详细设计

详细设计就是要在总体设计阶段成果的基础上，考虑如何实现定义的软件系统，直到对系统中的每个模块给出足够详细的过程描述。

结构化程序设计技术是进行详细设计的逻辑基础。它采用自顶向下逐步求精的设计方法和单入口单出口的控制结构。

结构化程序设计的三种基本控制结构是顺序、选择、循环。

详细设计的工具有程序流程图、盒图、PAD 图、判定表、判定树和 PDL 语言等。

程序复杂性度量

程序复杂性主要指模块内程序的复杂性。它直接关联到软件开发费用的多少、开发周期的长短和软件内部潜伏错误的多少等。

1） 代码行度量法

度量程序的复杂性，最简单的方法就是统计程序的源代码行数。该方法的基本考虑是统计一个程序模块的源代码行数目，并以源代码行数作为程序复杂性的度量。

Lipow 及其他研究者得出一个结论：对于少于 100 个语句的小程序，源代码行数与出错率是线性相关的。随着程序的增大，出错率以非线性方式增长。

2） McCabe 度量法

McCabe 度量法是一种基于程序控制流的复杂性度量方法。McCabe 定义的程序复杂性度量值又称环路复杂性，它是基于程序图中环路的个数。

如果把程序流程图中每个处理符号都退化成一个结点，原来联结不同处理符号的流线变成连接不同结点的有向弧，这样得到的有向图就叫做程序图。

计算有向图 G 的环路复杂性的公式：

$$V(G) = m - n + 2$$

其中，V(G)是有向图 G 中的环路个数，m 是图 G 中有向弧个数，n 是图 G 中结点个数。

详细设计说明书

1. 引言
2. 总体设计。软件结构
3. 程序描述。逐个模块描述其功能、性能、输入、输出、算法等

编码

所谓编码就是把软件设计的结果翻译成计算机可以“理解”的形式—用某种程序设计语言书写的程序。

按照软件工程的方法论，程序的质量基本上由设计的质量决定。但是，编码使用的语言，特别是写程序的风格和途径也对程序质量有相当大的影响。

程序设计语言的分类、选择

大量实践表明，高级程序设计语言较汇编语言有很多优点。

程序设计风格包括四个方面：

源程序文档化、数据说明、语句结构和输入/输出方法

测试

软件测试的目的就是软件投入生产性运行之前，尽可能多地发现软件中的错误。目前软件测试仍然是保证软件质量的关键步骤。

软件测试在软件生命周期中横跨两个阶段：编码和测试。

Grenford J.Myers 就软件测试目的提出以下观点：

1） 测试是程序的执行过程，目的在于发现错误。

2） 一个好的测试用例在于能发现至今未发现的错误

3） 一个成功的测试是发现了至今未发现的错误的测试

软件测试的原则

1） 应当尽早地不断地进行软件测试

2） 测试用例应由测试输入数据和与之相对应的预期输出结果这两部分组成

3） 程序员应避免检查自己的程序

4） 在设计测试用例时，应当包括合理的输入条件和不合理的输入条件

5） 充分注意测试中的群集现象

6） 严格执行测试计划，排除测试的随意性

7） 应当对每个测试结果做全面检查

8） 妥善保存测试计划、测试用例、出错统计和最终分析报告，为维护提供方便

常分单元测试/模块测试，集成测试，确认测试和系统测试。关系重大的还可做平行运行。

设计测试方案是测试阶段的关键技术问题，基本目标是选用最少量的高效测试数据，做到尽可能完善的测试，从而尽可能多地发现软件中的问题。

动态测试两个基本方法

黑盒测试/功能测试

等价类划分

边界值分析

错误推测法

因果图法

白盒测试/结构测试

语句覆盖：使得每一可执行语句至少执行一次

判定覆盖：使得程序中每个判断的取真分支和取假分支至少执行一次

条件覆盖：使得程序中每个判断的每个条件的可能取值至少执行一次

判定-条件覆盖：使得判断中每个条件的所有可能取值至少执行一次，同时每个判断的所有可能判断结果至少执行一次

条件组合覆盖：使得每个判断的所有可能的条件取值组合至少执行一次

路径覆盖：覆盖程序中所有可能的路径

设计测试方案的实用策略是，用黑盒法设计基本的测试方案，再用白盒法补充一些必要

的测试方案。

应该认识到，软件测试不仅仅指利用计算机进行的测试，还包括人工进行的测试（例如，代码审查）。两种测试途径各有优缺点，互相补充，缺一不可。

因果图方法最终生成的就是判定表。它适合于检查程序输入条件的各种组合情况。

利用因果图生成测试用例的基本步骤是：

1） 分析软件规格说明描述中，哪些是原因（即输入条件或输入条件的等价类），哪些是结果（即输出条件），并给每个原因和结果赋予一个标识符。

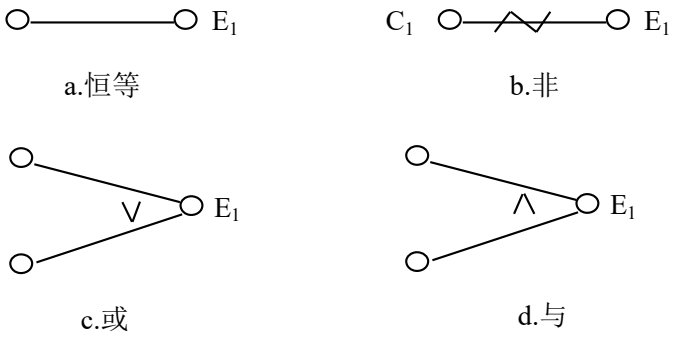
2） 分析软件规格说明描述中的语义，找出原因与结果之间、原因与原因之间对应的是什么关系？根据这些关系，画出因果图。

3） 由于语法或环境限制，有些原因与原因之间、原因与结果之间的组合情况不可能出现。为表明这些特殊情况，在因果图上用一些记号标明约束或限制条件。

4） 把因果图转换成判定表。

5） 把判定表的每一列拿出来作为依据，设计测试用例。

通常在因果图中，用 Ci 表示原因，Ei 表示结果，其基本符号如图所示。各结点表示状态，可取值 0 或 1。0 表示某状态不出现，1 表示某状态出现。



a.恒等　　b.非

c.或　　d.与

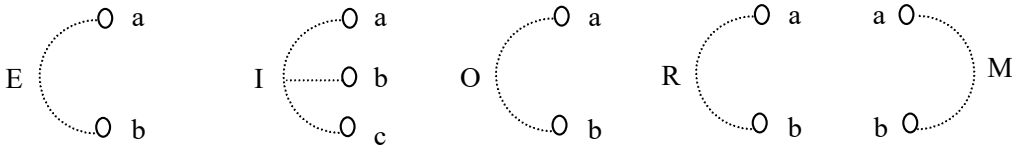
a. 恒等：若原因出现，则结果出现；若原因不出现，则结果也不出现。

b. 非：若原因出现，则结果不出现；若原因不出现，结果反而出现。

c. 或：若几个原因中有 1 个出现，则结果出现；若几个原因都不出现，则结果不出现。

d. 与：若几个原因都出现，结果才出现。若其中有 1 个原因不出现，则结果不出现。

为了表示原因与原因之间、结果与结果之间可能存在的约束条件，在因果图中可以附加一些表示约束条件的符号。从输入（原因）考虑，有 4 种约束；从输出（结果）考虑，还有 1 种约束，如图所示：



a.E(互斥*排它)　b.I(包含*或)　c.O(唯一)　d.R(要求)　e.M(屏蔽)

a. E（互斥）：表示 a、b 两个原因不会同时成立，两个中最多有一个可能成立。

b. I（包含）：表示 a、b、c 三个原因中至少有一个必须成立。

c. O（唯一）：表示 a 和 b 中必须有一个，且仅有一个成立。

d. R（要求）：表示当 a 出现时，b 必须也出现。a 出现时不可能 b 不出现。

e. M（屏蔽）：表示当 a 是 1 时，b 必须是 0。而当 a 为 0 时，b 的值不定。

调试

软件调试是在进行了成功的测试之后才开始的工作。

调试活动由两部分组成：

1） 确定程序中可疑错误的确切性质和位置

2） 对程序（设计、编码）进行修改，排除这个错误

几种主要的调试方法：

强行排错

回溯法排错

归纳法排错

演绎法排错

测试和调试是软件测试阶段的两个关系极端密切的过程，它们通常交替进行。

测试中的可靠性分析

在软件开发的过程中，利用测试的统计数据，估算软件的可靠性以控制软件的质量是至关重要的。

推测错误的产生频度

估算错误产生频度的一种方法是估算平均失效等待时间 MTTF（Mean Time To Failure）。MTTF 估算公式（Shooman 模型）是：

$$MTTF = \frac{1}{K\left(E_T / I_T - E_C(t) / I_T\right)}$$

其中，K 是一个经验常数，美国一些统计数字表明，K 的典型值是 200；E_T 是测试之前程序中原有的故障总数；I_T 是程序长度（机器指令条数或简单汇编语句条数）；t 是测试（包括排错）的时间；$E_C(t)$是在 0-t 期间内检出并排除的故障总数。

公式的基本假定是：

1)、单位（程序）长度中的故障数 E_T/I_T 近似为常数，它不因测试与排错而改变。统计数字表明，通常 E_T/I_T 值的变化范围在 $0.5*10^{-2}$—$2*10^{-2}$ 之间。

2)、故障检出率正比于程序中残留故障数，而 MTTF 与程序中残留故障数成正比。

3)、故障不可能完全检出，但一经检出立即得到改正。

可靠性累积曲线公式

$$E_C(t) = E_T\left(1 - e^{-K_1 t}\right)$$

维护

所谓软件维护就是在软件已经交付使用之后，为了改正错误或满足新的需要而修改软件

的过程。

软件工程学的主要目的就是提高软件的可维护性，降低维护的代价。软件可维护性，是指纠正软件系统出现的错误和缺陷，以及为满足新的要求进行修改、扩充或压缩的容易程度。

软件的可理解性、可测试性和可修改性是决定软件可维护性的基本因素。

软件维护通常包括四类活动：

1） 为了纠正使用过程中暴露出来的错误而进行的改正性维护；

2） 为了适应外部环境的变化而进行的适应性维护；

3） 为了改进原有的软件而进行的完善性维护；

4） 为了改进将来的可维护性和可靠性而进行的预防性维护。

在软件维护中，影响维护工作量的程序特性有以下 6 种：

a. 系统大小

b. 程序设计语言

c. 系统年龄

d. 数据库技术的应用

e. 先进的软件开发技术

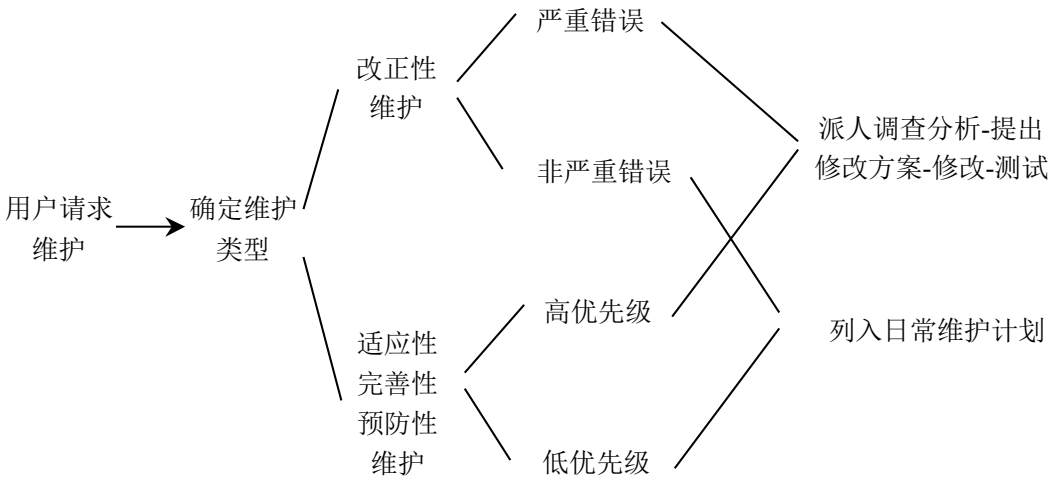
f. 其他

维护有三种副作用：文档、代码、数据。

软件维护工作流程

1、 确认维护要求

2、 由维护组织管理员确认维护类型



可理解性：表明人们通过阅读源代码和相关文档，了解程序及其如何运行的容易程度。

可靠性：表明一个程序按照用户的要求和设计目标，在给定的一段时间内正确执行的概率。

可测试性：表明论证程序正确性的容易程度。

可修改性：表明程序容易修改的程度。

可移植性：表明程序转移到一个新的计算环境的可能性的大小，或者它表明程序可以容易地、有效地在各种各样的计算环境中运行的容易程度。

效率：表明一个程序能执行预定功能而又不浪费机器资源的程度。

可使用性：从用户观点出发，把可使用性定义为程序方便、实用及易于使用的程度。

各阶段结束标准

计划阶段	问题定义	关于规模和目标的报告书
	可行性研究	可行性论证报告（系统的高层逻辑模型）
开发阶段	需求分析	需求规格说明书（系统的逻辑模型）
	概要设计	概要设计说明书（系统层次图或结构图）
	详细设计	详细设计说明书（各模块内部详细算法）
	编码	源程序清单，单元测试方案与结果
	测试	综合测试方案和结果
实施阶段	维护	完整准确的维护记录

软件管理

成本估算

代码行技术

任务分解技术

效益估计

系统效益包括经济效益和社会效益两部分。

货币的时间价值、纯收入、投资回收期、投资回收率

进度安排

进度安排是软件项目管理中的一项重要内容。管理复杂的工程项目非常困难，最好的办法是把它分解成一系列比较容易管理的子任务。但是分解后又容易只注意对各个子任务的管理，以致忽略了对工程总体情况的了解和管理。因此需要有某种工具既支持把项目分解成较小的子任务，又能帮助管理人员保持对工程总体情况的洞悉和管理。

1、横道图（甘特图）

Gantt 图能很形象地描绘任务分解情况，以及每个子任务（作业）的开始时间和结束时间，因此是进度计划和进度管理的有力工具，它具有直观简明和容易掌握、容易绘制的优点，但是 Gantt 图也有三个主要缺点：

1.不能显式地描绘各项作业彼此间的依赖关系；

2.进度计划的关键部分不明确，难于判定哪些部分应当是主攻和主控的对象；

3.计划中有潜力的部分及潜力的大小不明确，往往造成潜力的浪费

当把一个工程项目分解成许多小任务，并且它们彼此间的依赖关系又比较复杂时，仅仅用 Gantt 图作为安排进度的工具是不够的，不仅难于作出既节省资源又保证进度的计划，而且还容易发生差错。

2、工程网络

工程网络是制订进度计划时另一种常用的图形工具，它同样能描绘任务分解情况以及每项作业的开始时间和结束时间。此外，它还显式地描绘各个作业彼此间的依赖关系。因此，工程网络是系统分析和系统设计的强有力的工具。

涉及的几个概念：机动时间、关键路径

在制定进度计划时仔细考虑和利用工程网络中的机动时间，往往能够安排出既节省资源

又不影响最终竣工时间的进度表。

人员组织

管理软件开发通常采用层次结构。

一般来说，程序设计小组的规模应该较小，视工程规模以 2-8 人为宜。

主程序员组用经验多、技术好、能力强的程序员作为主程序员。同时，利用人和计算机在事务性工作方面给主程序员提供充分支持，而且所有通信都通过一两个人进行。

软件项目的开发实践表明，软件开发各个阶段所需要的技术人员类型、层次和数量是不同的。软件项目的计划与分析阶段只需要少数人，主要是系统分析员、从事软件系统论证和概要设计的软件高级工程师和项目高级管理人员。概要设计时增加一部分高级程序员，详细设计时，要增加软件工程师和程序员，在编码和测试阶段还要增加程序员、软件测试员。到测试阶段结束时，软件项目开发人员的数量达到顶峰。软件运行初期，参加软件维护的人员比较多，过早解散软件开发人员会给软件维护带来意想不到的困难。软件运行一段时间后，由于软件开发人员参与改正性维护，软件出错率会很快衰减，这时软件开发人员可以逐步撤出。如果系统不做适应性或完善性维护，需要留守的维护人员就不多了。

人员-时间折衷定律：在时间允许的情况下，适当减少人员会提高工作效率，降低软件开发成本。即软件开发宁可时间长一点，人员少一点。这样可以大大减少人员之间的通信开销，工作效率会更高。

软件配置管理

Babich 曾说过：“协调软件开发使得混乱减到最小的技术叫做配置管理。配置管理是一种标识、组织和控制修改的技术，目的是使错误达到最小并最有效地提高生产率。”

软件配置管理（SCM，Software Configuration Management）应用于整个软件生存期。因为变更在任何时刻都可能发生，因此，软件配置管理活动的目标就是为了标识变更、控制变更、确保变更更正确地实现，并向其他相关的人报告变更。

基线是软件生存期中各开发阶段末尾的特定点，又称里程碑。由正式的技术评审而得到的软件配置协议和软件配置的正式文本组成基线。它的作用是把各阶段工作的划分更加明确化，使连续的工作在这些点上断开，以便检验和肯定阶段成果。

CMM（Capability Maturity Model）软件机构能力成熟度模型

五个级别：初始级、可重复级、已定义级、已管理级、已优化级

软件复用

对建立软件目标系统而言，所谓复用（reuse），就是利用某些已开发的、对建立新软件系统有用的软件元素来生成新系统。这里所说的软件元素可以包括需求规格说明、设计过程、设计规格说明、程序代码、测试用例、度量等。对于新的软件开发项目而言，这些元素或者是构成新软件系统的构件，或者能在软件开发过程中发挥某种作用。通常，将这些软件元素称为可复用构件。

复用的分类

软件复用的范围不仅仅涉及源程序代码，Caper Jones 定义了 10 种可能复用的软件要素：

1、 项目计划

2、 成本估计

3、 体系结构
4、 需求模型和规格说明
5、 设计
6、 源程序代码
7、 用户文档和技术文档
8、 用户界面
9、 数据结构
10、 测试用例

可复用构件的 3C 模型

可有很多方法来描述可复用构件，其中最理想的方法是由 Tracz 提出的 3C 模型，即概念（Concept）、内容（Content）和上下文（Context）。

1、 概念
2、 内容
3、 上下文

面向对象的软件工程

面向对象=对象+类+继承+通信

对象可以定义为系统中用来描述客观事物的一个实体，它是构成系统的一个基本单位，由一组属性和一组对属性进行操作的服务组成。三个主要特点：自治性、封装性、通信性。

属性一般只能通过执行对象的操作来改变。

操作又称为方法或服务，它描述了对象执行的功能，若通过消息传递，还可以为其他对象使用。

而所谓的消息是一个对象与另一个对象的通信单元，是要求某个对象执行类中定义的某个操作的规格说明。

把具有相同特征和行为的对象归结在一起就形成了类。类是某些对象的模板，抽象地描述了属于该类的全部对象的属性和操作。

属于某个类的对象叫做该类的实例。

如果某几个类之间具有共性的部分（信息结构和行为），将其抽取出来放在一个一般类中，而将各个类的特有的部分放在特殊类中分别描述，则可建立起特殊类对一般类的继承。

继承是软件开发中重用概念的核心，另外便于模块修改，同时继承可使模块简化，导致模块比较小容易理解。

Coad 与 Yourdon 的方法（Object-oriented Analysis and Design，OOAD）：分为 OOA 和 OOD。

1.OOA

1）形式地说明所面对的应用问题，最终成为软件系统基本构成的对象，还有系统所必须遵从的，由应用环境所决定的规则和约束。

2）明确地规定构成系统的对象如何协同合作，完成指定的功能。

OOA 要建立分析模型，有五个层次：类和对象层、属性层、服务层、结构层、主题层。允许两种结构：整体-部分，通用-特殊。

通过 OOA 建立的系统模型是以概念为中心的，因此称为概念模型。这样的模型由一组相关的类组成。构造和评审 OOA 概念模型的顺序由 5 个层次组成，即类和对象层、属性层、

服务层、结构层、主题层。这 5 个层次不是构成软件系统的层次，而是分析过程中的层次也可以说是问题的不同侧面。每个层次的工作都为系统的规格说明增加了一个组成部分。当 5 个层次的工作全部完成时，OOA 的任务也就完成了。

2.OOD

OOD 还是采用这样五个层次，但分别用在建立系统的四个组成成分上：问题论域、人机交互、数据管理、任务管理。OOD 模型类似于构造蓝图，以最完整的形式全面地定义了如何用特定的实现技术建立起一个目标系统。在 OOA 模型和 OOD 模型中使用了共同的表示法，这有助于从分析到设计的转换，并有助于在当前的设计和实现中维护 OOA 模型。与 OOA 模型一样，OOD 模型也有 5 层结构，又被划分为四个部分：问题论域、人机交互、数据管理、任务管理。这些组成部分把实现技术隐藏起来，使之与系统的基本问题论域行为分离开。在 OOA 中，实际上只涉及到问题论域部分，其他 3 个部分是在 OOD 中加进来的。即，对于问题论域部分只是做进一步的细化，其它 3 个部分中将识别和定义新的类和对象。

OMT（Object Modeling Technique 对象建模技术）以面向对象思想为基础，通过构造一组相关模型（对象模型、动态模型和功能模型）来获得关于问题的全面认识（即问题的领域模型）。对象模型（object model）代表了系统的静态的、结构方面的特性。动态模型（dynamic model）代表了系统对象之间的时间的、行为的、控制方面的特性。功能模型（functional model）主要描述值与值之间的函数关系。

其中，对象模型是 3 个模型中最关键的模型，它的作用是描述系统的静态结构，包括构成系统的类和对象，它们的属性和操作，以及它们之间的关系。在 OMT 中，类与类之间的关系叫做关联。

要想对一个系统了解得比较清楚，首先应考察它的静态结构，即在某一时刻它的对象和这些对象之间相互关系的结构。然后，应考察在任何时刻对对象及其关系的改变。系统的这些涉及时序和改变的状况，用动态模型来描述。动态模型着重于系统的控制逻辑。它包括两个图，一是状态图，一是事件追踪图。状态图是一个状态和事件的网络，侧重于描述每类对象的动态行为。事件追踪图侧重于说明发生于系统执行过程中的一个特定场景，是完成系统某个功能的一个事件序列。概括地讲，状态图叙述一个对象的个体行为，而事件追踪图则给出多个对象所表现出来的集体行为。

功能模型着重于系统内部数据的传送和处理。功能模型定义“做什么”，动态模型定义“何时做”，对象模型定义“对谁做”。功能模型表明，通过计算，从输入数据能得到什么样的输出数据，不考虑参加计算的数据按什么时序执行。功能模型由多个数据流图组成。

这三个模型从不同角度对系统进行描述，分别抓住了系统的一个重要方面，组合起来构成了对系统的完整描述。OMT 认为一个典型的软件过程是三个方面的合作：它的 DS（对象模型）、它按时间顺序的操作（动态模型）和它所改变的值（功能模型）。每个模型都和其它模型的实体相关。对象模型指出事件要发生在什么方面，动态模型指出什么时候发生，功能模型则指出要发生什么。

对象模型、动态模型和功能模型都包含了同样的概念：数据、序列和操作，但它们描述了系统的不同方面，同时也互相引用。

OTM 法已发展成支持整个的软件生命周期，由以下四个阶段组成：分析、系统设计和对象设计和实现。

Booch 方法用到六种图形：类图、对象图、模块图、进程图、交互作用图、状态迁移图。

类图和对象图，着重于类和对象的定义。其中，类图描绘类和类之间的关系。对象图表示具体的对象和在对象间传递的消息。

模块图和进程图，针对着软件系统的结构。类和对象被分配给具体的程序构件，模块图就是用来描绘这些程序构件的。由于许多面向对象系统包括可能在一组分布式处理器上执行的多个程序，进程图就使得设计者能在一个大系统中描绘过程如何被分配给特定的处理器。

状态迁移图和交互作用图，这两个文件主要用于动态地模拟事件的发生和它对系统状态的作用。状态迁移图用来说明每一类的状态空间，触发状态迁移的事件（从一个对象到另一个对象的单个消息叫作一个事件），以及在状态迁移时所执行的操作。交互作用图用于追踪系统执行过程中的一个可能的场景，也就是几个对象在共同完成某一系统功能中所表现出来的交互关系。

Booch 强调在 OO 设计中反复的处理和开发人员的创造性是最重要的。Booch 认为软件开发是一个螺旋上升的过程。在这个螺旋上升的每个周期中，有以下几个步骤：发现类和对象；确定它们的含义；找出它们之间的相互关系；说明每一个界面及其实现类与对象。

《软件工程》中英文缩写及其含义

SE（Software Engineering）软件工程
SA（Structured Analysis）结构化分析
SD（Structured Design）结构化设计
SP（Structured Programming）结构化编程
OOA（Object-Oriented Analysis）面向对象的分析
OOD（Object-Oriented Design）面向对象的设计
OOP（Object-Oriented Programming）面向对象的编程
SRS（Software Requirements Specification）软件需求规格说明
E-R 图（Entity-Relationship）实体-关系图
IPO（IPO 图，Input Process Output）输入-处理-输出图
HIPO（HIPO 图，Hierarchical Input Process Output）层次图加输入-处理-输出图
JSD（Jackson System Development）Jackson 系统开发方法
JSP（Jackson Structured Programming）Jackson 结构程序设计方法
SADT（Structured Analysis and Design Technique）结构化分析与设计技术
LCP（Logical Construction of Programs）程序逻辑构造
DSSD（Data Structured System Development）数据结构化系统开发方法
SC（Structure Chart）结构图
PAD（Problem Analysis Diagram）问题分析图
N-S（N-S 图，Box Diagram）盒图
PDL（Procedure Design Language）过程设计语言，伪码
MRR（Maintenance Request Report）维护申请报告
SPF（Software Problem Form）软件问题报告
SCR（Software Change Report）软件修改报告
MTTF（Mean Time To Failure）平均失效等待时间
CASE（Computer-Aided Software Engineering）计算机辅助软件工程
SCM（Software Configuration Management）软件配置管理

SQA（Software Quality Assurance）软件质量保证
CPM（Critical Path Method）关键路径法
CMM（Capability Maturity Model）软件过程成熟度模型
OMT（Object Modeling Technique）对象建模技术
UML（Unified Modeling Language）统一建模语言
CRC（Class Responsibility Collaborator）类-职责-伙伴