计算机科学1999Vol.26№.10

基于 UML 集成化支持环境的柔性软件开发过程*)

A Flexible Software Development Process Based on the UML Integrated Supporting Environment

王 云 周伯生

(北京航空航天大学软件工程研究所 北京100083)

Abstract UML is an unified modeling language of Oriented-Object. UML supports not only Oriented-Object analysis and design, but also the whole process starting from requirement analysis. How to use the visual modeling technology correctly to solve problems faced in the process of developing software, and the research on software modeling process and supporting tools are always the international focus. In the thesis, a modeling process is introduced which based on the UML integrated supporting environment, and then a flexible software development process is put forward.

Keywords UML, UML integrated supporting environment, UML flexible software development process

面向对象开发方法主要由以下三方面组成:开发语言、开发过程和支持工具。开发语言包括建模语言、程序语言和按文档格式组织的文字语言等。开发过程指规范化地使用开发语言,系统化地组织管理开发。支持工具在建模语言的基础上对开发过程提供支持。UML 是建模语言而不是方法,这是因为 UML 中没有过程的概念,而过程正是方法的一个重要组成部分。UML 本身独立于过程,这意味着用户在使用 UML 进行建模时,可以选用任何适合的过程。过程的选用与软件开发过程的不同因素有关,诸如软件的种类(实时系统、信息系统和桌面产品等)、开发组织的规模(单人开发、小组开发和团队开发等)等。用户将根据需要选用不同的过程。然而对大多数用户来说,如果有一个可供参考的大致统一的过程框架,无疑对使用 UML 建模有着积极的意义。

UML 不仅支持面向对象的分析与设计,还支持从需求分析开始的软件开发全过程。但是,如何恰当地将这些可视化图形建模技术,用于解决软件开发所面临的问题以及对建模过程和支持工具的研究,仍是目前该领域的热点问题。本文将在我们开发的 UML 集成化支持环境的基础上,给出使用该环境进行建模的过程,提出一个基于 UML 集成化支持环境的柔性软件开发过程。

1. UML 集成化支持环境简介

我们认为,UML 集成化支持环境是一个在需求牵引下自适应的系统开发环境,其组成应包括 UML 可视化建模系统、UML 模拟系统、UML 代码生成系统、UML 逆向变换系统、UML 质量控制系统及其支持软件等。这些环境均应基于 UML 的语法规则和语义定义,如图1所示。由此可知,该支持环境是一个能支持系统建模、系统模拟和系统生成的"闭环式开发"的集成化支持环境。

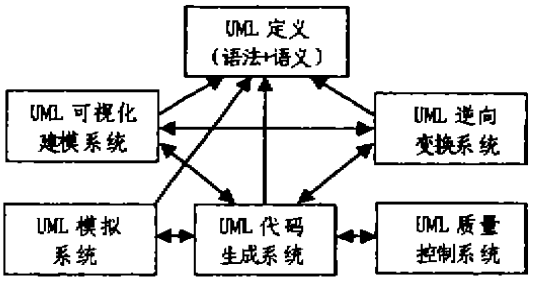

图1 UML 集成化支持环境的框架

UML 可视化建模系统。支持从系统需求、系统分析到系统设计的建模过程,提供 UML 图形的编辑和美化工具,保证得到语法正确、语义完备且一致的

*)本文项目和本文是在美国 Funsoft 公司科研基金的资助下完成。王云 硕士,主要研究领域为面向对象软件构造、软件开发方法学、软件工程环境、UML 集成化支持环境。周伯生 博士生导师,主要研究领域为软件工程、软件工程环境、过程工程、过程工程环境。

UML 图形模型，并提供包括文档管理和图形打印等辅助支持。

UML 模拟系统。支持对 UML 模型的功能模拟和性能模拟，包括系统功能(需求)和用户界面的模拟、行为模型的模拟和系统性能的模拟。

UML 代码生成系统。能支持可视化对象和行为的代码生成。UML 代码生成系统也称之为 UML 支持环境的正向变换系统，它将建模和编码过程有机地统一起来。

UML 软件质量控制。软件系统的质量往往是大型、复杂系统成败的关键。难以保证软件系统质量的主要原因首先是由于软件系统固有的复杂性。其次是由于软件系统及其管理工作固有的不可见性。采用定量软件工程有利于改善可见性，但是还不可能做到完全透明。因此，在当代软件技术水平的前提下，建造软件质量控制环境势在必行。

UML 逆向变换系统。当用户对生成的代码进行修改后，逆向转换机制将用户的修改转换到模型上。保证模型和代码间的一致性，使系统的扩充、增删和维护工作顺利进行。

2. UML 建模过程的基本特性

使用 UML 进行建模的过程应包含以下三个基本特性：用例驱动、以体系结构为中心、迭代式的增量开发。

用例驱动。在 UML 中，用例捕获系统的功能需求，驱动软件的整个开发过程，确保分析中罗列的所有功能均被实现。用例是审核和测试系统的依据。由于用例中包含了系统中的所有功能描述，因此将影响到所有的开发阶段。如图2所示，在图中的需求分析阶段，用例用于捕获系统的功能并与用户进行交流和商讨，从而达成共识；在设计和实现阶段，用于指导设计和编码，即用例中描述的功能必须实现；在测试阶段，用例是审核系统的依据，是测试系统的基础。

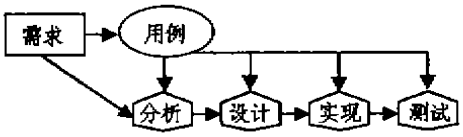

图2 用例影响软件开发的各个阶段

以体系结构为中心。使用 UML 进行建模的过程应该以体系结构为中心。这意味着应该在开发的早期尽量建立一个良好的系统体系结构，然后建立原型和进行评估，并在开发过程中不断地细化。体系结构把系统划分为几个部分，描述各部分间的关系、相互作用、通信机制以及添加和修改某一部分的原则。定义一个良好的体系结构对于实现一个易于修改、易于理解和可重用的系统至关重要。

迭代式的增量开发。使用 UML 建立模型时，不应试图一次定义出模型的所有细节，而应分成一系列较小的迭代过程，在每个迭代过程中逐步增加信息，逐步进行细化。一次迭代或一系列迭代可认为是对系统进行功能完善的过程，即完成系统的一次增量。迭代和增量式的开发就是定义一系列的开发阶段。每一个阶段都将产生系统的一个版本(完成系统的一次增量)，并要"交付使用"。当然，这里并非是交付给真正的客户去使用，而是内部提交，并从技术、经济效益以及迭代过程本身这几方面进行评估。除了要评估技术和经济效益，还要对迭代过程本身进行评估，看迭代过程是否存在缺陷，总结经验教训为下次迭代做准备。图3显示了迭代和增量式的开发过程。

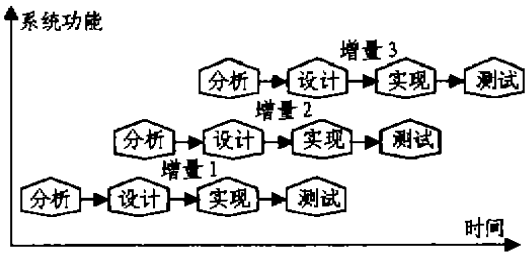

图3 迭代和增量式的开发过程

3. UML 柔性软件开发过程

由于软件系统的规模越来越大，复杂程度不断提高，同时又面临着激烈的竞争对手和瞬息万变的市场，所以传统的软件开发模式越来越难以满足当前市场的需求。新的产品开发周期已不再是一次性的从需求定义、软件设计、实现和交付，迭代式增量开发方式已得到了广泛采用。除了传统概念上的时间、质量和花费三个主要要求外，产品支持系统的"柔性"成为更加重要的核心需求。因此本文在研究 Rational 公司提出的 Objectory 过程框架的基础上，提出了 UML 集成化支持环境支持的开发过程，并体现出柔性软件开发的特点。图4显示了本文定义的软件开发过程框架。

图4中的决策、计划、构造和提交四个阶段是从管理人员看到的宏过程；而分析、设计、实现和测试是从开发人员看到的微过程。宏过程中各阶段的含义介绍如下：

决策阶段。定义项目的目标和范围，即在这个产品周期中应做什么、如何做并进行可行性分析。另外可以进行基本的域分析。在这个阶段，需要考虑项目的商业属性。比如粗略估计项目的开发费用、市场潜力，进行

风险分析，调查是否有其它的竞争产品等等。计划完成后，向项目决策人汇报，由决策人决定是否开发该项目。

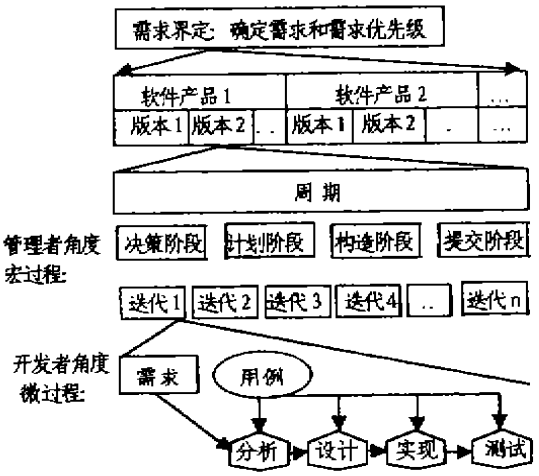

图4 软件过程开发框架

计划阶段。进入这一阶段，对需求已经有了大致的了解。应根据决策阶段所做的计划进行更详细的系统分析。详尽地进行功能域和问题域分析，定义基本的系统结构，借助原型法来验证一些想法是否合理，然后起草项目计划。计划的实质是为构造阶段建立迭代开发序列，并将用例分派到各次迭代中，当所有用例都已分配到各个迭代中，且各个迭代的开始日期确定下来后，计划工作便可宣告完成。由此可见，用例是进行项目计划的基础，这正是UML强调用例的原因。

构造阶段。通过一系列迭代过程构建系统。每次迭代都包括分析、设计、实现和测试几项活动。完成一次迭代后，应为用户演示，并完成系统测试，表明已正确实现了用例中描述的功能，这样做的目的是为了减少风险，如果困难被推到最后，将隐藏着很大的风险。除了刚提到的这些活动外，还应编写系统的开发和使用文档。在构造阶段的最后一个迭代完成之际，将交付一个版本，包括系统的开发文档和用户手册。

提交阶段。提交阶段将产品交付到最终用户手中，这意味着系统进入实际的使用阶段。提交阶段包括下列活动：销售、包装、安装、配置、培训、技术支持、维护。除此之外，对构造阶段形成的用户手册进行补充、增加有关销售、安装、配置等内容。

从图4可以看出，尽管产品开发周期分成不同的阶段，面向对象开发过程中通常的活动还是存在的，这些活动将在阶段的每次迭代中进行，本文称之为微过程。各阶段对应如下活动：(1)决策阶段：在较高的层次上进行项目的计划、分析和结构设计，重在建立产品的整体概貌，而非定义其中的细节。(2)计划阶段：包括项目计划、分析、结构化设计和详细设计，细化决策阶段建立的产品概貌，为系统的实现奠定基础。(3)构造阶段：包括结构设计、详细设计、实现、集成和测试。(4)提交阶段：包括集成、测试及系统维护。另外还包括产品包装、销售和培训等活动。

UML集成化支持环境支持的是从开发人员看到的微过程，并且在微过程中贯穿了柔性软件开发的概念。图5显示了柔性软件开发模型。所谓柔性软件开发是指软件开发过程在需求的牵引下，首先建立系统的顶层模型，对模型进行模拟、分析和调整。然后将顶层模型自顶向下进行分解，建立该系统各个子系统的模型，对这些子模型进行模拟、分析和调整。之后将子模型的模拟结果，逐次代入上层，再对该上层模型进一步进行模拟、分析和调整，如有不适，则进行修改。因此整个建模过程是一个"自顶向下建模，由底向上修改"的反复迭代的过程。简言之，柔性软件开发过程是在需求牵引下，首先自顶向下分层细化地建模，然后通过对模型的虚拟执行，由底向上地逐层上移修改，直至各层的模拟结果都满足需求为止。

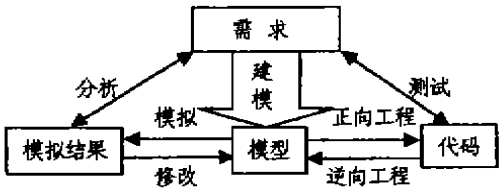

图5 柔性软件开发模型

在保证模型正确的基础上，最后进行代码的生成。同时考虑到对需求修改的灵活性和快速响应能力，实施能够反馈修改的"闭环开发"。即不仅能支持从模型到代码的自动生成，将新的模型转换为代码，还能支持从代码到模型的逆向变换，将原有的代码转化成模型，进行再次分析、修改和调整，进行新一轮的开发，从而为增量式开发提供支持。这样不仅能做到分阶段提交产品，也提高了对用户需求变化的响应速度和应变能力，以满足用户不断变化的新的需求。

下面就详细介绍如何使用UML集成化支持环境进行软件开发的过程。

需求分析。目的是在用户和系统开发者间达成共识。在需求分析阶段，应该从用户的角度去看待系统，而不应过早地考虑技术细节。除了要注重系统功能性的一面，系统中一些非功能性的需求也应引起足够的重视，例如系统运行的效率、可靠性，系统的规模、运行环境、与其它系统的集成问题以及使用何种编程语言等等。另外还应考虑系统的开发费用。然后列出系统中的所有用例，使用UML可视化建模系统画出系统的

用例图。在实际运作中，即便得不到所有用例，至少也应列出最重要的用例。因此，在这一阶段应安排与用户交流以便收集用例。用例的描述不必很详细，通常一段文字或几句话就够了，但应保证用户和开发人员都能明白其内涵。另一个任务是勾画系统的概念层模型，借助 UML 可视化建模系统描绘概念层类图和活动图来帮助理解用例中描述的功能需求和工作方式。需求分析的结果就是完成一份用户、开发者双方均认可的需求规格说明，可使用 UML 可视化建模系统生成需求规格说明文档。

领域分析。目的是建立特定领域模型。建立域模型有助于考察用例，即从需求规格说明中抽取出类，并描述类之间的关系。在分析阶段应抛弃所有技术或实现细节。所建模型是理想化的模型，仅需列出要重点解决的问题。该阶段包括如下几个典型活动：

(1)学习系统中涉及到的领域知识。研究需求规格说明、术语表、系统描述或与用户和领域知识专家进行交流，这是一个研究和学习的过程。

(2)定义或分析用例。如果在需求分析阶段已经定义了用例，那么在这一阶段则可对这些用例做进一步的分析。如果在前一阶段没有定义用例，那么在这一阶段应该使用用例技术进行系统的功能定义。

(3)列出特定领域中的关键类。通过开发人员和有关领域专家的交流讨论，列出系统特定领域中需要处理的关键类。列出的所有类都不是一成不变的，在整个开发过程中，由于经验的不断积累，可能会增加一些新类，或删除一些不再适用的类。

(4)使用 UML 可视化建模系统描绘类、类之间的静态关系(包括关联关系、聚集关系、依赖关系等等)。

(5)使用 UML 可视化建模系统描述类的对象行为和对象间的合作关系。顺序图和合作图适合描述单个用例中几个对象的行为。当行为较为简单时，顺序图和合作图是最好的选择。但行为变复杂时，它们将失去其清晰度。因此，如果想显示跨越多用例或多线程的复杂行为，可考虑使用活动图。活动图描述一个用例或类操作的工作流程，活动图鼓励发现并行过程，以减少商业过程中不必要的串行工作。另外，顺序图和合作图仅适合描述对象间的合作关系，而不适合对行为进行精确定义，如果想描述跨越多个用例单个类的行为，则应使用状态图，并且为帮助理解类而画它的状态图。

(6)进行评审。在模型通过语法检查、一致性检查和完备性检查后，可使用 UML 模拟系统对行为模型进行模拟。根据预先定义的语义，模拟行为模型对系统的描述是否真实地反映了客观现实和用户需求。而特定领域类模型则应交给领域专家进行讨论，考察该模型是否正确。

(7)建立基本的用户界面原型。只需搭建一个大致的框架，不一定面面俱到，然后与用户代表进行讨论，征求他们的意见。

(8)作为需求理解的一部分，还需为复杂难懂的用例建立原型。原型法是理解动态工作的一种极有价值的技术。借助 UML 代码生成系统的支持，产生系统原型，并尽早提供给用户，以确保建模的有效性。

设计。设计阶段将考虑系统中所有的技术细节，对分析阶段的结果进行扩展、细化和修正，这一阶段的重点在于：如何在计算机上实现分析阶段中建立的各个模型，所以应考虑运行环境上的限制及具体的实现细节。在分析阶段描述的类属性和操作应尽量保持不变，除非常有必要(比如有错误)外不应进行修改。过渡到设计阶段，将类进一步细化，并可辅助交互图来描述实现用例时类之间的交互关系。如果一个类具有复杂的生命周期行为，则可画出它的状态图。设计阶段中典型的活动包括：

(1)结构设计。将分析阶段定义的类根据其功能放进不同的功能包中。另外还需定义新的包(例如数据库包、通信包和用户界面包)，并定义包之间的依赖关系和通信机制。包有利于描述系统的逻辑组成部分，以及各部分之间的依赖关系。

(2)考虑系统的并发特性。考察系统中是否存在并发行为，如果存在，则应使用 UML 中的并发机制(如异步消息、同步技术)来描述系统中对共享资源的访问。

(3)定义系统的输出格式。包括用户界面的布局、风格。报告打印的格式以及与其它系统通信的格式等等。

(4)使用类库和组件。考虑使用必要的类库和现有的组件，以提高系统的结构化程度。缩短系统的开发周期。

(5)如果使用关系数据库，则应建立类与数据库的映射关系。这一关系应在类图中进行描述，从数据库中读取数据的机制同样属于结构设计的范畴。

(6)处理系统运行过程中出现的异常情况和错误。在设计阶段应考虑如何处理系统运行过程中出现的异常情况和错误，比如系统工作时突然停电，或由于计算机系统资源耗尽而引起的死机等。当出现这些异常情况时，应保证不丢失数据。

(7)完成系统配置。显示系统高层的物理结构，使用组件图和配置图将类分配给实现它的源文件，将可执行的组件分配给节点(通常指物理设备，如计算机)。

另外，在详细设计中还应包括所有类的属性、接口和操作的规格说明，规格说明应足够详细，它与各个模型一起为编码提供必要的信息。

在设计阶段使用了除用例图之外的所有 UML 图形，使用 UML 可视化建模系统在前一阶段绘制的图形基础上添加信息或增加新的图来完成设计。重点放在技术实施方案的细节部分，为实现阶段奠定基础。

实现，实现阶段实际上就是编码阶段，如果设计正确并且足够详细，编码应该是一项非常简单的工作，只须将最终的设计结果转化成某种编程语言即可。当然评审也是不可缺少的，这有助于提高代码的质量。在保证模型正确的基础上，可使用支持环境中的 UML 代码生成系统生成代码框架，将建模和编码过程有机地统一起来。在实现阶段，一般不应再去建立新模型，但有可能对设计阶段所建模型进行细化或修正。在修改设计模型时，要保持模型与编码的一致，以便将来易于维护。

测试。测试的目的是发现代码中的错误。错误包括功能性的（例如某个功能未实现，或实现的不正确）、非功能性的（例如运行太慢）、逻辑的（例如用户界面中的某些地方不合逻辑）。测试包括单元测试、集成测试、系统测试和验收测试。不同类型的测试使用不同的 UML 模型作为测试依据。(1)单元测试使用类图和类的规格说明。(2)集成测试使用组件图和合作图。(3)系统测试使用用例图来验证系统的行为。(4)验收测试由用户进行，验证系统的测试结果是否满足分析阶段确定的需求。可使用 UML 软件质量控制子环境进行软件的测试和软件质量的度量。

在测试阶段不可避免地会发现错误，对生成的代码进行修改后，可能造成模型和代码的不一致。此时可采用 UML 逆向转换系统将修改结果映射到模型，使得系统的扩充、增删和维护得以顺利进行，从而可以进行再次分析和修改，进行新一轮的开发。

结束语 目前，我们正在致力于 UML 集成化支持环境的开发，并已完成了其中的建模系统。希望通过深入开展有关 UML 的研究，研制基于 UML 的集成化支持环境，以期促进我国 UML 技术的发展、普及 UML 技术的应用，为我国软件产业的发展做出贡献。

参 考 文 献

1 Booch G. Object-Oriented Design With Applications. The Benjamin/Cummings Publishing Company, Inc. 1991

2 Booch G, et al. The Unified Modeling Language User Guide, Addison-Wesley Longman, Inc, Oct, 1998

3 Erikson Hans-Erik , Penker M. UML Toolkit, John Wiley & Sons, Inc, 1998

4 Flowler M, Scott K. UML Distilled, Addison-Wesley Longman, Inc, May, 1997

5 Jacobson I. Object-Oriented Software Engineering. A Use Case Driven Approach, The ACM Press, A Division of the Association for Computing Machinery, Inc, 1992

6 Jacobson I, et al. The Unified Modeling Language Development Process, Addison-Wesley Longman, Inc, December 1998

7 Rational Software Corp. UML Summary V1.1, September 1997

8 Rational Software Corp. UML Notation Guide V1.1, September 1997

9 Rational Software Corp. UML Semantics V1.1, September 1997

10 Rumbaugh J, et al. Object-oriented Modeling and Design, Prentice-Hall, Inc. 1991

11 Rumbaugh J, et al. The Unified Modeling Language Reference Manual, Addison-Wesley Longman, Inc, Dec 1998

（上接第94页）

这是一个传统的问题，有许多解决的算法，可直接应用于这类 CSCW 系统的实现中，故不再详述。

本文提出的方法，是针对用户实际情况的一些要求而提出并以一具体系统做为实例来实现的。虽然具体细节都与具体网有关，但此方法也可应用于其它微机网及分时系统。这种方法程序的难易程度虽据实际环境的要求而不一，但只要掌握和正确灵活地使用三种标志，解决复杂的问题会容易得多。

参 考 文 献

1 Hayer-Roth B. A blackboard architecture for control. Artificial Intelligence, 1985, 26: 251～321

2 Barthes T-P A. Computer-supporting cooperative work and knowledge management. In: Proc. of Workshop on CSCW in Design' 97. Bangkok: International Academic Publishers, 1997. 1～5