程或组织的标准软件过程中被标识的主要软件过程成分的操作性定义。它通过文档以一种完备的、精确的、可验证的方式记载软件过程的需求、设计、行为或其他特征。

参见： 过程描述 process description（2.1195）。

2.1515

软件过程元素 software process element

描述软件过程的构成元素。每个过程元素包括一组妥善定义的、界定的、密切相关的任务（例如软件估计元素、软件设计元素、编码元素和同行评审元素）。过程元素的描述可以是待填入的模板、待完成的片段、待精炼的抽象概念、或者待修改的或已使用而无需修改的完整描述。

2.1516

软件过程改进计划 software process improvement plan

从软件过程评估建议中导出的一种计划，它标识为改进软件过程而将采取的具体措施，并且概述实施这些措施的计划。

2.1517

软件过程改进建议 software process improvement proposal

一个文档化的、对过程或过程有关项所作的、将提高软件过程能力和性能的更改建议。

参见： 措施建议 action proposal（2.27）。

2.1518

软件过程成熟度 software process maturity

一个具体过程被明确定义、管理、测量、控制以及有效的程度。成熟度隐含了能力上的增长潜力，并揭示组织的软件过程的丰富性和在全组织的所有项目中采用这些过程的一致性。

2.1519

软件过程有关文档 software process-related documents

预期未来项目在剪裁组织的标准软件过程时可能使用的实例文档或文档片段。实例文档可能包括诸如项目定义的软件过程、标准、规程、软件开发计划、测量计划和过程培训材料等。

2.1520

软件产品 software product

a) 指定交付给用户的计算机程序、规程和可能相关的文档和数据的完整集。

b) a)中的任一单独的项。

2.1521

软件项目 software project

一项需要协同工作的任务，它关注系统中软件成分的分析、说明、设计、开发、测试和（或）维护等工作及其相关文档。一个软件项目可以是建造硬件/软件系统的项目的一部分。

2.1522

软件质量 software quality

a) 软件产品中能满足给定需要的性质和特性的总体。例如，符合规格说明。

b) 软件具有所期望的各种属性的组合程度。

c) 顾客和用户觉得软件满足其综合期望的程度。

d) 确定软件在使用中将满足顾客预期要求的程度。

2.1523

软件质量保证 software quality assurance

见： 质量保证 quality assurance（2.1294）。

2.1524

软件质量目标 software quality goal

为软件工作产品确定的定量的质量目标。

2.1525

软件质量管理 software quality management

确定软件产品的质量目标，制定实现这些目标的计划，以及为了满足顾客和最终用户的需要和希望而监控和调整软件计划、软件工作产品、活动与质量目标的过程。

2.1526

软件质量度量 software quality metric

见： 质量度量 quality metric（2.1300）。

2.1527

软件有关组 software-related group

从事支持（但不直接负责）软件开发和（或）维护工作的软件工程人员（包括经理和技术人员）的集合。例如软件质量保证组和软件配置管理组。

2.1528

软件可靠性 software reliability

a) 在规定条件下，在规定的时间内软件不引起系统失效的概率。该概率是系统输入和系统使用的函数，也是软件中存在的缺陷的函数。系统输入将确定是否会遇到已存在的缺陷〈如果有缺陷存在的话〉。

b) 在规定的时间周期内所述条件下程序执行所要求的功能的能力。

2.1529

软件储藏库 software repository

为软件和相关的文档提供永久性的档案的软件库。

相对： 主库 master library（2.916）、产品库 product library（2.1217）、软件开发库 software development library（2.1487）和系统库 system library（2.1657）。

2.1530

软件需求 software requirement

为解决用户的问题或实现用户的目标，用户所需的软件必须满足的能力和条件。

2.1531

软件需求评审 software requirement review(SRR)

a) 一种对一个或多个软件配置项规定的需求的评审，它评价对系统需求的响应和解释以确定它们能否形成进一步的配置项的初步设计的满意的基础。

参见： 系统需求评审 system requirement review（2.1665）。

b) 任一软部件如a)中的评审。

2.1532

软件需求规格说明 software requirement specification(SRS)

软件和它的外部接口的基本需求（功能、性能、设计约束和属性）的文档。

2.1533

软件服务 software service

实施与软件产品有关的活动、工作或义务，比如软件开发、维护和运作。

2.1534

软件潜行分析 software sneak analysis

适用于软件的一种技术，它用以识别潜伏的（潜行的）逻辑控制路径或条件。这些路径或条件会禁止所期望进行的操作或引起不希望有的操作出现。

2.1535

软件规格说明评审 software specification review(SSR)

见： 软件需求评审 software requirement review（2.1531）。

2.1536

软件测试待查事件 software test incident

在软件测试执行期间发生的要求调查的任一事件。

2.1537

软件工具 software tool

一种计算机程序，用来帮助开发、测试、分析或维护一计算机程序或它的文件。例如，比较器、交叉引用生成器、反编译程序、驱动程序、编辑程序、流程图器、监控程序、测试用例生成器、定时分析器。

2.1538

软件单元 software unit

一段可分开编译的代码。

2.1539

软件工作产品 software work product

作为定义、维护或实施软件过程的一部分而生成的任何制品，包括过程说明、计划、规程、计算机程序和相关的文档等，无论是否打算将它们交付给顾客或者最终用户。

参见： 软件产品 software product（2.1520）。

2.1540

源地址 source address

要传送的数据所在的设备或存储单元的地址。

相对：目标地址 destination address（2.460）。

2.1541

源[代]码 source code

以适合于作为汇编程序、编译程序或其他转换程序输入的形式表示的计算机指令和数据定义。注：源程序由源代码构成。

相对： 目标代码 object code（2.1031）。

2.1542

源[代]码生成器 source code generator

见： 代码生成器 code generator（2.221）b）。

2.1543

源语言 source language

输入至机器辅助转换过程的语言。例如，用于写计算机程序的语言。

相对： 目标语言 target language（2.1676）。

2.1544

源程序 source program

必须被编译、汇编或其他转换的计算机程序以被计算机执行。

相对： 目标程序 target program（2.1678）。

2.1545

缺陷的特殊原因 special cause of a defect

在某种瞬态情况下所特有的而不是过程所固有的缺陷原因。特殊原因造成过程性能随机变化。

参见： 共同原因 common case(of a defect)（2.240）。

2.1546

特定地址 specific address

见： 绝对地址 absolute address（2.9）。

2.1547

特定码 specific code

见： 绝对码 absolute code（2.11）。

2.1548

规格说明，规约 specification

a) 以一种完全的、精确的、可验证的方法规定系统或部件的需求、设计、行为或其他特性的文件。并经常指明一过程，以确定给定的这些规定是否满足。

参见： 正式的规格说明 formal specification（2.648）、产品规格说明 product specification（2.1219）和需求规格说明 requirement specification（2.1367）。

b) 规范的明确的约定。

2.1549

规格说明变更通知 specification change notice(SCN)

在配置管理中所用的文档，它用以对规格说明的变更提出建议并加以传送和记录。

参见：配置控制 configuration control（2.307）、工程变更 engineering change（2.546）和修订通知 notice of revision（2.1028）。

2.1550

规格说明语言 specification language

一种语言，它通常是机器可处理的自然语言和形式语言的组合。用来表示系统或部件的需求、设计、行为或其他特征。例如，设计语言或需求规格说明语言。

相对： 编程语言 programming language（2.1252）和查询语言 query language（2.1303）。

2.1551

规格说明树 specification tree

描写给定的系统的所有规格说明并表示它们相互之间关系的图。

参见： 文档树 documentation tree（2.498）。

2.1552

规格说明验证 specification verification

见： 验证 verification（2.1826）。

2.1553

螺旋模型 spiral model

一种软件开发过程模型，在此种模型中，构成的活动，典型的为需求分析、初步和详细设计、编码、集成和测试是重复地执行的，直至软件完成。

参见： 增量开发 incremental development（2.738）和快速原型 rapid prototyping（2.1307）。

相对： 瀑布模型 waterfall model（2.1844）。

2.1554

假脱机 spool

读输入数据或写输出数据至辅助存储器或主内存，以备以后处理或输出，其目的是允许输入输出设

备与作业执行并发操作。这是从的缩略语词 SPOOL（Simultaneous Peripheral Output On Line－外设联机同时输出）导出的。

2.1555

假脱机程序 spooler

启动和控制假脱机的程序。

2.1556

软件需求评审 SRR

softwars requirement review 或 system requirement review 的缩略语。

2.1557

软件需求规格说明 SRS

software requirement specification 的缩略语。

2.1558

软件规格说明评审 SSR

software specification review 的缩略语。

见： 软件需求评审 Software Requirement Review（2.1531）。

2.1559

稳定性 stability

a) 在有干扰或破坏事件影响下仍能保持不变的能力。

b) 在干扰或破坏性事件之后返回到原始状态的能力。

2.1560

栈 stack

按后进先出方法进行存取的一个列表。

相对： 队列 queue（2.1304）。

2.1561

独立的，单立的 stand-alone

用于说明无需连接至其他部件就能执行它的功能的硬件或软件、例如，单机字处理系统。

2.1562

标准 standard

对于软件开发使用的和强制规定的统一的方法的强制要求，即在事实标准中的强制的约定和实践。

参见： 实践 practices（2.1156）。

2.1563

标准过程 standard process

在组织中用于指导一般过程建立的基本过程的可操作性定义。

注：一个标准过程描述可以应用于任何已定义的过程的功能过程元素，并描述这些过程元素之间的关系（如排序、接口等）（见已定义的过程）。

2.1564

标准实施器 standards enforcer

一种软件工具，它确定指定的开发标准是否得到遵循。标准可以包括模块大小、模块结构、注释的约定、某些语句形式的使用以及文件编制约定。

2.1565

备用冗余 standby redundancy

在容错系统中，在主要元素发生失效以前，一直不用的冗余元素的使用。

相对： 活动冗余 active redundancy（2.34）。

2.1566

备用时间 standby time

见： 空闲时间 idle time（2.717）。

2.1567

启动地址 starting address

在主内存中，计算机程序的第一条指令的地址。注：此地址可以是也可以不是程序的起点（origin），取决于是否有数据领先于第一条指令。

参见： 汇编的起点 assembled origin（2.81）和装入的起点 loaded origin（2.867）。

相对： 起点 origin（2.1079）。

2.1568

状态 state

a) 在系统、部件或模拟中已经存在的条件或方式。例如，航行器导航程序的预飞状态或给定的通道的输入状态。

b) 在给定的瞬间由定义系统、部件或模拟的特征的变量假定的值。

2.1569

状态数据 state data

定义测试单元的内部状态和用以建立状态或与已存在状态相对的数据。

2.1570

状态图 state diagram

描写系统或部件能假定的状态并表示从一种状态变更为另一个状态的事件或情况的图。

2.1571

状态机 state machine

规定如下状态的序列的一种行为：某一对象或交互在其生存线内，为对各事件作出响应所经历的（状态）以及该对象的响应和动作。

2.1572

状态转移图 state transition diagram

见： 状态图 state diagram（2.1570）。

2.1573

语句 statement

在编程语言中，定义数据、规定程序动作或指挥汇编程序或编译程序的有意义的表达式。

参见： 赋值语句 assignment statement（2.99）、控制语句 control statement（2.343）和声明 declaration（2.411）。

2.1574

工作说明书 statement of work

需方使用的一种文件，它用来叙述和规定按合同必须执行的任务。

2.1575

语句测试 statement testing

用以对执行计算机程序的每个语句的测试。

2.1576

静态的 static

无计算机程序执行发生的事件或过程。例如，静态分析、静态绑定。

相对： 动态的 dynamic（2.511）。

2.1577

静态分析 static analysis

基于它的格式、结构、内容或文档评价系统或部件的过程。

参见： 审查 inspection（2.764）和走查 walk-through（2.1843）。

相对： 动态分析 dynamic analysis（2.513）。

2.1578

静态分析程序 static analyzer

一种软件工具，它有助于分析计算机程序而无需执行该程序，例如语法检验程序、编译程序、交叉引用表生成程序、标准实施器以及流程图。

相对： 动态分析程序 dynamic analyzer（2.514）。

2.1579

静态绑定 static binding

在程序执行之前实现的，执行期间不加变更的绑定。

相对： 动态绑定 dynamic binding（2.515）。

2.1580

静态断点 static breakpoint

能在编译时设置的断点，例如，进入给定的例程。

参见： 代码断点 code breakpoint（2.220）、数据断点 data breakpoint（2.391）、跋断点 epilog breakpoint（2.559）、可编程的断点 programmable breakpoint（2.1249）和序断点 prolog breakpoint（2.1268）。

相对： 动态断点 dynamic breakpoint（2.516）。

2.1581

静态分类 static classification

其中的对象不可变更类目的泛化的一种语义变化。

相对： 动态分类 dynamic classification（2.518）。

2.1582

静态转储 static dump

在计算机程序执行前或执行后产生的转储。

参见： 变更转储 change dump（2.196）、内存转储 memory dump（2.928）、事后转储 postmortem dump（2.1153）、有选择性转储 selective dump（2.1425）和快照转储 snapshot dump（2.1466）。

相对： 动态转储 dynamic dump（2.4951）。

2.1583

静态差错 static error

独立于输入随时间变化的错误。

相对： 动态差错 dynamic error（2.520）。

2.1584

统计测试模型 statistical test model

一种模型，它把程序故障与输入数据集〈或多个数据集〉联系起来。模型也给出了这些故障将引起程序失效的概率。

2.1585

状态码 status code

条件码 condition code

用于指示计算机程序操作结果的码。例如，指示进位、溢出或奇偶错误的码。

2.1586

步进操作 step-by-step operation

见： 单步操作 single-step operation（2.1461）。

2.1587

逐步细化 stepwise refinement

一种软件开发技术，首先，广泛地定义数据和处理步骤，然后逐步详细。

参见： 以数据结构为中心的设计 data structure-centered design（2.405）、输入-处理-输出 input-processoutput（2.762）、模块分解 modular decomposition（2.973）、面向对象的设计 object-oriented（2.1037）、快速原型 rapid prototyping（2.1307）、结构设计 structured design（2.1607）、事务分析 transaction analysis（2.1760）和转换分析 transform analysis（2.1764）。

2.1588

衍型 stereotype

一种扩展元模型语义的建模元素的新类型。衍型必须基于元模型中现有的某些类型或类。衍型可扩展业已存在的类型和类的语义，但不可扩展其结构。某些衍型在 UML 中是预定义的，另一些可由用户定义。衍型是 UML 中三种可扩展性机制之一。

参见： 约束 constraint（2.323）和加标值 tagged value（2.1673）。

2.1589

激励 stimulus

信息从一个实例到另一实例的传递，例如激起一个信号或启用一个操作。接收信号通常认为是一个事件。

参见： 消息 massage（2.932）。

2.1590

停止 stop

终止计算机程序的执行。与暂停 halt（2.688）a）同义。

相对： 暂停 pause（2.1125）。

2.1591

存储分配 storage allocation

计算机资源分配的元素，由赋存储区域给规定的作业并执行相关的过程构成，例如，在主内存和辅助存储器之间传送数据以支持所做的赋值。

参见： 缓冲器 buffer（2.162）、邻接分配 contiguous allocation（2.329）、循环搜索 cyclic search（2.386）、内存紧缩 memory compaction（2.927）、覆盖 overlay（2.1085）、调页 paging（2.1101）和虚拟存储器 virtual storage（2.1840）。

2.1592

存储断点 storage breakpoint

见： 数据断点 data breakpoint（2.391）。

2.1593

存储容量 storage capacity

在给定的存储设备中能保持的项的最大数。通常，它用字（word）或字节（byte）来测量。

参见： 通道容量 channel capacity（2.198）和内存容量 memory capacity（2.926）。

2.1594

存储效率 storage efficiency

系统或部件执行指定的功能而具有可用的存储器的最小耗费的程度。

参见： 执行效率 execution efficiency（2.578）。

2.1595

存储 store

a） 在存储设备中存放数据或保持数据。

b） 复制计算机指令或数据，从寄存器至内存或从内存至外部存储器。

参见： 读取 fetch（2.622）和移动 move（2.983）。

相对： 装入 load（2.863）b）。

2.1596

直线代码 straight-line code

没有循环的计算机指令序列。

2.1597

直线编码 straight-line coding

一种编程技术，在此种技术中，由显式说明和填满在每一次循环执行时所引用的全部指令来避免循环。

参见： 展开 unwind（2.1796）。

2.1598

分层语言 stratified language

不能用作自身的元语言的语言。例子包括，FORTRAN、COBOL。

相对： 非分层语言 unstratified language（2.1794）。

2.1599

强度测试 stress testing

评价系统或部件在它规定的需求的限定或超出时情况的测试。

2.1600

串 string

实体（如字符或物理元素）的线性序列。

2.1601

强类型法 strong typing

一种程序设计语言特性，它要求对每个数据对象的数据类型都作出说明，并排除操作符施用于不适当的数据对象上的情况。因此，防止了不相容类型的数据对象的相互作用。

2.1602

结构性特征 structural feature

某一模型元素的一种静态特征，例如一个属性。

2.1603

结构性模型方面 structural model aspect

侧重于系统中各对象的结构的一种模型方面，它包括对象的类型、类、联系、属性和操作。

2.1604

结构测试 structural testing

白盒测试 glass-box testing

白盒测试 white-box testing

侧重于系统或部件内部机制的测试。类型包括分支测试、路径测试、语句测试。

相对： 功能测试 function testing（2.669）(a)。

2.1605

结构图 structure chart

层次图 hierarchy chart

程序结构图 program structure chart

一种图，它标识在系统或计算机程序中的模块、活动或其他实体并显示如何把较大的或更通用的实体分解为较小的、更特定的实体（见图16）。

注： 结果不必与在调用图中所示的相同。

相对： 调用图 call graph（2.177）。

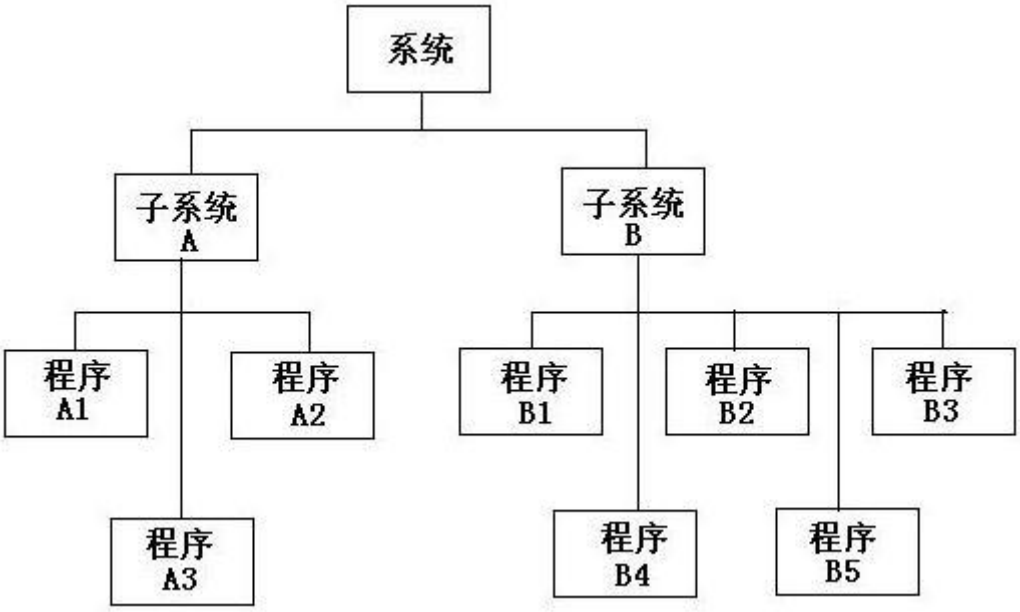

图16　结构图

2.1606

结构相搅　structure clash

在软件设计中，一个模型必须处理两个或多个具有不兼容数据结构的数据集的情况。

参见：　以数据结构为中心的设计 data structure －centered design（2.405）、次序抵触 order clash（2.1070）。

2.1607

结构化设计　structured design

a)　软件设计的任何遵守纪律的方法，它遵守基于例如模块化、自顶向下设计和数据、系统结构和处理步骤的逐步细化等原理的规定的规则。

参见：　以数据结构为中心的设计 data structure －centered design（2.405）、输入-处理-输出 input-processoutput（2.762）、模块分解 modular decomposition（2.973）、面向对象的设计 object-oriented（2.1037）、快速原型 rapid prototyping（2.1307）、逐步细化 stepwise refinement（2.1587）、事务分析 transaction analysis（2.1760）和转换分析 transform analysis（2.1764）。

b)　应用在 a)中的方法的结果。

2.1608

结构化程序　structured program

由一组基本的控制结构构造而成的程序。每一个控制结构有一个入口点和一个出口点。控制结构组典型地包括：由两条或多条指令组成的序列、两个或多个指令或指令序列的条件选择、一个指令或指令序列的重复执行。

参见：　结构化设计 structured design（2.1607）。

2.1609

结构化程序设计　structured programming

包括结构化设计和结构化程序开发中的结果的任何软件开发技术。

2.1610

结构化程序设计语言　structured programming language

一种程序设计语言，它提供了结构化程序的结构、命名、单入口－单出口序列、分支和循环并有利于结构化程序的开发工作。

参见： 块结构语言 block-structured language（2.146）。

2.1611

桩[模块] stub

a) 一种软件模块的框架或特殊目的的实现，它用于开发或测试调用它或依赖于它的模块。

b) 用来代替软件模块体的计算机程序语句，该模块是在别处定义或将在别处定义。

2.1612

子活动状态 subactivity state

在活动图中，一种表示执行具有持续期的非原子步骤的序列的状态。

2.1613

子类 subclass

在泛化关系中，对另一类即超类的特化。

参见： 泛化 generalization（2.674）。

相对：超类 superclass（2.1624）。

2.1614

子合同经理 subcontract manager

在主承制方的组织中直接负责控制和管理一个或多个子合同的经理。

2.1615

分包方 subcontractor

依据合同向合同当事人的一方提供系统、产品或服务的一个机构。

2.1616

子机[器]状态 submachine state

如下状态机中的一种状态：等效于某一组合状态，但其内容由另一状态机来描述。

2.1617

子包 subpackage

一种包含在另一包中的包。

2.1618

子程序 subprogram

一单独的、兼容的、可执行的计算机程序的部件。

注： 术语“例程（routine）”、“子程序（subprogram）”和“子例程（subroutine）”在不同编程语言中的定义是不同的，上述的定义已建议作为标准。

参见： 协同例程 coroutine（2.353）、主程序 main program（2.902）、例程 routine（2.1402）和子例程 subroutine（2.1619）。

2.1619

子例程 subroutine

返回控制至调用它的程序或子程序的例程。

注： 术语“例程（routine）”、“子程序（subprogram）”和“子例程（subroutine）”在不同编程语言中的定义是不同的，上述的定义已建议作为标准。

参见： 关式子例程 closed subroutine（2.216）和开式子例程 open routine（2.1052）。

相对： 协同例程 coroutine（2.353）。

2.1620

子例程踪迹 subroutine trace

调用踪迹 call trace

在计算机程序执行期间执行的全部的或所选择的子例程或函数调用的记录，它并可任选地包括传送

至或从每个子例程或函数返回的参数值。

参见：执行踪迹 execution trace（2.582）、回顾踪迹 retrospective trace（2.1383）、符号踪迹 symbolic trace（2.1639）和变量踪迹 variable trace（2.1823）。

2.1621

子状态 substate

作为某一组合状态一部分的状态。

参见： 并发子状态 concurrent substate（2.300）和互斥子状态 disjoint sunstate（2.490）。

2.1622

子系统 subsystem

在较大的系统中的二级或下级系统。

2.1623

子类型 subtype

数据类型的子集，它由限制数据类型的可能值的集得到。

注： 适用于子类型的操作与那些数据的原始的类型的操作相同。

参见： 导出类型 derived type（2.437）。

2.1624

超类 superclass

在泛化关系中，对另一类即子类的泛化。

参见： 泛化 generalization（2.674）。

相对： 子类 subclass（2.1613）。

2.1625

超类型 supertype

在泛化关系中，对另一类型即子类型的泛化。

参见： 泛化 generalization（2.674）。

相对： 子类型 subtype（2.1623）。

2.1626

监督程序 supervisor

见： 监督程序 supervisory program（2.1628）。

2.1627

监督状态 supervisor state

执行状态 execution state

主状态 master state

特权状态 privileged state

在计算机系统的操作中，管理程序执行的状态。此状态，通常比应用程序的执行状态有更高的优先权。

相对： 问题状态 problem state（2.1178）。

2.1628

监督程序 supervisory program

控制程序 control program

执行程序 executive program

监督程序 supervisor

一种计算机程序，它通常是操作系统的一部分，控制其他计算机程序的执行并且调节数据处理系统中的工作流程。

参见： 监督状态 supervisor state（2.1627）。

2.1629

供方 supplier

按照所签的合同向需方提供系统、产品或服务的一个机构（是合同当事人、生产者、卖方、批发商的同义词）。

注：需方可以指定它的机构中的某一部门做为供方。

2.1630

支持 support

操作系统或部件实现它的原始要求和任何后续对这些要求的修改所需的活动集。例如，软件和硬件维护、用户培训。

参见：软件生存周期 software life cycle(2.1506)和系统生存周期 system life cycle(2.1658)。

2.1631

支持手册 support manual

维护手册 maintenance manual

提供操作系统和它的部件在整个生存周期内服务和维护所需信息的文档。典型地描述它的构成系统或部件的硬件和软件以及对它服务、修理或重编程。

参见： 诊断手册 diagnostic manual（2.476）、安装手册 installation manual（2.766）、操作员手册 operator manual(2.1068)、程序员手册 programmer manual(2.1250)和用户手册 user manual（2.1813）。

2.1632

支持软件 support software

辅助其他软件开发或维护的软件。例如，编译程序、装入程序和其他实用程序。

参见： 系统软件 system software（2.1667）。

相对： 应用软件 application software（2.71）。

2.1633

对换 swap

a) 两个存储区域的内容的交换，它通常是主存储区域与辅助存储区域交换。

参见： 转入 roll in（2.1399）和转出 roll out（2.1400）。

b) 执行如 a 中的交换。

2.1634

泳道 swimlane

在活动图上，一种用于组织动作职责的分割。在典型情形下，泳道对应于业务模型中的组织单元。

参见： 分区 partition（2.1113）。

2.1635

符号表 symbol table

表示程序符号与它们相应的地址、值和其他属性的表。

2.1636

符号地址 symbolic address

用名或标记表示的地址，对其访问时必须转换为设备或存储单元的绝对地址。

相对： 绝对地址 absolute address（2.9）。

2.1637

符号执行 symbolic execution

一种软件分析技术，其中，程序的执行是使用符号，例如变量名，而不是输入数据的真实值来模拟，而程序输出则表示成包含这些符号的逻辑或数学表达式。

2.1638

符号语言 symbolic language

一种编码语言，它用对人方便的符号而不是用机器语言表示操作和地址。例子是汇编语言、高级语言。

相对： 机器语言 machine language（2.890）。

2.1639

符号踪迹 symbolic trace

当计算机程序用符号而不是用输入数据的实际值运行时遇到的源语句或分支出口的记录。

参见： 执行踪迹 execution trace（2.582）、回顾踪迹 retrospective trace（2.1383）、子例程踪迹 subroutine trace（2.1620）和变量踪迹 variable trace（2.1823）。

2.1640

同步状态 synch state

在状态机中，用于对其中的并发区域同步化的一种顶点。

2.1641

句法差错 syntactic error

句法差错 syntax error

为语言定义的结构的或文法规则的违反。例如，在 FORTRAN 中用语句 B+C=A，而不是正确的 A=B+C。

相对： 语义差错 semantic error（2.1430）。

2.1642

句法 syntax

在语言中，定义符号如何组合以形成字、词组、表达式和其他允许的结构的、文法的规则。

相对： 语义 semantics（2.1431）。

2.1643

句法差错 syntax error

见： 句法差错 syntactic error（2.1641）。

2.1644

综合地址 synthetic address

见： 生成的地址 generated address（2.675）。

2.1645

系统 system

a) 人、机器和方法的集合，用来实现一组规定的功能。

b) 一个完整的整体，它由种类不同的、相互作用的、专门的结构和子功能部件所组成。

c) 由某些相互作用或相互依赖关系联合起来的小组或子系统。它可执行多种职能，但是作为一个单位而发挥其作用。

2.1646

系统分析 systems analysis

对实在的或计划好和系统所进行的一种系统性调研，其目的是确定该系统的信息要求和过程，它们之间的相互关系，以及与其他系统的关系。

2.1647

系统体系结构 system architecture

系统各部件之间的结构和关系。系统体系结构也可以包括系统和它的运行环境之间的界面。

2.1648

系统描述 system description

由系统设计形成的文档，它定义对系统的组织、实质特性以及硬件和软件要求。

2.1649

系统设计 system design

a) 为系统定义硬件和软件结构、部件、模块、接口及数据，以满足规定的系统需求的过程。

b) 系统设计过程的结果。

2.1650

系统设计评审 system design review(SDR)

一种评审行为，它评价系统需求已经分配给配置项、产生分配的系统工程过程、下一阶段要做的计划、制造考虑和产品工程计划等的方式。

参见： 关键设计评审 criitical design review（2.365）和概要设备评审 preliminary design review（2.1163）。

2.1651

系统开发 system development

一种过程，它通常包括需求分析、系统设计、实现、文件编制和质量保证。

2.1652

系统开发周期 system development cycle

从决定开发系统开始至系统交付给用户结束的时间周期。注：此术语有时用于意味着更长的时间周期，或者至系统不再增强结束或整个系统生存周期。

参见： 软件开发周期 software development cycle（2.1506）。

相对： 系统生存周期 system life cycle（2.1658）。

2.1653

系统文档 system documentation

表达系统的需求、设计思想、设计细节、能力、限制，以及其他特性的文档。

2.1654

系统流程图 system flowchart

见： 流程图 flowchart（2.637）。

2.1655

集成 integration

由系统部件到整个系统的逐级装配。

2.1656

系统工程组 system engineering group

负责以下工作的人员的集合（包括经理和技术人员）：规定系统需求；将系统需求分配给硬件、软件和其他成分；规定硬件、软件和其他成分间的接口；以及为保证与相应规范的符合性而监控这些成分的设计和开发。

2.1657

系统库 system library

包含系统常驻软件的软件库。可供访问和使用，或通过引用而合并到其他程序中去。例如，宏库。

相对：主库 master library(2.916)、产品库 production library(2.1217)、软件开发库 software development library（2.1487）和软件储藏库 software repository（2.1529）。

2.1658

系统生存周期 system life cycle

从系统构思开始至系统不再可用结束的时间周期。

参见： 系统开发周期 system development cycle（2.1652）和软件生存周期 software life cycle（2.1506）。

2.1659

系统维护 system maintenance

对某一系统的修改，其目的是排除故障、提高性能，或者使之适应于变化了的环境或变化了的要求。

2.1660

系统模型 system model

在计算机性能评价中，系统描述在系统中的工作负载与性能测量之间关系的表示。

参见： 工作负载模型 workload model（2.1857）。

2.1661

系统特征文件（系统轮廓） system profile

在计算机性能评价中所用的一组测量,它描述计算机系统中每个主要资源忙的时间部分被资源可用时间除。

2.1662

系统可靠性 system reliability

包括全部硬件和软件子系统在内的某个系统,在规定的环境及时间里正确执行所要求的任务或使命的概率。

参见： 运行可靠性 operational reliability（2.1063）和软件可靠性 software reliability（2.1528）。

2.1663

系统需求 system requirement

为了满足用户解决问题需要的条件或能力，系统或系统成分必须满足或具有的条件或能力。

2.1664

分配给软件的系统需求 system requirements allocated to software

拟由系统的软件成分实现的系统需求的子集。分配需求是软件开发计划的主要输入。软件需求分析将精心加工和细化分配需求，并产生文档化的软件需求。简称**分配需求**。

2.1665

系统需求评审 system requirement review(SRR)

一种评审行为，它评价为系统定义的需求是否完整和适当、评价产生这些需求的系统工程过程、评定系统工程试作的结果和评价系统工程计划。

参见： 软件需求评审 software requirement review（2.1531）。

2.1666

系统资源图 system resource chart

见： 框图 block diagram（2.145）。

2.1667

系统软件 system software

设计以帮助计算机系统和相关的程序操作和维护的软件。例如，操作系统、编译程序、实用程序。

参见： 支持软件 support software（2.1632）。

相对： 应用软件 application software（2.71）。

2.1668

系统支持 system support

为使用和改进已实现的系统所必须的服务和物资的后续提供。

2.1669

系统测试 system testing

在完整的、集成的系统上的测试行为，它用以评价系统与规定的需求的遵从性。

参见： 部件测试 component testing（2.264）、集成测试 integration testing（2.788）、接口测试 interface testing（2.800）和单元测试 unit testing（2.1792）。

2.1670

系统确认 system validation

见： 确认 validation（2.1819）。

2.1671

系统验证 system verification

见： 验证 verification（2.1826）。

2.1672

表[格] table

a) 数据的一个阵列，其中每一项均可借助于一个或多个变元显式地予以标识。

b) 数据的一个集合，其中每一项可由标号、位置，或按某些其他方法唯一地标识。

2.1673

加标值 tagged value

对某一性质的显式定义的“名称-值”对。加标值中的名称称为标志。某些标志在 UML 中是预定义的、另一些可由用户定义。加标值是 UML 中三种可扩展性机制之一。

参见： 约束 constrsint（2.323）和衍型 stereotype（2.1588）。

2.1674

目标能力 target capability

由过程能力确定发起人鉴定的过程能力，它代表对成功地实施特定要求的一种可接受的过程风险。

2.1675

目标计算机 target computer

拟运行交付软件的计算机。

参见： 宿主计算机 host computer（2.709）。

2.1676

目标语言 target language

目标语言 object language

一种语言，它表示从机器辅助转换过程的输出。例如，由汇编程序或编译程序输出的语言。

相对： 源语言 source language（2.1543）。

2.1677

目标机 target machine

a) 打算在其上运行程序的计算机。

相对： 宿主机 host machine（2.710）a）。

b) 正由另一台计算机仿真的计算机。

相对： 宿主机 host machine（2.710）b）。

2.1678

目标程序 target program

见： 目标程序 object program（2.1039）。

2.1679

任务 task

a) 由操作系统的管理程序作为基本工作单元对待的指令序列。

b) 在软件设计中，能与其他软部件并行操作的软部件。

2.1680

任务启动会议 task kick-off meeting

在项目任务开始时举行的会议，目的使有关人员为有效地执行任务的各项活动做好准备，。

2.1681

任务组长 task leader

从事某一特定任务的技术团队的领导人，他对任务负技术责任，并对任务的工作人员进行技术指导。

2.1682

分类学 taxonomy

分割知识体并定义其片断之间关系的方案，它用于分类和了解知识体。

2.1683

团队 team

一组人,这些人通常来自各个不同的但相关的组,受指派去执行组织或项目的一个妥善定义的职能。团队成员可以是兼职的，而且可有其他主要职责。

2.1684

技术管理 technical management

为计划、组织和控制工程功能，技术的和管理的资源的应用。

2.1685

技术需求 technical requirements

描述软件必须做些什么及其运行约束条件的需求，例如功能、性能、接口和质量需求。

2.1686

技术标准 technical standard

一种标准，它描述在创建产品或执行服务中应用积累的技术的或管理技巧和方法的特征的标准。

2.1687

工艺规程 techniques

技术的和管理的规程，它帮助软件开发过程的评价和改进。

2.1688

模板 template

见： 参数化元素 parameterized element（2.1106）。

2.1689

暂时内聚度 temporal cohesion

一类内聚度，其中，由软件模块执行的任务是在所有程序执行的具体阶段都要求的。例如，一包含所有程序的初始化任务的模块。

相对： 一致内聚度 coincidental cohesion（2.229）、通信内聚度 communication cohesion（2.246）、功能内聚度 functional cohesion（2.661）、逻辑内聚度 logical cohesion（2.874）、过程内聚度 procedural cohesion（2.1179）和顺序内聚度 sequential cohesion（2.1439）。

2.1690

终止构造 termination construct

一种程序结构，它导致暂停（halt）或退出（exit）。

2.1691

终止证明 termination proof

正确性证明中的一项内容，它证明在全部规定的输入条件下，程序将终止。

2.1692

测试 test

a) 一种活动，在此活动中，系统或部件在一定的条件下执行，观察或记录其结果，对系统或部件的某些方面进行评价。

b) 实施如在 a)中的活动。

c) 一个或多个测试用例集。

d) 一个或多个测试规程集。

e) 一个或多个测试用例和规程集。

2.1693

测试性 testability

a) 系统或成分便于建立测试准则并进行测试以确定是否满足测试准则的程度。

b) 使用允许建立测试准则并进行测试以确定是否满足测试准则方面的措辞来描述需求的程度。

2.1694

测试台 test bed

a) 测试环境，其中包括测试系统或部件所必须的硬件、探测工具、模拟程序、软件工具以及其他支持软件。

b) 在测试系统或系统的部件时所必需的全部测试用例的汇集。

2.1695

测试用例 test case

a) 为具体的目标（例如，为练习具体的程序路径或验证对特定需求的遵循性）而开发的一组测试输入、执行条件和预料的结果。

b) 对于测试项，规定输入、预料的结果和一组执行条件的文档。

参见： 测试用例生成器 test case generator（2.1696）和测试用例规格说明 test case specification（2.1697）。

2.1696

测试用例生成器 test case generator

测试数据生成器 test data generator

测试生成器 test generator

一种软件工具，它接受输入源代码、测试准则、规格说明或数据结构定义、用这些输入以产生测试输入数据、有时，还确定预料的结果。

2.1697

测试用例规格说明 test case specification

测试描述 test description

测试规格说明 test specification

对于要测试的项，规定测试输入、执行条件和预料的结果的文档。

参见： 测试待查事件报告 test incident report（2.1708）、测试项传输报告 test item transmittal report（2.1710）、测试日志 test log（2.1711）、测试计划 test plan（2.1714）、测试规程 test procedure（2.1715）和测试报告 test report（2.1719）。

2.1698

测试覆盖（范围） test coverage

给定的测试或测试集，对于给定的系统和部件实现所有规定的需求的程度。

2.1699

测试准则 test criteria

系统或部件为通过给定的测试必须满足的准则。

参见： 验收准则 acceptance criteria（2.18）和通过 / 失败准则 pass-fail criteria（2.1116）。

2.1700

测试数据 test data

用来测试系统或系统部件的数据。

参见： 测试用例 test case（2.1695）。

2.1701

测试数据生成器 test data generator

见： 测试用例生成器 test case generator（2.1696）。

参见： 自动测试生成器 automated test generator（2.111）。

2.1702

测试描述 test description

见： 测试用例规格说明 test case specification（2.1697）。

2.1703

测试设计 test design

一种文档，它规定对软件特性或软件特性与标识相关测试的组合的详细测试方法。

2.1704

测试文档集 test documentation

描述系统或部件的测试计划或测试结果的文档。类型包括：测量用例规格说明、测试事件报告、测试日志、测试计划、测试规程和测试报告。

2.1705

测试驱动程序 test driver

测试装具 test harness

一种软件模块，它用于引用测试下的模块并常常提供测试输入、控制和监控执行并报告测试结果。

2.1706

测试生成器 test generator

见： 测试用例生成器 test case generator（2.1696）。

2.1707

测试装具 test harness

见： 测试驱动程序 test driver（2.1705）。

2.1708

测试待查事件报告 test incident report

描述在测试期间发生要求进一步调查的事件的文档。

参见： 测试用例规格说明 test case specification（2.1696）、测试项传输报告 test item transmittal report（2.1710）、测试日志 test log（2.1711）、测试计划 test plan（2.1714）、测试规程 test procedure（2.1715）和测试报告 test report（2.1719）。

2.1709

测试项 test item

是测试目标的软件项。

2.1710

测试项传输报告 test item transmittal report

标识为测试传送的一个或多个项的文档。它包括当前状态和位置信息。

参见： 测试用例规格说明 test case specification（2.1696）、测试待查事件报告 test incident report（2.1708）、测试日志 test log（2.1711）、测试计划 test plan（2.1714）、测试规程 test procedure（2.1715）和测试报告 test report（2.1719）。

2.1711

测试日志 test log

按年月日对全部测试活动有关细节所做的记录。

参见： 测试用例规格说明 test case specification（2.1696）、测试待查事件报告 test incident report（2.1708）、测试项传输报告 test item transmittal report（2.1710）、测试计划 test plan（2.1714）、测试规程 test procedure（2.1715）和测试报告 test report（2.1719）。

2.1712

测试目标 test objective

在规定的条件下要度量的软件特性的标识集，它由把实际行为与在软件文档中描述的所要求的行为进行比较来度量。

2.1713

测试阶段 test phase

软件生存周期中的一段时间，在此期间对软件产品的部件进行评价且进行集成。并评价软件产品以确定需求是否已得到满足。

2.1714

测试计划 test plan

a) 描述要进行的测试活动的范围、方法、资源和进度的文档。它标识测试项、要测试的特性、测试任务、谁做每一个任务和任务意外事故的风险计划。

b) 描述为测试系统或部件必须遵循的技术和管理方法的文档。典型包括：标识要测试的项、要执行的任务、响应、进度和测试活动要求的资源。

参见： 测试用例规格说明 test case specification（2.1696）、测试待查事件报告 test incident report（2.1708）、测试项传输报告 test item transmittal report（2.1710）、测试日志 test log（2.1711）、测试规程 test procedure（2.1715）和测试报告 test report（2.1719）。

2.1715

测试规程 test procedure

测试规程规格说明 test procedure specification

测试脚本 test script

a) 对于给定的测试用例的设置、执行和评价测试结果的详细指令。

b) 包含如 a 中一组相关指令的文档。

c) 规定执行测试的动作序列的文档。

参见： 测试用例规格说明 test case specification（2.1696）、测试待查事件报告 test incident report（2.1708）、测试项传输报告 test item transmittal report（2.1710）、测试日志 test log（2.1711）、测试计划 test plan（2.1714）和测试报告 test report（2.1719）。

2.1716

测试规程规格说明 test procedure specification

见： 测试规程 test procedure（2.1715）。

2.1717

测试就绪评审 test readiness review

a) 对于一个或多个配置项评价初始测试结果的评审行为、以验证对于每个配置项的测试规程是完整的、遵循测试计划和描述且满足测试需求、并验证项目准备好进行配置项的正式测试。

b) 对于任何硬件或软部件像在 a)中那样评审。

相对： 代码评审 code review（2.224）、形式合格评审 formal qualification review（2.646）、设计评审 design review（2.452）和需求评审 requirement review（2.1366）。

2.1718

测试可重复性 test repeatability

测试的一种属性。指明每次进行的测试时，产生相同的结果。

2.1719

测试报告 test report

测试总结报告 test summary report

描述对系统或部件进行测试产生的行为及结果的文件。

参见： 测试用例规格说明 test case specification（2.1696）、测试待查事件报告 test incident report（2.1708）、测试项传输报告 test item transmittal report（2.1710）、测试日志 test log（2.1711）、测试计划 test plan（2.1714）和测试规程 test procedure（2.1715）。

2.1720

测试脚本 test script

见： 测试规程 test prpcedure（2.1715）。

2.1721

测试集体系结构 test set architecture

在直接反应测试目标的层次分解的测试用例集之间的嵌套关系。

2.1722

测试规格说明 test specification

见： 测试用例规格说明 test case specification（2.1697）。

2.1723

测试总结报告 test summary report

总结测试活动和结果的文件。它也包含相应测试项的评价。

参见： 测试用例规格说明 test case specification（2.1696）、测试待查事件报告 test incident report（2.1708）、测试项传输报告 test item transmittal report（2.1710）、测试日志 test log（2.1711）、测试计划 test plan（2.1714）、测试规程 test procedure（2.1715）和测试报告 test report（2.1719）。

2.1724

测试单元 test unit

一个或多个计算机程序模块与相关的控制数据（例如，表）、用法规程和操作规程一起的集合，这些集合满足下列条件：a)所有模块来自单个计算机程序、b)在集中至少有一个新的或更改的模块未完成单元测试、c)模块与它相关的数据和规程是测试过程的唯一目标。

2.1725

测试有效性 test-validity

完成测试规定目标的程度。

2.1726

可测试性 testability

a) 系统或部件帮助测试准则和测试性能建立的程度，以确定那些准则是否已满足。

b) 需求以允许测试准则和测试性能建立说明的程度，它用以确定那些准则是否已满足。

2.1727

测试 testing

a) 在规定的条件下操作系统或部件、观察或记录结果并对系统或部件的某些方面作评价的过程。

b) 分析软件项以检测在存在的和要求的条件之间的区别（即，隐错）以评价软件项的特征。

参见： 验收测试 acceptance testing（2.19）、基准测试 benchmark（2.134）、检查 checkout（2.203）、部件测试 component testing（2.264）、开发测试 development testing（2.468）、动态分析 dynamic analysis（2.513）、正式测试 formal testing（2.649）、功能测试 functional testing（2.669）、非正式测试 informal testing（2.749）、集成测试 integration testing（2.788）、接口测试 interface testing（2.800）、回送测试 loopback testing（2.884）、变异测试 mutation testing（2.1001）、运行测试 operational testing（2.1065）、性能测试 performance testing（2.1135）、合格性测试 qualification testing（2.1291）、回归测试 regression testing（2.1329）、强度测试 stress testing（2.1599）、结构测试 structural testing（2.1604）、系统测试 system testing（2.1669条）和单元测试 unit testing（2.1792）。

2.1728

文本编辑程序 text editor

编辑程序 editor

一种计算机程序，它通常作为字处理系统的一部分，允许用户输入、替换和浏览文本。

2.1729

思考时间 think time

由交互系统产生的提示符或消息的结束至用户响应开始之间消逝的时间。

参见： 端口至端口时间 port-to-port time（2.1148）、响应时间 response time（2.1377）和往返时间 turnaround time（2.1775）。

2.1730

第三代语言 third generation language(3GL)

见： 高级语言 high order language（2.702）。

2.1731

系统颠簸 thrashing

计算机系统正耗费大部分或全部资源用于开销操纵(例如数据在主和辅助存储器之间对换而不是做计算）的一种状态。

2.1732

[控制的]线程 thread [of control]

贯穿某一程序、动态模型或其他控制流中表示的一种单一执行路径。也是作为轻量级进程实现的主动对象的一种衍型。

参见： 进程 process（2.1183）。

2.1733

三地址指令 three-address instruction

包含三地址字段的计算机指令。例如，一指令加单元 A 和 B 的内容并放结果至单元 C 中。

相对： 一地址指令 one-address instruction（2.1047）、两地址指令 two-address instruction（2.1777）、四地址指令 four-address instruction(2.652)和零地址指令 zero-address instruction（2.1859）。

2.1734

三加一地址指令 three-plus-one address instruction

包含四个地址字段的计算机指令，其第四个地址字段包含下一条要执行的指令的地址。例如，一条指令加单元 A 和 B 的内容，把结果放至单元 C，然后执行在单元 D 中的指令。

相对： 一加一地址指令 one-plus-one address instruction（2.1050）、二加一地址指令 two-plus-one address instruction(2.1780)和四加一地址指令 four-plus-one address instruction（2.653）。

2.1735

吞吐量 throughput

对计算机系统或部件在给定的时间周期内规定执行的工作总量的度量。例如，每天的作业数。

参见： 往返时间 turnaround time（2.1775）和工作负载模型 workload model（2.1857）。

2.1736

排列图 tier chart

见： 调用图 call graph（2.177）。

2.1737

用时事件 time event

一种指称自进入当前状态起所经历时间的事件。

参见： 事件 event（2.573）。

2.1738

时间表达式 time expression

一种决定时间的绝对值或相对值的表达式。

2.1739

超时 time out

a) 当达到预先确定的消逝的时间总数而未发生预料的事件时产生的条件。例如，若在规定的时间周期内未接收到用户的输入，引起联机进程终止的条件。

b) a 中条件的实现。

2.1740

分时 time sharing

一种操作方式，它用程序的交叉执行，允许两个或多个用户在同一计算机系统上并发执行计算机程序。

注： 分时可以用时间片、基于优先权的中断或其他调度方法实现。

2.1741

时间分片 time slicing

一种操作方式，其中的两个或多个进程，每一个都赋予一小的、在同一处理器上固定的连续处理时间，而处理器以轮转方式执行，每一个执行为它分配的时间，直至全部完成。

2.1742

计时 timing

估计或测量软件系统或部件要求的总执行时间的过程。

相对： 规模估计 sizing（2.1464）。

2.1743

计时分析程序 timing analyzer

一种软件工具，它用于估计或度量计算机程序的执行时间或部分计算机程序的执行时间。这可通过求每条路径中指令的执行时间之和得到，或由在程序中的规定点处插入探头并度量探头之间的执行时间而得到。

2.1744

容错 (error)tolerance

系统在各种异常条件下提供继续操作的能力。

2.1745

工具 tool

a) 见： 软件工具 software tool（2.1537）。

b) 用于分析软件或其性能的执行时间的软件。

2.1746

自顶向下的 top-down

用于说明从层次的最高级部件处开始，逐步推进到较低级的活动。例如，自顶向下设计、自顶向下程序设计和自顶向下测试。

参见： 关键部分优先 critical piece first（2.368）。

相对： 自底向上 bottom-up（2.153）。

2.1747

自顶向下设计 top-down design

由整体到局部逐级细化的设计过程，即先标出系统的主要部件，并把它们分解为较低级成分，然后重复进行直到不能（或不必）再分解为止。

相对： 自底向上设计 bottom-up design（2.154）。

2.1748

自顶向下测试 top-down testing

通过对较低级部件进行模拟的办法来从顶到底逐步地检查按层次方式所构造的程序的过程。

2.1749

顶级 top level

指称包容层次结构中最顶的包的一种包的衍型。顶级衍型在命名空间向外“看”时，定义查阅名称的外层界限。例如：顶级子系统表示子系统包容层次结构的顶部。

2.1750

完全正确性 total correctness

在正确性证明中，指出程序的输出断言是它的输入断言和处理步骤的合乎逻辑的结果，并且在全部规定的输入条件下程序能终止。

相对： 部分正确性 partial correctness（2.1111）。

2.1751

追踪【动】，踪迹 trace

a) 计算机程序执行的记录，它显示执行的指令的顺序、变量的名和值或两者。类型包括执行踪迹、回顾的踪迹、子例程踪迹、符号踪迹、变量踪迹。

b) 如像 a 产生的记录。

c) 建立在两个或多个开发过程的产品之间的关系。例如，建立在给定的需求和实现此需求的设计元素之间的关系。

2.1752

可追踪性 traceability

a) 在两个或多个开发过程的产品，特别是相互之间有前任与后任或主、次关系的产品之间，能建立关系的程度。例如，给定的软件部件的需求和设计之间匹配的程度。

参见： 一致性 consistency（2.320）。

b) 在软件开发产品中每个元素为建立它存在的理由的程度。例如，在泡图中每个元素引用满足需求的程度。

2.1753

可追踪性矩阵 traceability matrix

记录两个或多个开发过程的产品之间的关系的矩阵。例如，记录给定的软部件的需求和设计之间的关系的矩阵。

2.1754

追踪程序 tracer

用于追踪的软件工具。

2.1755

尾部 trailer

放在文件或消息的结尾的标识或控制信息。

相对： 头部 header（2.695）b）。

2.1756

尾随判定 trailing decision

在循环体之后执行的循环控制。

参见： “直至”语句 UNTIL（2.1795）。

相对： 先导判定 leading decision（2.808）。

2.1757

培训组 training group

负责为组织协调和安排培训活动的人员的集合（包括经理和职员）。这个组准备和讲授大多数的培训课程，并协调其他培训工具的使用。

2.1758

培训大纲 training program

重点阐述组织的培训需求的一组相关元素。它包括组织的培训计划、培训教材、培训的开发、培训的执行、培训设施、培训的评价以及培训记录的维护。

2.1759

事务 transaction

在软件工程中，一数据元素、控制元素、信号、事件、或状态的变更，它引起、触发或启动一动作或动作序列。

2.1760

事务分析 transaction analysis

以事务为中心设计 transaction-centered design

一种软件开发技术，在这种技术中，系统的结构来自系统要求处理的事务的分析。

参见： 以数据结构为中心的设计 data structure-centered design（2.405）、输入-处理-输出 input-process-output（2.762）、模块分解 modular decomposition（2.973）、面向对象的设计 object-oriented（2.1037）、快速原型 rapid prototyping（2.1307）、逐步细化 stepwise refinement（2.1587）、结构设计 structured design（2.1607）和转换分析 transform analysis（2.1764）。

2.1761

以事务为中心设计 transaction-centered design

见： 事务分析 transaction analysis（2.1760）。

2.1762

事务矩阵 transaction matrix

标识对数据库访问的可能的要求和每次要求在数据库中相关的信息类别或元素的矩阵。

2.1763

传送，转移 transfer

a) 从一个地方送数据而在另外地方接收它。

b) 由一个进程放弃控制而假定由另一个恢复控制，或者期待返回（见 call）或不期待返回（见 jump）。

2.1764

转换分析 transform analysis

转换分析 transformation analysis

转换中心设计 transform-centered design

一种软件开发技术，在此技术中，系统的结构来自对通过系统的数据流和在数据上必须执行的转换的分析。

参见： 以数据结构为中心的设计 data structure-centered design（2.405）、输入-处理-输出 input-process-output（2.762）、模块分解 modular decomposition（2.973）、面向对象的设计 object-oriented（2.1037）、快速原型 rapid prototyping（2.1307）、逐步细化 stepwise refinement（2.1587）、结构设计 structured design（2.1607）和转换分析 transform analysis（2.1764）。

参见： 以数据结构为中心的设计 data structure-centered design（2.405）、输入-处理-输出 input-process-output（2.762）、模块分解 modular decomposition（2.973）、面向对象的设计 object-oriented（2.1037）、快速原型 rapid prototyping（2.1307）、逐步细化 stepwise refinement（2.1587）和结构设计 structured design（2.1607）。

2.1765

以转换为中心的设计 transform-centered design

见：转换分析 transform analysis（2.1764）。

2.1766

转换分析 transformation analysis

见：转换分析 transform analysis（2.1764）。

2.1767

瞬态差错 transient error

发生一次或在不可预测的间隔发生的错误。

参见： 间歇故障 intermittent fault（2.803）和随机失效 random failure（2.1306）。

2.1768

瞬时对象 transient object

一种仅在创建它的进程或线程的执行期间存在的对象。

2.1769

转移 transition

两个状态之间的一种关系，它指明当规定的事件出现，且规定的条件得到满足时，一个处于第一个状态的对象将执行某些特定动作并进入第二状态。在这样的状态变更中，称此转移受到激发。

2.1770

翻译程序，转换程序 translator

一种计算机程序，它把一种语言表达的语句序列变换为另一种语言表达的等价的语句序列。

参见： 汇编程序 assembler（2.84）和编译程序 compiler（2.254）。

2.1771

可运输性 transportability

见： 可移植性 portability（2.1149）。

2.1772

陷入 trap

a) 条件转移至异常或中断处理例程，常常由硬件激活，带有转移发生记录的位置。

b) 执行在 a 中的操作。

2.1773

树 tree

由通过分支互相连接的结点组成的抽象的层次结构，其中：a）每个分支把一个结点连接到一个直属的下级结点、b）有唯一称为根的一个结点，它不附属于任何其他结点、（c）根之外的每个结点只直接从属于另一个结点。

2.1774

测试就绪评审 TRR

test readiness review 的缩略语。

2.1775

往返时间 turnaround time

在作业的提交至批处理系统和完全的输出返回之间消逝的时间。

参见： 端口至端口时间 port-to-port time（2.1148）、响应时间 response time（2.1377）和思考时间 think time（2.1729）。

2.1776

交钥匙的 turnkey

用于说明硬件或软件以完整的、可操作的状态交付。

2.1777

二地址指令 two-address instruction

双操作数指令 double-operand instruction

包含两个地址字段的计算机指令。例如，加 A 单元的内容至 B 单元内容上。

相对： 一地址指令 one-address instruction（2.1047）、三地址指令 three-address instruction（2.1733）、四地址指令 four-address instruction（2.652）和零地址指令 zero-address instruction（2.1859）。

2.1778

两级地址 two-level address

一种间接地址，它规定包含所希望的操作数的地址的存储单元。

参见： n级地址 n-level address（2.1007）。

2.1779

两级编码 two-level encoding

一种微程序技术，在此种技术中，不同的微操作同时编码在微指令的同一字段中，且一个执行依赖于在微指令内部或外部的另一字段的值。

参见： 残留控制 residual control（2.1374）。

相对： 单级编码 single-level encoding（2.1459）。

2.1780

二加一地址指令 two-plus-one address instruction

包含三个地址字段，第三个包含下一条要执行的指令的地址的计算机指令。例如，把A单元的内容加至B单元的内容上，然后执行在C单元的指令。

相对： 一加一地址指令 one-plus-one address instruction（2.1050）、三加一地址指令 three-plus-one address instruction（2.1734）和四加一地址指令 four-plus-one address instruction（2.653）。

2.1781

类型 type

见： 数据类型 data type（2.4083）。

2.1782

类型表达式 type expression

一种求出对一个或多个类型的引用的表达式。

2.1783

单元开发文件夹 UDF

unit development folder的缩略语。

2.1784

无条件分支 unconditional branch

见： 无条件跳转 unconditional jump（2.1785）。

2.1785

无条件跳转 unconditional jump

不管执行的条件而发生的转移。

相对：conditional jump 条件跳转 （2.304）。

2.1786

下溢异常 underflow exception

当算术操作的结果太小，无法用指定接收它的存储单元表示时发生的异常。

参见： 寻址异常 addressing exception（2.50）、数据异常 data exception（2.395）、操作异常 operation exception（2.1059）、溢出异常 overflow exception（2.1082）和保护异常 peotection exception（2.1274）。

2.1787

无向图 undirected graph

在节点的相互连接中无隐含方向的图。

相对： 定向图 directed graph（2.485）。

2.1788

未解释的 uninterpreted

用于说明对于其实现未由UML规定的某一或某些类型的一种占位符。每一未解释值都有一个对应的串表示。

2.1789

单元 unit

a) 在计算机软部件设计中规定的一独立的可测试的元素。

b) 计算机程序的逻辑独立的部分。

c) 不可分为其他部件的软部件。

d) 见： 测试单元 test unit（2.1724）。

注： 术语“模块（module）”、“部件（component）”和“单元（unit）”常常可以交互使用或在不同的情况下相互可以定义为子元素，取决于上下文。这些术语的关系尚未标准化。

2.1790

单元开发文件夹 unit development folder

见： 软件开发文件 software development file（2.1485）。

2.1791

单元需求文档集 unit requirements documentation

对于测试单元，设置功能、接口、性能和设计约束的文档。

2.1792

单元测试 unit testing

独立的硬件或软件单元或相关单元组的测试。

参见： 部件测试 component testing（2.264）、集成测试 integration testing（2.788）、接口测试 interface testing（2.800）和系统测试 system testing（2.1669）。

2.1793

解除紧缩 unpack

从紧缩的数据，恢复一个或多个数据项的原始形式。

相对： 紧缩 pack（2.1088）。

2.1794

非分层语言 unstratified language

能用作其他元语言的语言。例如，英语、德语。

相对： 分层语言 stratified language（2.1598）。

2.1795

“直至”语句 UNTIL

后测试迭代 post-tested iteration

单入口、单出口循环，在此循环中，控制在循环体后执行（见图17）。

参见： 尾随判定 trailing decision（2.1756）。

相对： 闭合循环 closed loop（2.215）和 “当”语句 WHILE（2.1847）。

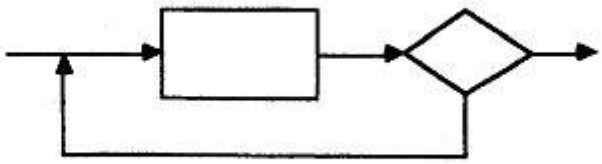

图17 UNTIL结构

2.1796

展开 unwind

在程序设计中，显式地且充分地说明在循环的多次执行中引用的所有指令。

参见：直线编码 straightline coding（2.1596）。

2.1797

开机的 up

用于说明可操作的正在服务中的系统或部件，这样的系统是忙的或空闲的。

参见： 忙 busy（2.168）和空闲的 idle（2.716）。

相对： 关机的 down（2.502）。

2.1798

开机时间 up time

系统或部件是可操作的和正在服务中的时间周期。即，忙时间和空闲时间之和。

参见： 忙时间 busy time（2.169）、空闲时间 idle time（2.717）、平均失效时间间隔 mean time between failures（2.920）和设置时间 set-up time（2.1444）。

相对： 关机时间 down time（2.503）。

2.1799

向上兼容的 upward compatible

关于与后一个或它自己的更复杂的版本兼容的硬件或软件。例如，能处理由它自己的后续版本创建的文件的程序。

相对： 向下兼容的 downward compatible（2.504）。

2.1800

向上压缩 upward compression

在软件设计中，去模块化的一种方式，在此方式中，下级模块复制至上级模块的体中。

相对： 侧向压缩 lateral compression（2.841）和向下压缩 downward compression（2.505）。

2.1801

易用性，可使用性 usability

方便用户学习操作、为系统或部件准备输入和解释系统或部件的输出。

2.1802

可使用性测试（适用性测试） usability test(fitness-for-use test)

为确定已实现的系统是否达到用户确定的功能目标所作的一种测试。

2.1803

用法 usage

一种依赖关系，其中一个元素（客户）要求另一元素（供方）在场，以使其正确行使职能或实现。

2.1804

用况[类] use case[class]

对一系列动作（包括变体）的规格说明，系统（或其他实体）能够执行它或者与原先的施动者进行交互。

参见： 用况实例 use case instance（2.1806）。

2.1805

用况图 use case diagram

一种显示系统中的各施动者与用况之间的关系的图。

2.1806

用况实例 use case instance

用况中规定的动作序列的执行。用况的一个实例。

参见： 用况类 use case class（2.1804）。

2.1807

用况模型 use case model

一种借助用况来描述系统的功能要求的模型。

2.1808

用户 user

使用可操作的系统完成一项特定的功能的个人或机构。(可以是买主或需方的同义词。)

2.1809

用户文档集 user documentation

一套文档，它为使用系统或部件以期获得所希望结果的最终用户提供系统指令方面的信息。例如，用户手册。

参见： 数据输入表 data input sheet (2.401) 和用户手册 user manual (2.1813)。

2.1810

用户友好 user friendly

关于以方便用户使用作为设计的主要目标的计算机系统、设备、程序或文档。

2.1811

用户指南 user guide

见：用户手册 user manual (2.1813)。

2.1812

用户界面，用户接口 user interface

允许信息在个人用户与计算机系统的硬件或软件部件间传送的接口。

2.1813

用户手册 user manual

用户指南 user guide

描述使用系统或部件以得到希望的结果所必需的信息的文档。典型地描述的是系统或部件的功能、限定、选项、允许的输入、预料的输出、可能的错误消息和特定的指令。

注： 当在操作计算机系统（安装磁带等）的人和为一定的目的使用系统的人之间有区分时，用户手册和操作员手册有区分。

参见： 数据输入表 data input sheet (2.401)、诊断手册 diagnostic manual (2.476)、安装手册 installation manual (2.766)、操作员手册 operator manual (2.1068)、程序员手册 programming manual (2.1250) 和支持手册 support manual (2.1631)。

2.1814

用户状态 user state

见： 问题状态 problem state (2.1178)。

2.1815

公（实）用程序 utility

一种软件工具，其设计的目的是用于频繁使用的支持功能。例如，复制磁带的程序。

2.1816

公（实）用软件 utility software

计算机程序或例行程序，其设计目的是为其他应用软件、操作系统或系统用户提供他们所要求的某些通用支持功能。

2.1817

利用率 utilization

在计算机性能评价中，表示系统或部件忙的时间总数被可用的时间总数除的比率。

参见：忙时间 busy time (2.169)、空闲时间 idle time (2.717) 和开机时间 up time (2.1798)。

2.1818

验证和确认 V&V

verification and validation 的缩略语。

2.1819

确认 validation

在开发过程期间或结束时对系统或部件进行评价，通过检查和提供客观证据，以确定它是否满足特定预期用途的需求的过程。

相对： 验证 verification（2.1826）。

注 1： 在设计和开发中，确认关系到检查产品是否符合用户要求的过程、

注 2： 确认一般是在规定的操作条件下对最终产品进行的。在早期阶段，这样做是必要的、

注 3： “确认过的”一词用来表示相应的状况、

注 4： 如果有几种不同的预期用途，可进行多种确认。

注 5： 特定预期用途的需求通常是指需求规格说明或合同中规定的需求。

2.1820

值 value

某一类型域中的一种元素。

2.1821

值踪迹 value trace

见： 变量踪迹 variable trace（2.1823）。

2.1822

变量 variable

值能变更的数量或数据项。例如，变量 当前时间（current-time）。

参见： 全局变量 global variable（2.681）和局部变量 local variable（2.871）。

相对： 常量 constant（2.321）。

2.1823

变量踪迹 variable trace

数据流踪迹 data-flow trace

数据踪迹 data trace

值踪迹 value trace

在计算机程序执行期间访问的或变更的变量的名和值的记录。

参见： 执行踪迹 execution trace（2.582）、回顾踪迹 retrospective trace（2.1383）、子例程踪迹 subroutine trace（2.1620）和符号踪迹 symbolic trace（2.1639）。

2.1824

变体 variant

在容错系统中，从软件的多样性的应用程序导出的版本。

2.1825

版本描述文件 VDD

Version Description Document 的缩略语。

2.1826

验证 verification

a) 评价系统或部件，以确定软件开发周期中的一个给定阶段的产品是否满足在阶段的开始确立的需求的过程。

相对： 确认 validation（2.1819）。

b) 程序正确性的形式证明。见： 正确性的证明 proof of correctness（2.1270）。

注 1： 在设计和开发中，验证是指对某项指定活动的结果进行检查的过程，以确定该活动是否符合明确的需求。

注 2： “验证过的”一词用来表示相应的状况。

2.1827

验证和确认 verification and validation(V&V)

确定系统或部件的需求是否完成和正确,每一开发阶段的产品是否实现在上一阶段规定的需求或条件，以及最后的系统或部件是否依从规定的需求的过程。

参见： 独立的验证和确认 independent verification and validation（2.739）。

2.1828

验证系统 verification system

见： 自动验证系统 automated verification system（2.112）。

2.1829

版本 version

a) 与计算机软件配置项的完全编纂或重编纂相关的计算机软件配置项的初始发行或再发行。

b) 作为与对以前的发行发出变更页导致的修订不同的文件的初始发行或完全再发行。

参见：配置控制 configuration control（2.307）和版本描述文件 version description document（2.1830）。

2.1830

版本描述文件 version description document(VDD)

伴随并标识系统或部件的给定的版本的文件。典型的内容包括系统或部件的详细目录、在此版本中变更的标识和版本描述的安装和操作信息。

2.1831

顶点 vertex

在状态机中，某一转移的一种源或目标。顶点可是一个状态或伪状态。

参见： 状态 state（2.1568）和伪状态 pseudo state（2.1283）。

2.1832

纵向微指令 vertical microinstruction

规定实现机器语言指令所需的操作序列之一的微指令。

注： 纵向的微指令相对是较短的，12至24位，所以称为“垂直的”，因为实现单个机器语言指令所要求的这样的指令序列，正常地是在页上垂直地列出的。

相对： 对角微指令 diagonal microinstruction（2.477）、横向微指令 horizontal microinstruction（2.708）。

2.1833

视图 view

某一些模型的一种如下投影：从一定的透视或有利位置观看，且略去与此透视无关的实体。

2.1834

视图元素 view element

对模型元素的某一汇集的一种文本的与图形的投影。

2.1835

视图投影 view projection

模型元素在视图元素上的一种投影。视图投影为视图元素提供位置和式样。

2.1836

虚拟地址 virtual address

在虚拟存储系统中,赋给一辅助存储单元的地址,它允许此存储单元像主存储器部分那样进行访问。

相对： 实地址 real address（2.1311）。

2.1837

虚拟机 virtual machine

对计算机及其有关设备进行功能模拟的设备。

2.1838

虚拟存储器 virtual memory

见： 虚拟存储器 virtual storage（2.1840）。

2.1839

虚拟空间 virtual space

a）计算机制图中的坐标空间，此空间中的每一单元由用户定义的坐标系表示。

b）用户程序运行的一个逻辑地址空间.

2.1840

虚拟存储器 virtual storage

多级存储器 multilevel storage

虚拟存储器 virtual memory

一种存储分配技术，其中的辅助存储器像主存储器那样能寻址。部分用户程序和数据放在辅助存储器中，操作系统在需要时自动把它们对换进入主存储器或对换出去。

参见： 虚拟地址 virtual address（2.1836）和调页 paging（2.1101）b）。

相对： 实存储器 real storage（2.1312）。

2.1841

可见性 visibility

一种枚举，其值（公用的，受保护的或私用的）指称该值引用的模型元素，在环绕的命名空间之外如何可以看到。

2.1842

放弃书 waiver

一个所写的授权以接受一配置项或其他指定的项,这些项在生产期间或交付审查后发现偏离规定的需求，但认为仍是可用的或经某种批准的方法重加工后可用。

参见： 配置控制 configuration control（2.307）。

相对： 偏离 deviation（2.473）和工程变更 engineering change（2.546）。

2.1843

走查 walk-through

一种静态分析技术或评审过程，在此过程中，设计者或程序员引导开发组的成员通读已书写的设计或编码，其地成员负责提出问题并对有关技术、风格、可能的错误、是否违背开发标准等方面进行评论。

2.1844

瀑布模型 waterfall model

一种软件开发过程的模型，其组成活动典型地按概念阶段、需求阶段、设计阶段、实现阶段、测试阶段和安装及验收阶段按这样的顺序执行：可能有重叠但只有小的或没有重复。

相对： 增量开发 incremental development（2.738）、快速原型 rapid prototyping（2.1307）和螺旋模型 spiral model（2.1553）。

2.1845

磨损失效周期 wearout-failure period

系统或部件生存周期中的周期，在此期间由于恶化，硬件的失效以递增的速度发生。

参见： 浴盆曲线 bathtub curve（2.129）。

相对： 定失效率周期 constant-failure period（2.322）和早期失效周期 early-failure period（2.526）。

2.1846

妥善定义的过程 well-defined process

包含就绪准则、输入、进行工作的标准和规程、验证机制（例如同行评审）、输出和完成准则的过程。

参见： 有效过程 fffective process（2.533）。

2.1847

“当”语句 WHILE

预测试迭代 pre-tested iteration

一种单入口、单出口的循环控制语句，其中循环控制在循环体之前执行（见图18）。

参见： 先导判定 leading desision（2.843）。

相对： 闭循环 closed loop（2.215）和“直至”语句 UNTIL（2.795）。

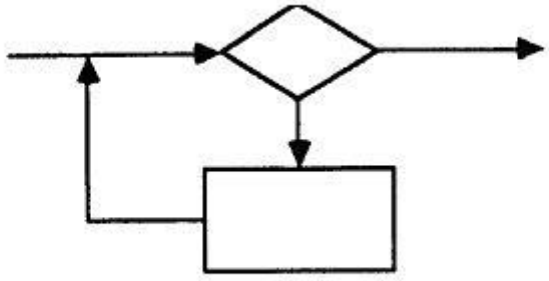

图18 WHILE结构

2.1848

白盒 white box

见： 白盒 glass box（2.677）。

2.1849

白盒测试 white-box testing

见： 结构测试 structural testing（2.1604）。

2.1850

字 word

计算机字 computer word

a) 在给定计算机中，作为一个单元存储、寻址、发送和操作的位或字符的序列。

b) 一计算机存储器的元素，它能容纳如a中的位或字符序列。

c) 在某些语言中具有意义作为一实体考虑的位或字符序列。例如，在计算机语言中的保留字。

参见： 二进制数字（位） binary digit(bit)（2.138）、字节 byte（2.170）。

2.1851

工作产品 work product

与执行过程相关的人工制品。

注：工作产品可以是由过程所使用的，产生的或更改的。

2.1852

工作区 working area

见： 工作空间 working space（2.1854）。

2.1853

工作集 working set

在存储分配的调页方法中，在程序执行的任何点最适合驻留在主内存中的一组页。

2.1854

工作空间 working space

工作区 working area

工作存储器 working storage

赋给计算机程序为数据暂时存储的主内存部分。

2.1855

工作存储器 working storage

见： 工作空间 working space（2.1854）。

2.1856

[工作]负载 workload

典型地在给定的计算机系统上运行的任务的混合。主要特征包括输入输出需求、计算的总数和种类和要求的计算机资源。

参见： 工作负载模型 workload model（2.1857）。

2.1857

[工作]负载模型 workload model

一种用在计算机性能评价中的模型,它为计算机系统中预期的或实际的工作负载规定资源利用和性能测量。

参见： 系统模型 system model（2.1660）。

2.1858

写 write

在存储设备或数据媒体上记录数据。

相对： 读 read（2.1310）。

2.1859

零地址指令 zero-address instruction

未包含地址字段的计算机指令。

相对：一地址指定 one-address instruction（2.1047）、二地址指令 two-address instruction（2.1777）和三地址指令 three-address instruction（2.1733）。

英 文 索 引

中 文 索 引