返回总目录

目　　录

第 1 章　UML 简介

UML （统一建模语言，Unified Modeling Language）是一种建模语言，是第三代用来为面向对象开发系统的产品进行说明、可视化和编制文档的方法。它是由信息系统（IS，Information System）和面向对象领域的三位著名的方法学家：Grady Booch，James Rumbaugh 和 Ivar Jacobson（称为“三个好朋友”，the Three Amigos）提出的。这种建模语言得到了“UML 伙伴联盟”的应用与反馈，并得到工业界的广泛支持，由 OMG 组织（Object Management Group）采纳作为业界标准。UML 取代目前软件业众多的分析和设计方法（Booch，Coad，Jacobson，Odell， Rumbaugh，Wirfs-Brock 等），成为一种标准，这是软件界的第一次有了一个统一的建模语言。目前，OMG 已经把 UML 作为公共可得到的规格说明（Publicly Available Specification，PAS）提交给国际标准化组织（ISO）进行国际标准化。预计 PAS 进程将在今年完成，使 UML 最终正式成为信息技术的国际标准。

1.1　UML 的产生和成长

从二十世纪八十年代初期开始，众多的方法学家都在尝试用不同的方法进行面向对象的分析与设计。有少数几种方法开始在一些关键性的项目中发挥作用，包括 Booch、OMT、Shlaer/Mellor、Odell/Martin、RDD、OBA 和 Objectory。到了二十世纪九十年代中期，出现了第二代面向对象方法，著名的有 Booch'94、OMT 的沿续以及 Fusion 等。此时，面向对象方法已经成为软件分析和设计方法的主流。这些方法所做的最重要的尝试是，在程序设计艺术与计算机科学之间寻求合理的平衡，来进行复杂软件的开发。

由于 Booch 和 OMT 方法都已经独自成功地发展成为世界上主要的面向对象方法，因此 Jim Rumbaugh 和 Grady Booch 在 1994 年 10 月，共同合作把他们的工作统一起来，到 1995 年成为“统一方法（Unified Method）”版本 0.8。随后，Ivar Jacobson 加入，并采用他的用例（use case）思想，到 1996 年，成为“统一建模语言”版本 0.9。1997 年 1 月，UML 版本 1.0 被提交给 OMG 组织，作为软件建模语言标准化的候选。其后的半年多时间里，一些重要的软件开发商和系统集成商都成为“UML 伙伴”，如 Mircrosoft、IBM、HP 等。它们积极地使用 UML 并提出反馈意见，最后于 1997 年 9 月再次提交给 OMG 组织，于 1997 年 11 月 7 日正式被 OMG 采纳作为业界标准。UML 的形成过程见图 1-1 所示。现在，OMG 已经把 UML 作为公共可得到的规格说明（Publicly Available Specification，PAS）提交给国际标准化组织（ISO）进行国际标准化。

UML 是 Booch、Objectory 和 OMT 方法的结合，并且是这三者直接的向上兼容的后继。另外它还吸收了其它大量方法学家的思想，包括 Wirfs-Brock、Ward、Cunningham、Rubin、Harel、Gamma、Vlissides、Helm、Johnson、Meyer、Odell、Embley、Coleman、Coad、Yourdon、Shlaer 和 Mellor。 通过把这些先进的面向对象思想统一起来，UML 为公共的、稳定的、表达能力很强的面向对象开发方法提供了基础。

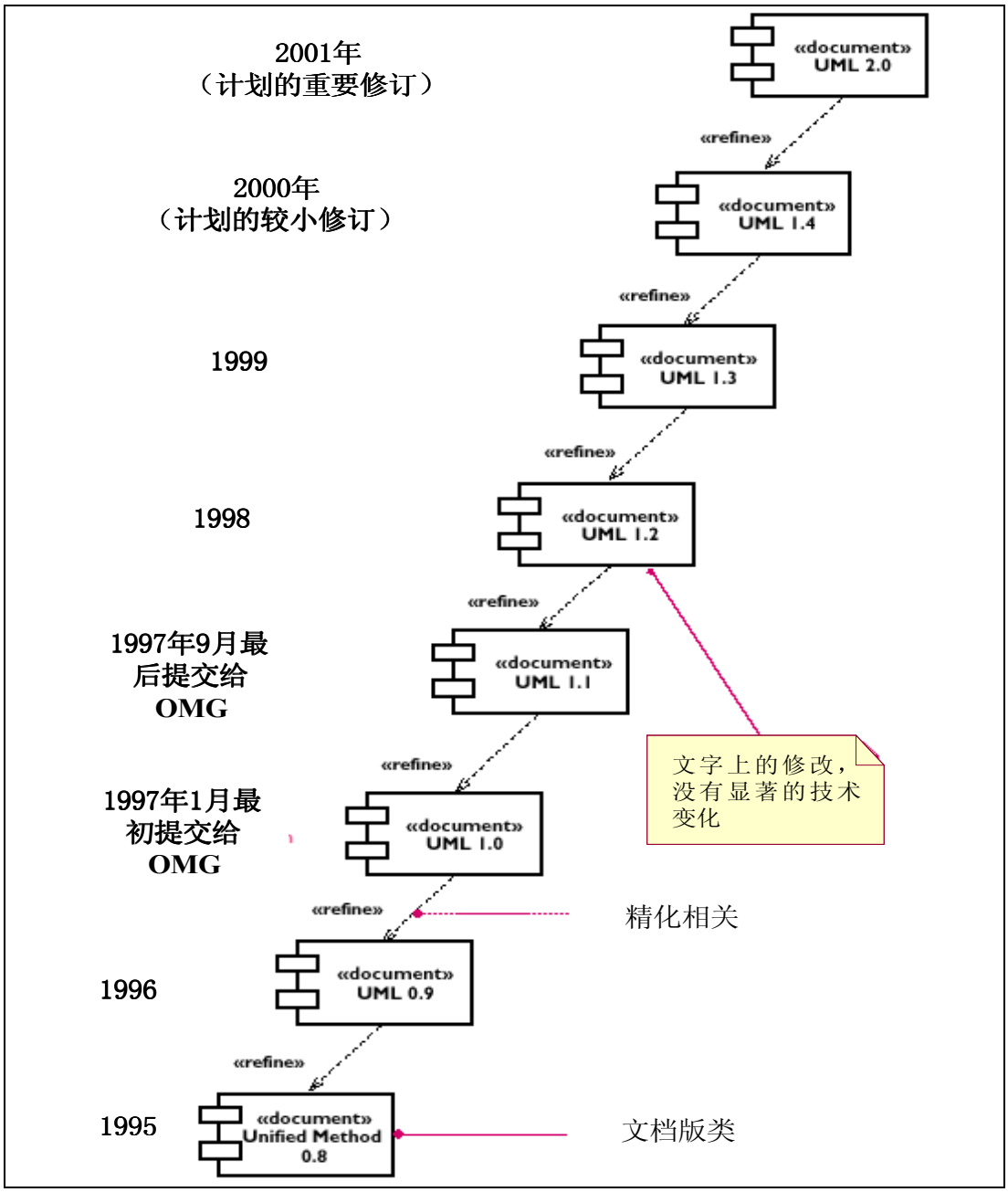

图 1.1　UML 的成长过程

1.2　什么是 UML

UML 是一种标准的图形化建模语言，它是面向对象分析与设计的一种标准表示。它：

- 不是一种可视化的程序设计语言，而是一种可视化的建模语言；

- 不是工具或知识库的规格说明，而是一种建模语言规格说明，是一种表示的标准；
- 不是过程，也不是方法，但允许任何一种过程和方法使用它。

UML 的目标是：

- 易于使用、表达能力强，进行可视化建模；
- 与具体的实现无关，可应用于任何语言平台和工具平台；
- 与具体的过程无关，可应用于任何软件开发的过程；
- 简单并且可扩展，具有扩展和专有化机制，便于扩展，无需对核心概念进行修改；
- 为面向对象的设计与开发中涌现出的高级概念（例如协作、框架、模式和组件）提供支持，强调在软件开发中，对架构、框架、模式和组件的重用；
- 与最好的软件工程实践经验集成；
- 可升级，具有广阔的适用性和可用性；
- 有利于面对对象工具的市场成长。

1.2.1 UML 的架构

UML 是由图和元模型组成的。图是 UML 的语法，而元模型则给出的图的意思，是 UML 的语义。UML 的语义是定义在一个四层（或四个抽象级）建模概念框架中的，这四层分别是：

- 元元模型（meta-metamodel）层，组成 UML 最基本的元素“事物（Thing）”，代表要定义的所有事物；
- 元模型（metamodel）层，组成了 UML 的基本元素，包括面向对象和面向组件的概念。这一层的每个概念都是元元模型中“事物”概念的实例（通过版类化）；
- 模型（model）层，组成了 UML 的模型，这一层中的每个概念都是元模型层中概念的一个实例（通过版类化），这一层的模型通常叫做类模型（class model）或类型模型（type model）；
- 用户模型（user model）层，这层中的所有元素都是 UML 模型的例子。这一层中的每个概念都是模型层的一个实例（通过分类），也是元模型层的一个实例（通过版类化）。这一层的模型通常叫做对象模型（object model）或实例模型（instance model）。

1.2.2 UML 的模型、视图、图与系统架构建模

UML 是用来描述模型的，它用模型来描述系统的结构或静态特征、以及行为或动态特征。它从不同的视角为系统的架构建模，形成系统的不同视图（view），包括：

- 用例视图（use case view），强调从用户的角度看到的或需要的系统功能，这种视图也叫做用户模型视图（user model view）或想定视图（scenario view）；
- 逻辑视图（logical view），展现系统的静态或结构组成及特征，也称为结构模型视图（structural model view）或静态视图（static view）；
- 并发视图（concurrent view），体现了系统的动态或行为特征，也称为行为模型视图（behavioral model view）、过程视图（process view）协作视图（collaborative）、动态视图（dynamic view）；

- 组件视图（component view），体现了系统实现的结构和行为特征，也称为实现模型视图（implementation model view）和开发视图（development view）；
- 展开视图（deployment view），体现了系统实现环境的结构和行为特征，也称为环境模型视图（implementation model view）或物理视图（physical view）；

在必要的时候，还可以定义其它架构视图。

每一种 UML 的视图都是由一个或多个图（diagram）组成的，一个图就是系统架构在某个侧面的表示，它与其它图是一致的，所有的图一起组成了系统的完整视图。UML 提供了九种不同的图，可以分成两大类，一类是静态图，包括用例图、类图、对象图、组件图、配置图；另一类是动态图，包括序列图、协作图、状态图和活动图。也可以根据它们在不同架构视图的应用，把它们分成：

- 在用户模型视图：用例图（Use case diagram），描述系统的功能；
- 在结构模型视图：类图（Class diagram），描述系统的静态结构；对象图（Object diagram），描述系统在某个时刻的静态结构；
- 在行为模型视图：序列图（Sequence diagram），按时间顺序描述系统元素间的交互；协作图（Collaboration diagram），按照时间和空间的顺序描述系统元素间的交互和它们之间的关系；状态图（ State diagram），描述了系统元素的状态条件和响应；活动图（Activity diagram），描述了了系统元素的活动；
- 在实现模型视图：组件图（Component diagram），描述了实现系统的元素的组织；
- 在环境模型视图：展开图（Deployment diagram），描述了环境元素的配置，并把实现系统的元素映射到配置上。

1.3 UML 与面向对象的软件分析与设计（OOA&D）

每一位软件设计方法学家都有许多有关软件质量的理论，他们会讨论软件危机、软件质量低下，以及良好的设计的重要性。那么 UML 对提高软件的质量有什么帮助吗？

1.3.1 标准的表示方法

UML 是一种建模语言，是一种标准的表示，而不是一种方法（或方法学）。方法是一种把人的思考和行动结构化的明确方式，方法需要定义软件开发的步骤、告诉人们做什么，如何做，什么时候做，以及为什么要这么做。而 UML 只定义了一些图以及它们的意义，它的思想是与方法无关。因此，我们会看到人们将用各种方法来使用 UML，而无论方法如何变化，它们的基础是 UML 的图，这就是 UML 的最终用途——为不同领域的人们提供统一的交流标准。

我们知道软件开发的难点在于一个项目的参与包括领域专家、软件设计开发人员、客户以及用户，他们之间交流的难题成为软件开发的最大难题。UML 的重要性在于，表示方法的标准化有效地促进了不同背景人们的交流，有效地促进软件设计、开发和测试人员的相互理解。无论分析、设计和开发人员采取何种不同的方法或过程，他们提交的设计产

品都是用 UML 来描述的，这有利地促进了相互的理解。

1.3.2 与软件开发的成功经验集成

UML 尽可能地结合了世界范围内面向对象项目的成功经验，因而它的价值在于它体现了世界上面向对象方法实践的最好经验，并以建模语言的形式把它们打包，以适应开发大型复杂系统的要求。

在众多成功的软件设计与实现的经验中，最突出的两条，一是注重系统架构的开发，一是注重过程的迭代和递增性。尽管 UML 本身没有对过程有任何定义，但 UML 对任何使用它的方法（或过程）提出的要求是：支持用例驱动（use-case driven）、以架构为中心（architecture-centric）以及递增（incremental）和迭代（iterative）地开发。

注重架构意味着不仅要编写出大量的类和算法，还要设计出这些类和算法之间简单而有效地协作。所有高质量的软件中似乎大量是这类的协作，而近年出现的软件设计模式也正在为这些协作起名和分类，使它们更易于重用。最好的架构就是“概念集成（conceptual integrity）”，它驱动整个项目注重开发模式并力图使它们简单。

迭代和递增的开发过程反映了项目开发的节奏。不成功的项目没有进度节奏，因为它们总是机会主义的，在工作中是被动的。成功的项目有自己的进度节奏，反映在它们有一个定期的版本发布过程，注重于对系统架构进行持续的改进。

1.4 UML 的应用领域

UML 被用来为系统建模，它可应用的范围非常广泛，可以描述许多类型的系统，它也可以用来系统开发的不同阶段，从需求规格说明到对已完成系统的测试。

1.4.1 在不同类型系统中的应用

UML 的目标是用面向对象的方式描述任何类型的系统。最直接的是用 UML 为软件系统创建模型，但 UML 也可用来描述其它非计算机软件的系统，或者是商业机构或过程。以下是 UML 常见的应用：

- 信息系统（Information System）：向用户提供信息的储存、检索、转换和提交。处理存放在关系或对象数据库中大量具有复杂关系的数据；
- 技术系统（Technical System）：处理和控制技术设备，如电信设备、军事系统或工业过程。它们必须处理设计的特殊接口，标准软件很少。技术系统通常是实时系统；
- 嵌入式实时系统（Embedded Real-Time System）：在嵌入到其它设备如移动电话、汽车、家电上的硬件上执行的系统。通常是通过低级程序设计进行的，需要实时支持；
- 分布式系统（Distributed System）：分布在一组机器上运行的系统，数据很容易从一个机器传送到另一台机器上。需要同步通信机制来确保数据完整性，通常是建立在对象机制上的，如 CORBA，COM/DCOM，或 Java Beans/RMI 上；

系统软件（System Software）：定义了其它软件使用的技术基础设施。操作系统、数据库和在硬件上完成底层操作的用户接口等，同时提供一般接口供其它软件使用；

- 商业系统（Business System）：描述目标、资源（人，计算机等），规则（法规、商业策略、政策等）和商业中的实际工作（商业过程）。

要强调的是，通常大多数系统都不是单纯属于上面的某一种系统，而是一种或多种的结合。例如，现在许多信息系统都有分布式和实时的需要。

商业工程是面向对象建模应用的一个新的领域，引起了人们极大的兴趣。面向对象建模非常适合为公司的商业过程建模。运用商业过程再工程（Business Process Reengineering，BPR）或全质量管理（Total Quality Management，TQM）等技术，可以对公司的商业过程进行分析、改进和实现。使用面向对象建模语言为过程建模和编制文档，使过程易于使用。

UML 具有描述以上这些类型的系统的能力。

1.4.2 在软件开发的不同阶段中的应用

UML 的应用贯穿在系统开发的五个阶段，它们是：

- 需求分析。UML 的用例视图可以表示客户的需求。通过用例建模，可以对外部的角色以及它们所需要的系统功能建模。角色和用例是用它们之间的关系、通信建模的。每个用例都指定了客户的需求：他或她需求系统干什么。不仅要对软件系统，对商业过程也要进行需求分析；
- 分析。分析阶段主要考虑所要解决的问题，可用 UML 的逻辑视图和动态视图来描述：类图描述系统的静态结构，协作图、状态图、序列图、活动图和状态图描述系统的动态特征。在分析阶段，只为问题领域的类建模——不定义软件系统的解决方案的细节（如用户接口的类、数据库等）；
- 设计。在设计阶段，把分析阶段的结果扩展成技术解决方案。加入新的类来提供技术基础结构——用户接口，数据库操作等。分析阶段的领域问题类被嵌入在这个技术基础结构中。设计阶段的结果是构造阶段的详细的规格说明；
- 构造。在构造（或程序设计阶段），把设计阶段的类转换成某种面向对象程序设计语言的代码。在对 UML 表示的分析和设计模型进行转换时，最好不要直接把模型转化成代码。因为在早期阶段，模型是理解系统并对系统进行结构化的手段。
- 测试。对系统的测试通常分为单元测试、集成测试、系统测试和接受测试几个不同级别。单元测试是对几个类或一组类的测试，通常由程序员进行；集成测试集成组件和类，确认它们之间是否恰当地协作；系统测试把系统当作一个“黑箱”，验证系统是否具有用户所要求的所有功能；接受测试由客户完成，与系统测试类似，验证系统是否满足所有的需求。不同的测试小组使用不同的 UML 图作为他们工作的基础：单元测试使用类图和类的规格说明，集成测试典型地使用组件图和协作图，而系统测试实现用例图来确认系统的行为符合这些图中的定义。