软件工程术语和概念

SCMChina ++C = XiaoChun

2002.09.29 v1.0

版权声明

8848software.com
E-Mail: goodxxc888@163.com
2002.09.29

目录

软件危机

软件危机是一种现象,是指由于软件复杂程度愈来愈高，在计算机软件开发和维护时所遇到的一系列问题，具体表现在：

- 软件开发成本高，成本难以控制；
- 研制周期长，软件开发进度难以控制，周期拖得很长;
- 正确性难保证，软件质量差，可靠性难以保证;
- 软件维护困难，维护人员和维护费用不断增长;
- 软件发展跟不上硬件的发展和用户的要求。

软件工程

软件工程是一类求解软件的工程。它应用计算机科学、数学及管理科学等原理。借鉴传统工程的原则、方法，创建软件以达到提高质量、降低成本的目的。其中，计算机科学、数学用于构造模型与算法。工程科学用于制定规范、设计范型、评估成本及确定权衡；管理科学用于计划、资源、质量、成本等管理。软件工程是一门指导计算机软件开发和维护的工程学科。软件工程是一门交叉学科。

Boehm：运用现代科学技术知识来设计并构造计算机程序及为开发、运行和维护这些程序所必需的相关文件资料。

IEEE：软件工程是开发、运行、维护和修复软件的系统方法。

Fritz Bauer：建立并使用完善的工程化原则，以较经济的手段获得能在实际机器上有效运行的可靠软件的一系列方法。

软件工程框架

软件工程的三要素：目标、原则、活动的三维框架。

目标：

- 可用性：软件基本结构的实现及文档的可用程度。
- 正确性：软件产品达到预期功能要求的程度。
- 合算性：软件开发、运行的整个开销满足用户要求的程度。

原则：

- 采用适当的开发模型作为开发的指导，以控制软件开发的易变性：
- 运用良好的设计方法，提高软件质量和软件生产率：
- 有效的工程支持（工具支持）；
- 有效的管理。

活动：

- 软件生命周期的各个阶段。（需求、设计、实现、确认、支持）

软件工程原则

抽象、信息隐蔽、模块化、局部化、一致性、完整性

软件工程学

软件工程学包括软件开发技术（软件开发方法学，软件工具，软件工程环境）和软件管理技术（软件管理学，软件工程经济学）。

软件工程学出现的直接诱因好像是软件危机，但实际上，软件工程技术出现的深刻历史背景是计算机应用技术的发展，尤其是硬件技术的迅猛发展。计算机应用软件的规模和复杂性不断增加，带有强烈个体手工生产特性的软件作坊式的开发技术，已不能与之相适应。因此，软

件工程学的出现是软件开发技术发展的必然结果，软件开发的大生产规模特性注定了要用工程的方式进行，即对项目开发人员进行严密的组织管理，良好的协同配合。

软件工程技术有两个明显的特点：

- 强调规范化：为了使由许多人共同开发的软件系统能正确无误地工作，开发人员必须遵守相同的约束规范（用统一的软件开发模型来统一软件开发步骤和应进行的工作，用产品描述模型来规范文档格式，使其具有一致性和兼容性），规范化使软件生产摆脱了个人生产方式，进入了标准化、工程化阶段。
- 强调文档化：一个复杂的软件要让其他人员读懂并且理解，除程序代码外，还应有完备的设计文档来说明设计思想、设计过程和设计的具体实现技术等有关信息。因此文档是十分重要的，它是开发人员相互进行通信以达到协同一致工作的有利工具。而且，按要求进度提交指定的文档，能使软件生产过程的不可见性变为部分可见，从而便于对软件生产进度进行管理。最后，通过对提交的文档进行技术审查和管理审查，以保证软件的质量和有效的管理。所以必须十分重视文档工作。

软件工程管理

软件工程管理研究如何有效地对软件开发项目进行管理，以便于按照进度和预算完成软件项目计划，实现预期的经济和社会效益。软件工程管理包括成本估算、进度计划、人员组织、质量保证等多方面内容。

软件生命周期

从软件的计划起到废弃不用为止，划分为若干阶段，并赋予任务和活动，它们分别是：软件计划、软件需求分析、软件设计、编码、软件测试、软件维护。

软件开发方法学

软件开发方法学是以软件方法为研究对象的学科。主要涉及指导软件设计的原理和原则，以及基于这些原理、原则的方法和技术。狭义的也指某种特定的软件设计指导原则和方法体系。从构造的角度，软件开发方法学主要由三部分组成：

- NOTATION
- PROCESS
- TOOLS

软件开发模型

软件开发模型是软件开发全部过程、活动和任务的结构框架。软件开发模型能清晰、直观地表达软件开发全过程，明确规定要完成的主要活动和任务，它用来作为软件项目工作的基础。模型应该是稳定和普遍适用的。本质与目的：表征了软件开发活动的组织，给出了软件求解的计算逻辑。

瀑布模型

瀑布模型是一种软件开发方法，它遵循软件生命周期的划分，将软件生存周期的各项活动规定为按固定顺序连接的若干阶段工作，形如瀑布流水，最终得到软件产品。其特点是：阶段间的顺序性和依赖性，上一阶段结束才能进入下一阶段;每一阶段以前一阶段的结果为基础;要求软件需求阶段十分完善。

原型法

原型法是一种软件开发方法（主要针对事先不能完整定义需求的软件开发），具体是借助开发工具尽快地构造一个实际系统的简化模型，作为系统的框架，便于开发者与用户之间进行交流，从而根据用户的反馈准确地获得用户的需求，再根据需求增加系统的功能，以支持系统的最终设计和实现。

增量模型

增量模型即对软件开发活动进行如下组织：在设计了软件系统整体体系结构之后，首先完整地开发系统的一个初始子集；然后根据该子集建造一个更加精细的版本。如此不断地进行系统的增量开发。

喷泉模型

在面向对象的方法中提出了与瀑布模型相对应的喷泉模型。该模型认为软件生命周期的各个阶段是多次重复和重叠的。喷泉模型体现了各阶段之间的无缝、迭代。

螺旋模型

螺旋模型在原型法基础上的渐进修正，即在原型评价改进的多次反复过程中引入风险分析。

软件可靠性

软件可靠性是指软件系统能否在既定的环境条件下运行并实现所期望的结果，软件可靠性是软件最重要的质量要素之一。一般包括正确性，安全性和健壮性。

软件安全性

对于合理的一组输入，系统会给出正确的结果;而对于用户的有意或者无意的不合理输入，系统应能拒绝这种输入，并指出输入的不合理性，提醒用户注意。

软件健壮性

软件健壮性是指软件系统对环境变化的适应性。当软件系统所处的环境发生变化时，系统都能按照某种预定的方式作适当的处理，有效地控制事故的蔓延，避免灾难性的后果。

软件需求

IEEE 软件工程标准词汇表（1997年）中定义软件需求为：

（1）用户解决问题或达到目标所需的条件或能力（Capability）。

（2）系统或系统部件要满足合同、标准、规范或其它正式规定文档所需具有的条件或能力。

（3）一种反映上面（1）或（2）所描述的条件或能力的文档说明。

软件需求分析

软件需求分析的基本任务是准确地定义未来系统的目标，确定为了满足用户的需求，系统必须做什么。需求分析包括需求获取和需求规约：需求获取是系统分析员通过学习以及同用户的交往，熟悉用户领域的知识，并获得对未来系统的需求；需求规约是系统分析员在获得了用户的初步需求后，必须进行一致性分析和检查，通过和用户协商解决其中存在的二义性和不一致性，并以一种规范的形式准确地表达用户的需求，形成软件需求规格说明书。

软件需求规格说明书（SRS）

软件需求规格说明书是软件需求分析阶段得到的最终文档，它以形式化的术语和表示对软件的功能和性能进行详细而具体的描述。它是用户和开发者之间的技术合同，是软件设计、编码阶段的基础,也是软件测试和验收的依据。

数据流分析（DFA）

数据流分析是基于数据流的建模方法，主要思想是:将系统抽象为一系列的加工单元，各单元之间以数据流发生联系。

数据流图（DFD）

数据流图一种描述数据流和加工的图形表示。当数据输入到系统后，经过一系列变换（加工），最后输出新的数据。

数据字典（DD）

数据字典进一步解释数据流图中数据流和加工的具体含义，只有将数据流图和数据字典结合起来才能完整地描述一个系统。

数据流

数据流是由一组数据项组成，作为加工单元之间的联系。

加工单元

加工单元是数据的转换机制,是对数据进行处理变换的单元。

分解

根据系统的逻辑特性和内部各组成成分之间的逻辑关系，将大的、复杂的模块，划分为小的、简单的模块，以降低系统的复杂度。

抽象（DFA 中的抽象）

抽象是人类认识世界、改造世界时普遍采用的原则和方法，即在解决问题的过程中集中考虑与当前目标有关的方面，忽略与当前目标无关的方面。

在逐层分解中，上层是下层的抽象。分解和抽象是数据流分析中降低系统复杂度的手段。（结构化设计中）

软件结构

软件结构是软件模块之间的组成关系和调用关系，表示软件的总体结构，隐含地指出软件的控制层次体系。好的软件结构体现自顶向下的方式分配控制。

模块

模块是软件的组成部分。将软件划分为可单独命名和编址的元素，这些元素是一个或多个程序语句的集合，完成某种特定的功能，可被系统中其它部分调用，它们被称为模块。

模块化

按一定原则将软件划分成若干个模块，使每个模块完成一个子功能，然后将这些模块组装起来就可以完成系统要求的功能，这个过程就是模块化。模块化的目的是降低系统复杂性。

软件设计

从系统规约出发，形成软件设计方案的过程。即确定怎么做。结构化软件设计分为总体设计（初步设计）和详细设计。总体设计是确定系统的整体模块结构，具体完成将系统划分成模块、决定每个模块的功能、决定模块间的调用关系、决定模块间的接口。详细设计则根据得到的模块结构图，对其中每个模块给出过程属性的描述，即算法设计。其结果为模块结构图、每个模块的详细说明。

模块独立性

一个模块同其它模块的独立程度，软件的各组成模块具有独立的功能且与其它模块没有过多的相互作用的性质称为模块独立性。模块独立性是模块抽象和信息隐蔽的直接结果，是评价一个模块设计好坏的重要度量尺度, 是保证软件质量的关键。

为什么强调模块独立性？模块独立性好的软件容易开发，容易理解，容易修改，即易于开发研制、测试维护。

模块独立性有两个定性的度量：内聚、耦合。

定义的要点：独立的功能; 符合信息隐蔽、信息局部化的原则; 模块之间的关联和依赖性要尽量小。

好处：软件质量好; 易开发易研制; 易测试易维护。

信息隐蔽

高层抽象模块隐藏了功能实现的细节，即在设计和确定模块时，使一个模块内包含的信息（过程和数据），如果它不允许外部模块访问的话，其它模块是不能访问的。

内聚（块内联系）

内聚是指模块内部各组成成分之间相互联系的强度。

耦合（块间联系）

耦合指模块之间的依赖程度的度量，是模块独立性的直接衡量。包括紧密耦合、松散耦合及无耦合。

各种内聚、耦合定义

无耦合： 两个模块彼此独立的工作，没有直接的关系。
数据耦合：模块之间通过传递数据参数来交换信息。
标志耦合：模块之间通过传递公共参数指针或地址（数据的标记）相互作用产生的耦合。
控制耦合：模块之间通过传递控制信息相互作用产生的耦合。
公共耦合：两个以上模块通过共同引用一个全局数据（公共数据）产生的耦合。
内容耦合：一个模块直接访问、修改或操作另一个模块的内部数据，或不通过正常入口直接转入另一个模块而产生的耦合。（应该消除）
偶然内聚：模块的各成分之间毫无联系，即模块内语句无任何联系合成一个模块。
逻辑内聚：将逻辑上相关的功能合成一个模块。
时间内聚，一个模块完成的功能必须在同一时间内执行。
过程内聚：一个模块内部的处理成分是相关的且必须以特定的次序执行。

通信内聚：一个模块内的所有处理成分都集中在同一数据结构上。
顺序内聚：一个模块内的各个成分与同一功能紧密相关，而且必须顺序执行。
功能内聚：一个模块内所有成分完成且只完成单一功能。

变换型数据流图

可明显地将数据流图分解为输入、输出和主加工三个部分。

逻辑输入／输出

未经过主加工处理之前的输入数据称为逻辑输入；经过主加工处理后而未形成最终的输出数据之前称为逻辑输出。

事务型数据流图

某一个加工将它的输出分离成一串发散的数据流，并根据输入值选择其中一条路径，这样的数据流图就是事务型数据流图，这个加工就是事务中心。

模块作用范围

受该模块内一个判定影响的所有模块的集合。指一个模块内的一个判定的作用范围，一个判定的作用范围是指所有受这个判定影响的那些模块，只要该模块中含有一些依赖于这个判定的操作，那么这个模块就在这个判定的作用范围之内。（从功能角度）

模块控制范围

指这个模块本身以及所有直接或间接从属于它的模块的集合。（从结构角度）

设计准则（启发式规则）

用于改善软件设计，提高软件质量。
改进软件结构，提高模块独立性。设计出软件结构后，评价该结构（审查分析），通过模块分解或合并，力求降低耦合，提高内聚。
模块规模应该适中。深度、宽度、扇入和扇出应该适中。

- 深度：表示软件结构中控制的层数；
- 宽度：软件结构中同一层上最大的模块总数；
- 扇出：一个模块直接控制（调用）的下层模块数目（影响模块宽度）；
- 扇入：表明一个模块有多少个直接调用它的上层模块数目。

一个模块的作用范围应处于这个模块的控制范围内，力争降低模块接口的复杂性，模块功能应该可以预测，只要输入的数据相同就产生同样的输出。

结构化程序设计

它是良好风格的程序设计技术，与结构化分析、结构化设计方法衔接，程序的控制结构只由三种基本结构（顺序、选择、循环）组成；程序是单入口单出口的；采用自顶向下、逐步求精的设计方法。

伪码

伪码是用正文形式表示数据结构和处理过程的设计工具，是一种混合语言。它以结构化程序设计语言为控制框架（严格的关键字外部语法），其内部则用自然语言描述各种操作、条件和过程。是一种详细设计表示方法。

IPO 图

表示输入输出设计与软件过程之间关系的图解式设计表示方法。

软件测试

按照特定规程，发现软件错误的过程。测试是程序的执行过程，检查软件是否满足规定的要求，或是清楚地了解预期结果与实际结果之间的差异。其目的在于发现软件中的错误。软件测试是对需求分析、设计、编码的最后复审，是保证软件质量的一个重要组成部分。

测试过程

软件测试是一个有规则的过程，包括测试设计、测试执行以及测试结果比较。包括：环境模型、对象模型和错误模型。

软件测试的基本原则

- 设计测试用例时，要给出测试的预期结果；
- 开发和测试小组分立；
- 要设计非法输入的测试用例；
- 在对程序修改之后，要进行回归测试；
- 在进行深入的测试时，要集中测试容易出错的程序段。

Myers 软件测试十原则

- 程序员应避免测试自己编制的程序
- 测试用例的设计必须包括预期的输出结果
- 测试用例应包括有效的和期望的输入情况，也要包括无效的和不期望的输入情况
- 彻底检查每个测试结果
- 只检查程序是否做了它应该做的事仅仅完成了测试工作的一半，另一半则是要检查程序是否做了它不该做的事
- 避免不可重复的即兴测试，保留全部测试用例
- 一段程序中存在错误的概率与在这段程序中已发现的错误数成正比
- 测试是一项非常复杂、创造性的和需要高度智慧的挑战性任务
- 不要为了便于测试擅自修改程序
- 测试工作必须有明确的目标

路径测试技术

路径测试技术是一种白盒测试技术，路径测试技术依据程序的逻辑结构，通过合理地选择一组穿过程序的测试路径，以实现达到某种测试度量。

事务处理流程测试技术

事务处理流程是系统行为的一种表示方法，为功能测试建立了程序的动作模式。事务处理流程测试技术的基本步骤是：定义有用的图形模式，设计必要的测试用例以覆盖之。

测试用例

测试用例是为发现软件错误而设计的数据，它由两部分组成：输入数据的描述，程序执行后应产生的正确结果的精确描述。

测试用例设计

确定一组最有可能发现某个错误或某类错误的测试数据。

黑盒法

一种软件测试方法，测试软件的功能是否达到了预期的要求（达到软件规格说明书的要求），着眼于软件的外部特性，而不考虑软件的内部逻辑构造。

白盒法

白盒法一种软件测试方法，着眼于软件的内部结构，测试软件中所有的逻辑路径上的活动是否符合设计要求。

逻辑覆盖

以程序内部结构为基础的测试技术，测试中考虑测试用例对程序内部逻辑路径的覆盖程度，属于白盒法。

各种逻辑覆盖

- 语句覆盖：设计足够的测试用例，使程序中的每条语句都至少执行一次。
- 判定覆盖（分支覆盖）：设计足够的测试用例，使程序中的每个判定都至少获得一次“真”和“假”的值，从而使程序中的每个分支都至少执行一次。
- 条件覆盖：设计足够的测试用例，使程序中判定的每个条件获得各种可能的结果。
- 判定／条件覆盖：设计足够的测试用例，使程序中判定的每个条件获得各种可能的结果，同时使每个判定取得各种可能的值。
- 条件组合覆盖：设计足够的测试用例，使程序中每个判定的条件的各种可能组合至少出现一次。
- 语句覆盖——条件组合覆益，弱——强

等价类划分

设计的每个测试用例能够独自发现某一类错误。分两步走：首先划分等价类，然后设计测试用例。

边界值分析

经验表明：程序往往在处理边界情况时易犯错误，所以检查边界情况的测试用例是比较高效的。设计正好等于、大于或小于边界值的数据对程序进行测试。

调试

在测试之后纠正错误的工作，包括：确定错误的确切性能和位置；修改错误。（软件测试只是发现程序中有错误的迹象，即测试结果与预期结果不符，并不知道错误的位置及错误的原因）

单元测试

单元测试是对软件的最小单位——模块进行测试。往往采用白盒测试技术，主要考虑模块的以下四个特征：模块接口；局部数据结构；重要的执行路径; 错误执行路径。即首先测试穿过模块接口的数据流; 继之进行数据结构的测试；还要进行执行路径的选择测试；最后进行边界测试。在单元测试中，由于模块不是一个独立的程序，必须为每个模块开发驱动模块和承接模块。

集成测试

集成测试是软件组装的一个系统化技术，其目的是发现与接口有关的错误，将经过单元测试的模块构成一个满足设计要求的软件结构。可“自顶向下”或“自底向上”进行。

有效性测试（验收测试）

有效性测试是为发现软件实现的功能与软件需求规格说明书不一致的错误而进行的测试。经常采用黑盒测试技术。

系统测试

系统测试是把软件与其它系统元素结合在一起，并进行一系列整体和系统有效性测试。包括：恢复测试、安全测试、强度测试、性能测试。

支持模块（承接模块，桩模块）

支持模块在单元测试中代替被测试模块的下属模块。

驱动模块

驱动模块在单元测试中模拟调用被测试模块的模块。其作用是模拟“主程序”，接收不同的测试用例的数据并传递给被测试模块。

回溯

回溯一种纠错技术，从发现错误征兆的地方开始，人工地往回追溯源程序代码，直至发现错误为止。

软件维护

软件维护是软件生命周期的最后一个阶段。对现有运行软件进行修改而同时保留其主要功能不变的过程，即对交付的软件继续进行排错、修改和扩充。

纠正性维护

纠正性维护是诊断并改正在软件使用过程中出现的错误的过程。

适应性维护

适应性维护视为适应系统环境的变化而对软件进行修改的过程。

完善性维护

完善性维护是为满足或部分满足软件使用过程中用户提出的增加、改进软件功能的要求而对软件进行修改的过程。

预防性维护

预防性维护是为进一步改进软件的易维护性和可靠性而对软件进行的修改的过程。

维护的副作用

维护的副作用指由于修改软件而引入的错误，主要有：代码修改的副作用、修改数据的副作用、文档的副作用。

易维护性

软件能被理解、修正、移植和改进的程度。是软件开发的一个关键目标。影响软件易维护性的因素有：软件的开发方法、软件的开发条件。

回归测试

回归测试是为了保证对软件所做的修改没有把引入新的错误而重复过去已经进行过的测试。一个有选择性的重新测试过程。这个过程是为了探测在一个系统或系统构件修改过程中引入的错误，并证明修改没有引起未曾预料到的灾难性后果，或证明一个修改的系统构件仍然能满足具体的需求。

对象

对象是对客观世界事物的一种抽象，是由数据（属性）及其上操作（行为）组成的封装体。其主要特点为：自治性、封闭性，通信性。

对象是一组数据及其上的操作的描述。一个对象是一个封装和一个抽象：封装是对属性以及这些属性上专有的操作的封装; 抽象是对问题空间的抽象，指问题空间某类事物的一次或多次出现。

对象类（类）

对象类是一组具有相同属性和服务的对象的集合。每个对象都属于某个对象类，这个对象就是这个对象类的一个实例。

属性

属性是用来描述对象状态的数据，在类的每个对象中均有确定的值。属性为类及对象提供了更多的描述细节。属性描述隐藏在对象内部的信息，由该对象的服务专门操纵。问题域中的类一般保持相对的稳定性，而属性却比较容易改变。

如何定义属性：

- 策略和启发：
  - 按一般常识这个对象应该有哪些属性；
  - 在当前的问题域中，这个对象应该有哪些属性；
  - 根据系统责任的要求，这个对象应该有哪些属性；
  - 建立这个对象是为了保存和管理哪些信息；
  - 对象为了在服务中实现其功能，需要增设哪些属性；
  - 对象有哪些需要区别的状态，是否增加一个属性来区别这些状态；
  - 用什么属性表示整体-部分和实例连接
- 审查与筛选
  - 这个属性是否体现了以系统责任为目标的抽象；
  - 这个属性是否描述这个对象本身的特性；
  - 该属性是否破坏了对象特征的“原子性”
  - 该属性是否可以通过继承得到；
- 可以从其它属性直接导出的属性。
- 推迟到 OOD 考虑的问题

结构

结构是一种思维组织的方式，用来反映问题域空间事物之间的复杂关系。在面向对象中讨论的结构包括两种：一般／特殊结构和整体／部分结构。

一般／特殊结构

针对的是事物类之间的组织关系，体现了现实世界中事物的一般性与特殊性。例如：交通工具同汽车、飞机、轮船之间形成了一般／特殊结构，将汽车、飞机、轮船的共有特性概括在交通工具之中，而汽车、飞机、轮船等专有的特性则包含在各自的类中。

整体部分结构

整体部分结构表示事物的整体与部分之间的组合关系。例如，把汽车看成一个整体，那么发动机、交速箱、刹车装置等都是汽车的部件，相对于这个整体，就分别是一个局部。

继承

继承是表达类之间相似性的一种机制，即在已有的类的基础之上增量构造新的类，前者称为父类（或超类），后者称为子类。子类除自动拥有父类的全部属性和服务外，还可以进一步定义新的属性和服务。如果子类只从一个父类继承，则称为单继承；如果子类从一个以上父类继承，则称为多继承。

在分类结构中，允许一次定义公共属性和服务。对象共享其父类（在它之上）所定义的属性和服务；同时允许针对特殊情况扩展那些属性和服务。继承提供了一个用于标识公共属性和服务的显示方法。

服务

服务是为了完成某一任务，一个对象所提供的、并体现其责任的操作。属于同一类的所有对象共享相同的服务。

一个服务就是接收到一条消息后所执行的处理。服务是对象的动作，而这个动作可能会改变对象内部的状况或向外界提供某种功能。

实例连接

通过刻画一个对象与其它对象的映射关系，实例连接可进一步加强对对象状态的描述能力（属性描述的是对象状态）。

消息连接

消息连接表达了对象之间的处理相关性，是指一个服务为了完成其处理功能，而向另一个对象发出的消息请求。前者称为消息的“发送者”，后者称为“接收者”。所需的处理在发送者的服务规范中命名。并在接收者的服务规范中具体定义。消息连接是在属性的强制封装性和处理这些属性的服务之间建立必要而有限的接口。

抽象（OO 的抽象）

抽象是忽视一个主题中与当前目标无关的那些方面，从而更充分地注意与当前目标相关的方面。抽象是 OOA 中控制复杂性的最重要的手段。

过程抽象

任何一个完成明确定义功能的操作都可被使用者看作单个实体，尽管这个操作实际上可能由一系列更低级的操作来完成。

数据抽象

数据抽象定义了数据类型和施加于该类型对象的操作，并限定只能通过这些操作修改和观察对象的值。

信息隐蔽（封装）

开发整体程序结构时应用的法则，即将每个程序的成分隐蔽或封装在一个单一的模块中，定义每一个模块时，尽可能少地显示其内部的处理。

面向对象分析（OOA）

面向对象分析是面向对象的建模方法，主要思想是：将系统分解为若干相互关联的对象，每个对象具有若干属性及一组操作。OOA 的五个步骤：标识对象、标识结构、定义主题、定义属性（及实例连接）、定义服务（及消息连接）。

面向对象方法和结构化方法相比，具有以下一些特点：

- 面向对象方法强调把问题域的概念直接映射到对象以及对象之间的接口，符合人们通常的思维方式，减少了结构化方法从问题域到分析阶段的映射误差；
- 面向对象方法从分析到设计再到编码采用一致的模型表示，后一阶段可以直接复用前一阶段的工作成果，弥合了结构化方法从数据流图到模块结构图转换的鸿沟，减少了工作量和映射误差；
- 在客观世界以及作为它的映射的软件系统中，实体的结构是相对稳定的。面向对象方法通过把属性和服务封装在“对象”中，当外部功能发生变化时，保持了对象结构的相对稳定，使改动局限于一个对象的内部，减少了改动所引起的系统波动效应。所以，按照面向对象方法开发的软件，具有易于扩充、修改和维护的特性；
- 面向对象方法具有的继承性和封装性支持软件复用，并易于扩充，能较好地适应复杂大系统不断发展和变化的要求。

面向对象设计（OOD）

以数据为中心组织软件系统结构，遵循模块的一般准则并具有数据抽象，属性继承等模块特征。

面向对象设计的四个组成部分

问题域部分、人机交互部分、任务管理部分、数据管理部分。

控制复杂性的手段

抽象、信息隐蔽、继承、构造方法。

构造方法

人类在认识和理解现实世界的过程中，普遍运用着下面三个构造法则：

- 区分对象及其属性。例如：区分一棵树和树的大小或空间位置关系；
- 区分整体对象及其组成部分。例如：区分一棵树和树枝；
- 不同对象类的形成及区分。例如：所有树的类和所有石头的类的形成和区分。

计算机辅助软件工程 CASE

CASE 是一组工具和方法的集合，可以辅助软件开发生命周期各阶段进行软件开发。是对方法的补充和替代，有助于生成高质量的软件。

CASE 特征

应支持软件工程各阶段；支持技术型软件工程；适应特定的阶段、特定的方法；应尽量使软件工程师的劳动自动化。

逆向软件工程

分析一个软件的过程，以最大的努力去建立比源代码层次更高的软件表达形式。亦是一个设计的恢复过程，可以从一个现有的软件中提取有关数据、体系结构及过程实际方面的信息。

软件工具

软件工具是用于辅助和支持计算机软件的开发、运行、维护和管理等活动的一类软件。

软件工程开发环境（CASE 环境）

相关的一组软件工具的集合，这些工具按照一定的软件开发方法或按照一定的软件开发模型组织起来，支撑整个软件生存周期的各个阶段或部分阶段的活动。其宗旨是为计算机软件的生产提供计算机辅助手段，改变长期以来懂得手工式生产方式，以提高软件产品的生产率，降低软件开发成本和改善软件的质量。

CASE 是一类软件，是具有集成机制的软件工具集：

- 集成机制：实现工具之间的互操作和协调
- 工具集：支持开发中各阶段的活动，也是软件。

软件工程环境可从以下几种角度分类：

- 按软件开发模型及开发方法分类，有支持瀑布模型、演化模型、螺旋模型、喷泉模型以及结构化方法、信息模型方法、面向对象方法等不同模型及方法的软件工程环境。
- 按应用范围分类，有通用型和专用型软件工程环境；其中专用型软件工程环境与应用领域有关，故又可称为应用型软件工程环境。
- 按开发阶段分类，有前端开发环境（支持系统规划、分析、设计等阶段的活动）、后端开发环境（支持编程、测试等阶段的活动）、软件维护环境和逆向工程环境等。此类环境往往可通过对功能较全的环境进行剪裁而获得。

一个软件工程环境一般有如下特征：

- 仓库：仓库是软件工程环境最重要的特征，其同义词有字典、数据库、库等。仓库保存着正被定义的目标系统的描述，这些描述是对软件工程环境所支持的软件开发各阶段的工作产品的描述。在实现上，库可以是集中式的，也可以是分布式的。
- 工具的集成：软件工程环境提供了一组集成化的软件工具，通过这些工具操作和分析系统描述的结果。
- 用户友好的界面：软件工程环境可以运行在主机、服务器系统、工作站或 PC 机上，甚至把功能分布在不同的机器上。为方便用户，应提供一致的图形界面。
- 提取信息的能力：软件工程环境的用户在整个开发过程中，需要以不同的格式提取信息。
- 分析的能力：把用户的工作产品以集成的方式存储在软件工程环境中后，在分析当前收集或定义的数据的基础上，进行一致性检查和完整性分析。
- 可裁剪性和可扩展性：为了满足用户和开发组织的不同需求，软件工程环境必须是可裁剪的和可扩充的。

- 项目控制和管理：软件工程环境应有助于良好管理的实施。如帮助管理构造系统描述，收集信息并进行度量。
- 方法学的支持：软件工程环境可以支持某一种或多种系统开发方法，可供项目和开发组织使用。

基本功能，较完善的软件工程环境通常具有如下功能：

- 软件开发的一致性及完整性维护；
- 配置管理及版本控制；
- 数据的多种表示形式及其在不同形式之间自动转换；
- 信息的自动检索及更新；
- 项目控制和管理；
- 对方法学的支持；

软件工程环境五级模型（Wasserman）

- 平台集成：工具运行在相同的硬件／操作系统平台上；
- 数据集成：工具使用共享数据模型来操作；
- 表示集成：工具提供相同的用户界面；
- 控制集成：工具激活后能控制其它工具的操作；
- 过程集成：工具在一个过程模型和“过程机”的指导下使用。

软件过程

软件过程是软件生存周期中一系列相关过程，过程是活动的集合，活动是任务的集合，任务是将输入变换为输出的操作。按照不同人员的工作内容来分类，软件过程可分为三类：基本过程、支持过程和组织过程。

- 基本过程：指与软件生产直接相关的过程，包括：获取过程、供应过程、开发过程、运行过程、维护过程。
- 支持过程：是有关各方按他们的支持目标所从事的一系列相关活动。支持过程有助于提高系统或软件产品的质量，有助于系统的运行。包括：文档过程、配置管理过程、质量保证过程、验证过程、确认过程、联合评审过程、审计过程、问题解决过程。
- 组织过程：是指那些与软件生产组织有关的过程。包括：管理过程、基础设施过程、改进过程、培训过程。

软件质量

软件质量是指与软件产品满足规定的和隐含的需求的能力有关的特征或特性的全体，即所有描述计算机软件优秀程度的特性的组合。

软件质量保证

软件质量保证，是为保证产品和服务充分满足消费者要求的质量而进行的有计划、有组织的活动。软件质量保证是面向消费者的活动，是为了使产品实现用户要求的功能，站在用户立场上来掌握产品质量的。软件的质量保证就是向用户及社会提供满意的高质量的软件产品。

软件配置管理

软件配置管理过程是在整个软件生命周期中实施管理和技术规程的过程，它标识、定义系统中软件项并制定基线；控制软件项的修改和发行；记录和报告软件项的状态和修改申请；保证软件项的完整性、协调性和正确性；以及控制软件项的储存、装载和交付。

或者：

软件配置管理是一门用来记录并控制软件产品数据的管理学科。
或者：
软件配置管理是在适当的时机从适当的人那里得到正确的软件数据。

软件配置控制委员会（SCCB）

软件配置控制委员会是一组负责评估和审批配置项的变更的人员，以确保所有的变更都是经过审核的。

基线

基线是已经过正式评审和认可，作为以后进一步开发的基础，并且只有通过正式的更改控制规程才能进行更改的规格说明或产品。

剪裁过程

为了有效地实施软件过程，应针对特定领域的软件工程，对选定的过程模型和标准进行剪裁，以形成这一工程的模型及标准，形成该工程的各个软件过程和活动。剪裁过程作为一类软件过程，是对软件过程和活动实施剪裁的过程。其主要活动有：指明工程环境；收集信息；选取过程、活动和任务；编制文档。剪裁可分为两级，第一级是根据应用领域的不同进行剪裁；第二级是根据每一具体项目或合同进行剪裁。

统一建模语言（UML）

UML(Unified Modeling Language)是一种构建软件系统和文档的通用可视化建模语言。UML 能与所有的开发方法一同使用，可用于软件开发的整个生命周期。UML 能表达系统的静态结构和动态信息，并能管理复杂的系统模型，便于软件团队之间的合作开发。UML 不是编程语言，但支持 UML 语言的工具可以提供从 UML 到各种编程语言的代码生成，也可以提供从现有程序逆向构建 UML 模型。

设计视图(Design view)

设计视图(design view)包含了类、接口和协作。主要反映系统的功能需求。该视图的静态方面由类图和对象图表现，动态方面由交互图、状态图和活动图表现；

进程视图(Process view)

进程视图(process view)包含了形成系统并发与同步机制的线程和进程。该视图对进程视图的静态方面和动态方面的表现与设计视图相同，但注重于描述线程和进程的主动类；

实现视图(Implementation view)

实现视图(implementation view)包含了用于装配与发布物理系统的构件和文件，主要针对系统发布的配置管理，可以用各种方法装配它们。该视图的静态方面由构件图表现，动态方面由交互图、状态图和活动图表现；

部署视图（Deployment view）

部署视图（deployment view）包含了节点主要描述对组成物理系统的部件的分布、交付和安装。该视图的静态方面由部署图表现，动态方面由交互图、状态图和活动图表现。

用况视图(Use case view)

用况视图(use case view)由用况组成，描述可被最终用户、分析人员和测试者看到的系统行为。在 UML 中，该视图的静态方面由用况图表现，动态方面由交互图、状态图和活动图表现。

软件过程

软件过程是一个将用户需求转化为软件系统所需要的活动的集合。

统一软件开发过程（RUP）

RUP 是一个通用过程框架，可以应付种类广泛的软件系统、不同的应用领域、不同的组织类型、不同的性能水平和不同的项目规模。

“统一过程”是基于构件的，这意味着利用它开发的软件系统是由构件构成的，构件之间通过定义良好的接口相互联系。在准备软件系统的所有蓝图的时候，“统一过程”使用的是“统一建模语言（Unified Modeling Language）”。事实上，UML 是“统一过程”的有机组成部分——它们是被同步开发的。

然而，真正使“统一过程”与众不同的方面可以用三个句话来表达：它是用例驱动的、以基本架构为中心的、迭代式和增量性的。

模型

模型是对系统的一种抽象，是从某个视点、在某种抽象层次上详细说明被建模的系统。模型是对构架设计师和开发人员构造的系统的抽象。

领域模型

领域模型能捕获系统语境中最重要的对象类型。领域对象代表系统工作的环境中存在的“事情”或发生的事件。

业务模型

业务模型是理解一个组织中业务过程的技术。

- □ 业务用况模型分别从业务用况和业务参与者的角度来描述业务过程。
- □ 业务对象模型是业务的内部模型。它描述了如何由一组工作员使用一些业务实体和工作单元来实现每个业务用况。

分析模型

分析模型是包含分析类和用况实现的分析包的层次结构

设计模型

设计模型是包含设计类、用况实现-设计和接口等设计子系的层次结构

实施模型

实施模型是一个描述系统物理分布的对象模型，它描述了系统功能在各个节点上的分布。

- □ 每个节点表示一个计算资源；
- □ 节点拥有表示相互间的通信方式的关联；
- □ 实施模型能够描述几种不同的网络配置，包括测试和模拟配置；
- □ 实施到一个节点上的构件定义了这个节点的功能（或者处理）；

- 实施模型本身表示软件拓扑结构和硬件拓扑结构之间的映射关系；

软件能力成熟度模型（CMM）

CMM 是对软件组织进化阶段的描述，随着软件组织定义、实施、测量、控制和改进其软件过程，它们经过这些阶段逐步前进。该模型使得确定当前过程能力的工作和识别软件质量和过程改进最关键的问题变得容易，从而对选择过程改进战略提供指导。

正式评审

正式评审是一次正式的会议，在此会议上将产品提交给最终用户、顾客或其他有兴趣的各方，以得到评论和批准。它也可以是对项目的管理和技术活动及项目进程的评审。

度量

度量是某物的线数、容量、数量或总量（例如300 源代码行或7 个设计文件页）。

度量单位

度量单位（Measure）是度量的一个单位（例如源代码行或者设计的文件页数）。