计算机科学2002Vol. 29№. 1

统一软件开发过程述评

Review of the Unified Software Development Process

麻志毅

(北京大学计算机科学与技术系　北京100871)

Abstract The Unified Software Development Process(USDP), published by a few masters in the software engineering field and Rational Software Corporation, and Supported by OMG, is attracting wide attention in the area of software engineering. After summarizing USDP, the paper introduces the phases and the core workflows of USDP in detail, then discusses positive influence of USDP on software development process, and points out some possible problems.

Keywords Software development process, Workflow, Object-oriented

1 引言

统一软件开发过程[1](Unified Software Development Process,USDP)是由统一建模语言[2,3](Unified Modeling Languang,UML)的缔造者们提出来的,并为对象管理组织(Object Management Group,OMG)所推荐。统一软件开发过程是缔造者们权衡了三十年的软件开发实践而形成的产物。它把当前处于分离的方法学集合于一起,称之为"统一方法"。它不仅统一了缔造者们的工作,而且使得开发方法得以统一,方法学家的工作得以统一。它也包含了很多其他人员和公司对UML作出的巨大贡献,例如它来自于数百个用户多年的现场经验以及Rational公司多年的工作成果。

USDP对于如何运用UML的概念进行软件开发提供了详细指导,它指导开发队伍安排其开发活动的次序,为各开发者和整个开发组指定任务,明确地规定需要开发的制品,提供对项目中的制品和活动进行监控与度量的准则。简言之,USDP将用户需求转换成开发软件系统的活动集合,适合于软件专业人员使用。

USDP中的重要因素有开发人员、项目、过程和工具。人员分为总设计师、开发者、测试者、管理者,还有用户、客户、其他关注者(投资者、市场人员、项目经理、产品线经理和法律代表等)等具体人员。项目是在生命期内面向机器、开发人员和关注者的所有制品的集合,每个周期产生"发布产品",若干周期为"一代",项目的成果是一个最终的发布产品。过程是将用户需求转换成产品所需要的活动集的完整定义,过程还提供活动指南以及对产生需求报告和文档的要求。工具是将过程定义中的活动进行自动化的软件。

2 统一过程概述

USDP是以用况(use case)为驱动的、体系结构(architecture)为中心的迭代的(iterative)和增量的(incremental)过程。

2.1 用况驱动

用况驱动的含义是在产品开发的各个阶段中都可以回溯到用户的真正需求,即它驱动开发过程,以用况为单位制定计划、分配任务、监控执行和进行测试等,将核心工作流结合为一体。用况的引入有助于模型之间的追踪和系统演化。

由用况能得到体系结构和其他制品。先选择在体系结构上具有重要意义的用况,实现那些关键的用况,构造出初步的体系结构,再逐步得到稳定的体系结构。通过枚举用况的不同执行路径,可导出测试案例和测试规程。通过用况还可估算系统性能、硬件需求和可用性,并能进行用户界面设计,用况也是编写用户手册的起点。

对用况的细化在一定程度上涉及系统的内部功能,对各用况进行完整的功能描述。细化时要区分用况中的三类事物:反映系统与参与者(actor)之间接口的事物,系统中的现实事物,对前两项进行协调和控制的事物。

2.2 以体系结构为中心

简单地讲,系统体系结构源自系统需求(用况模型),从不同角度描述要构造的系统,注重于系统的显著的静态和动态结构。用户和其他关注者要理解它,以对系统概貌的远景达到共识。要用系统的体系结构控制系统的开发、复用和演化。

系统体系结构描述系统中诸如子系统、依赖关系、接口、协作、节点和主动类这样的最重要的模型元素,而忽略其它细节,用这些元素指导系统开发。这些重要的元素由体系结构设计者完成,其余的由各设计者分别完成。如下是得到系统体系结构的简略步骤:

·在普遍了解用况之后,得到系统体系结构的粗略纲要(独立于特定用况和平台);

·关注关键用况;

·随着对用况的规约,导致更多成熟的用况,继续发现更多的系统体系结构部分。

系统的体系结构经过多次迭代,在精化阶段结束时,得到可执行的体系结构基线(Architecture Baseline)。它是系统框架,最初包括系统件、中间件、要复用的遗产系统、系统分布等情况。随后体系结构基线是由早期版本的用况模型、分析模型、设计模型等模型组成的集合(其中用况模型和分析模型较为成熟),关注于系统体系结构的内部发布版本,包括构造阶段结束时系统中的所有模型,它与最终系统(对客户发布的产品)有同样的骨干。从体系结构基线到最终系统之间要经过几个内部发布,体系结构基线的最后结果为一个模型集合和一个体系结构描述,其中包括最重要的用况及其实现。

建立可执行的体系结构基线是精化阶段的一个目标,进入构造阶段的体系结构基线应是坚实的。精化阶段结束时的体系结构基线在构造阶段变化不大,即体系结构基线到最终

麻志毅　副教授,主要研究方向为软件工程与软件工程环境、面向对象方法和自然语言理解等。

系统的体系结构之间的改动不大。

体系结构描述是模型的体系结构视图的集合，包括用况模型的体系结构视图（取用况的重要部分）、设计模型的体系结构视图、部署模型的体系结构视图和实现模型的体系结构视图。在开发过程中，要对体系结构基线进行改写，以凸现重要的设计问题，方便对问题的理解、讨论、解决和反馈，为开发者提供所需信息的高层视图，不遗漏大的技术问题。然而体系结构描述中不包括体系结构基线中的非体系结构级的元素，这是因为体系结构基线要对系统需求建模，以制订详细的开发计划，因此其用况模型中包含的内容比系统体系结构的要多。体系结构描述中包括：

·体系结构意义上的用况、子系统（不涉及子系统的隐含成分和私有成分，及其变种）、接口、类（量很少，主要为主动类）、构件、节点和协作；

·某些无法由用况描述的主要需求，例如性能、安全、分布和并发；

·平台、遗产、所用的商业软件、框架和模板机制等的简述；

·各种体系结构模式。

其实体系结构基线与体系结构描述同时开发，指导整个生命周期的软件开发。体系结构描述在系统生命周期中不断更新，以反映体系结构基线的变化，包括新的抽象类和接口的发现、为已有子系统增加新功能、可复用构件的版本更新、进程结构的重新组织、体系结构基线的新的表示形式等。

2.3 迭代与增量

简单地讲，迭代的开发过程是将整个项目划分为一系列的微项目，对每个微项目都进行一次迭代，即执行需求规约、分析、设计、实现和测试这五个活动（称为核心工作流），并要对每一次迭代进行计划和评价。迭代被组织到4个阶段中，请看图1。迭代的计划：

·早期迭代关注对问题和技术的理解，计划相对粗略，也更具有挑战性；

·精化阶段为构造阶段制定计划，其中构造阶段的第一次迭代最清楚，后面逐渐模糊，在后期要进行修订；

·前期得到的移交计划需要在构造结束时修订；

·计划时要考虑前期迭代的结果、新迭代的用况、与新迭代有关的风险和模型的最新版本集。

早期迭代要积累需求、迁移主要风险、明确问题、寻找解决问题的知识、进行项目定位、建立体系结构基线和为后续开发做详细计划，用较少的人力和物力进行此项工作；后期迭代要降低风险、实现构件和产生增量。

贯穿整个生命周期的迭代有主里程碑和副里程碑。在四个阶段的结束时都有一个主里程碑。主里程碑是管理者与开发者的同步点，以决定是否继续进行项目，以及确定项目的进度、预算和需求等。当一次迭代结束或完成一个建造时，可存在一个副里程碑，用以决定如何进行后续的迭代。在一次迭代结束时，把模型集合所处的具体状态定义为基线。

增量是由两次相邻迭代得到的内部发布之差，或是两个相邻基线之差，迭代的结果是增量。通过不断的迭代，不但获得增量，尽快获得反馈，而且也在不断地降低风险。

3 USDP 中的阶段和核心工作流

在 USDP 中，系统的生命周期由一系列的周期组成，每个周期产生的结果是用户产品的发布。每个周期有四个阶段，每个阶段又细分为若干次迭代，每次迭代都有一个核心工作流，请参见图1。

3.1 核心工作流

3.1.1 *捕获需求* 从客户、用户、计划者、开发者的想法（需求或用况）中搜取特征，形成特征表。对特征表中的每个特征给出简洁的定义，并描述其状态（提议、批准、合并和验证等）、实施的代价及风险、重要的程度以及对其它特征的影响等。

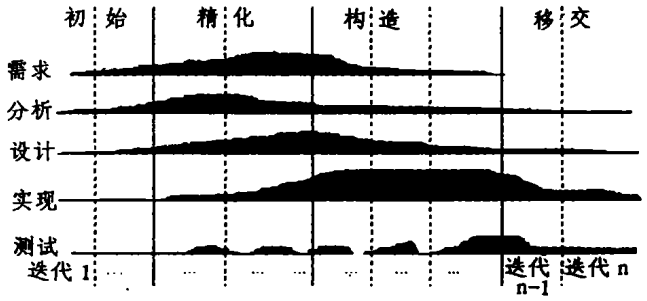


图1 USDP 中的核心工作流和4个阶段

先用领域模型和业务模型捕获需求。用领域模型捕获系统环境中最重要的对象，即业务对象（如帐目和合同等）、现实世界中的对象和概念以及事件。领域模型用 UML 的类图描述，此时类图中包含领域类（UML 类的构造型）及它们间的关系。业务模型由业务用况模型和业务对象模型支持。业务用况模型描述了具体的业务过程，其中含有业务用况和业务参与者。业务对象模型是一个业务的内部模型，它描述一个业务用况如何由工作者通过一组业务实体（如支票）和工作单元（一些业务实体形成的有机体）实现。用交互图和活动图显示业务用况的实现，即业务对象模型。

从特征表和领域模型（或业务模型）中产生的制品有参与者、用况、用况模型和体系结构描述。其中详述的用况模型由整体描述、一组图和详述的用况组成。如下为捕获需求阶段的活动描述：

序号	输入	活动	执行者	输出
1	业务模型或领域模型，补充需求，特征表	发现参与者和用况	系统分析员、客户、用户、其他分析员	用况模型$_{概述}$，术语表
2	用况模型$_{概述}$，补充需求，术语表	赋予用况优先级	体系结构设计者	体系结构描述$_{用况模型角度}$
3	用况模型$_{概述}$，补充需求，术语表	细化用况	用况描述者	用况$_{详述}$
4	用况$_{详述}$，用况模型$_{概述}$，补充需求，术语表	人机接口原型化	人机接口设计得	人机接口原型
5	用况$_{详述}$，用况模型$_{概述}$，补充需求，术语表	构造用况模型	系统分析员	用况模型$_{详述}$

工作流程：①→②→③→④ ⑤

3.1.2 分析 分析是对需求的精练和构造。此阶段产生的制品有：

·分析类。多为概念性的，粒度比类大，很少有操作和特征标记，要用责任定义其行为，在高层有概念性属性和关系。分析类可分为用于对系统和参与者之间的接口建模的边界类、对一些概念和现象的信息和相关行为建模的实体类，以及协调、顺序化、处理和控制其它类的控制类。

·用况$_{实现-分析}$。注重于功能需求，包括类图(使用分析类)、交互图(用协作图描述分析类之间的交互)、事件流$_{分析}$(用文本解释图)和补充需求(用文本描述的关于持久、分布、并发、安全特征、缺陷的忍耐等方面的非功能需求)。

·分析包。由分析类、用况$_{实现-分析}$以及其它分析包组成。一般地把支持一个特定的业务过程或参与者的一些用况组织在一个包中，或把具有泛化或扩展关系的用况组织在一个包中。

·系统体系结构描述$_{分析模型角度}$。主要包括由分析模型分解成的分析包和它们之间的依赖、关键分析类和实现重要及关键功能的用况$_{实现-分析}$。

如下为分析阶段的活动描述：

序号	输入	活动	执行者	输出
1	用况模型、补充需求、业务模型或领域模型、体系结构描述$_{用况模型角度}$	体系结构分析	体系结构设计者	分析包$_{概述}$、分析类$_{概述}$、体系结构描述$_{分析模型角度}$
2	用况模型、补充需求、业务模型或领域模型、体系结构描述$_{分析模型角度}$	分析用况	用况工程师	用况$_{实现-分析}$、分析类$_{概述}$
3	用况$_{实现-分析}$、分类$_{概述}$	对类分析	构件工程师	分析类$_{完成}$
4	系统体系结构描述$_{分析模型角度}$、分析包$_{概述}$	对包进行分析	构件工程师	分析包$_{完成}$

工作流程：①→②→③→④

3.1.3 设计 此阶段产生的制品有：

·设计模型。描述用况的物理实现的对象模型，注重于功能需求和实现环境对系统的影响。

·设计类。考虑与实现有关的因素，具体描述操作的参数、属性和类型等。

·用况$_{实现-设计}$。是设计模型中的一个协作，按设计类及其对象的术语，描述一个具体的用况如何实现和执行。由类图、交互图、事件流$_{设计}$(按对象或子系统的术语进行的文本描述)和与实现相关的需求组成。

·设计子系统。是组织设计模型制品的一种手段，用以描述大粒度的构件，由设计类、用况$_{实现}$、接口和其它子系统组成。

·接口。表示由设计类和子系统提供的操作。

·体系结构描述$_{设计模型角度}$。主要包括由设计模型分解的子系统、接口、依赖、关键设计类和用况$_{实现-设计}$。

·部署图。是一个对象模型，按在计算节点上的功能分布描述系统的物理分布。

·体系结构描述$_{部署图的角度}$。包括部署模型的体系结构方面的视图。

如下为设计阶段的活动描述：

序号	输入	活动	执行者	输出
1	用况模型、补充需求、分析模型、体系结构描述$_{分析模型角度}$	体系结构设计	体系结构设计者	子系统$_{概述}$、接口$_{概述}$、设计类$_{概述}$、部署模型$_{概述}$、体系结构描述$_{设计、部署模型角度}$
2	用况模型、补充需求、分析模型、设计模型、部署模型	设计用况	用况工程师	用况$_{实现-设计}$、设计类$_{概述}$、子系统$_{概述}$、接口$_{概述}$
3	用况$_{实现-设计}$、设计类$_{概述}$、接口$_{概述}$、分析类$_{完成}$	对类设计	构件工程师	设计类$_{完成}$
4	体系结构描述(从设计模型角度)、子系统$_{概述}$、接口$_{概述}$	设计子系统	构件工程师	子系统$_{完成}$、接口$_{完成}$

其中：活动1 把分析模型的分析包变为设计模型的子系统。活动2中的用况里的各个流应该各对应一个顺序图。

工作流程：①→②→④，②→③→②

3.1.4 实现 此阶段要实现设计类和子系统，由设计类生成构件，对构件进行单元测试，并对构件进行集成和连接，再把可执行的构件映射到部署模型。此阶段产生的制品有：

·实现模型。描述设计模型中的元素如何用构件实现，并描述如何按实现环境组织构件，也描述了构件间的依赖关系。

·构件。是对模型元素(如设计模型中的设计类)的物理封装，要把它映射到节点上。

·实现子系统。由构件、接口和其它子系统组成。

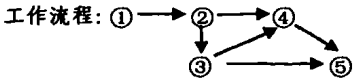
·接口。用于表示由构件和实现子系统所实现的操作，设计时的接口能用在此阶段。

·体系结构描述$_{实现模型的角度}$。包括由实现模型分解成的子系统、子系统间的接口、子系统间的依赖以及关键构件。

·集成建造计划。在增量的管理步中，每一个步的结果即为建造(Build)系统的一个可执行的版本。在一个迭代中，可能创建一个建造序列，该序列即集成建造计划。如下为实现阶段的活动描述：

序号	输入	活动	执行者	输出
1	设计模型、部署模型、体系结构描述设计模型、部署模型角度	实现体系结构	体系结构设计者	构件概述、体系结构描述实现模型、部署模型角度
2	补充需求、用况模型、设计模型、实现模型当前建造	集成系统	系统集成者	集成建造计划、实现模型连续的建造
3	集成建造计划、体系结构描述实现模型角度、设计子系统已设计、接口已设计	实现子系统	构件工程师	实现子系统建造完成，接口建造完成
4	设计类已设计、接口由设计类提供	实现类	构件工程师	构件完成
5	构件完成、接口	完成单元测试	构件工程师	构件已完成单元测试

其中：在活动3中，相应设计子系统中的每个类和每个接口要在实现子系统中构件实现。在活动4中，要列出包含源代码的文件构件，从设计类中生成代码和该设计类所涉及到的关系，以及为设计类提供操作的方法。构件提供的接口要与设计类提供的相一致。



3.1.5 *测试* 测试包括内部测试、中间测试和最终测试。特别是当精化阶段中体系结构基线变为可执行的、构造阶段中系统为可执行时和移交阶段中检测到缺陷时，要进行测试。在测试阶段，要产生如下制品：

·测试模型。主要描述在实现时可执行的构件怎样被集成测试者和系统测试者测试，以及描述系统的特殊方面（如用户接口、可用性、一致性、用户手册是否达到目的等）怎样被测试。测试模型是测试用况、测试过程和测试构件的集合体。

·测试用况。描述测试系统的方式。一般描述如何测试用况（或部分），包括输入、输出和条件。

·测试过程。描述怎样执行一个或几个测试用况，也可以描述其中的片段。

·测试构件。测试构件用于测试实现模型中的构件。测试时，要提供测试输入，并控制和监视被测构件的执行。用脚本语言描述或编程语言开发测试构件，也可以用一个测试自动工具进行记录，以对一个或多个测试过程或它们片段进行自动化。

·测试计划。测试计划描述测试策略、资源和时间表。测试策略包括对各迭代进行测试的种类、目的、级别、代码覆盖率以及成功的准则。

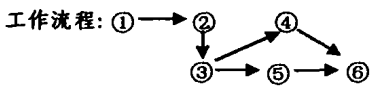
·缺陷。系统的异常现象。

·评价测试。对一次迭代，对测试用况覆盖率、代码覆盖率和缺陷情况（可绘制缺陷趋势图）进行评价。把评价结果与目标比较，准备度量。

如下为实现阶段的活动描述：

序号	输入	活动	执行者	输出
1	补充需求、用况模型、分析模型、设计模型、实现模型、体系结构描述模型的体系结构角度	计划测试	测试工程师	测试计划
2	补充需求、用况模型、分析模型、设计模型、实现模型、体系结构描述模型的体系结构角度、测试计划策略、时间表	设计测试	测试工程师	测试用况 测试过程
3	测试用况、测试过程、实现模型被测试的建造	实现测试	构件工程师	测试构件
4	测试用况、测试过程、测试构件、实现模型被测试的建造	执行集成测试	集成测试者	缺陷
5	测试用况、测试过程、测试构件、实现模型被测试的建造	执行系统测试	系统测试者	缺陷
6	测试计划、测试模型、缺陷	评价测试	测试工程师	测试评价

其中：在活动2中，对各建造要设计集成测试用况、系统测试用况和回归测试用况。在活动4中，对一个迭代中的各个建造执行集成测试。当集成测试满足当前迭代计划中的目标时，要进行活动5。



3.2 USDP的四个阶段

下面对USDP的四个阶段要做的工作进行简述。

3.2.1 *初始阶段* 本阶段确定所设立的项目是否可行，具体要做如下工作：

·对需求有一个大概的了解，确定系统中的大多数角色和用况，但此时的用况是简要的，这些用况构成了一个简化的用况模型。对给出的系统体系结构的概貌，细化到主要子系统即可。

·识别影响项目可行性的风险。

·考虑时间、经费、技术、项目规模和效益等因素。

·关注于业务情况，制订出开发计划。

3.2.2 *精化阶段*

·识别出剩余的大多数用况。对当前迭代的每个用况进行细化，分析用况的处理流程、状态细节以及可能发生的状态改变。细化流程时，可以使用程序框图和合作图，还可以使用活动图、类图分析用况。

·对风险的处理。考虑需求风险、技术风险、技能风险和政策风险。

·进行高层的分析和设计，并作出结构性决策。所产生的基线体系结构包括用况列表、领域概念模型和技术平台。以后

的阶段对精化阶段建立的体系结构不能进行过大的变动。体系结构基线的稳定是精化阶段结束的准则。

·为构造阶段制订计划。精化阶段完成，意味着已经完成了如下的任务：用况完全细化并被用户接受；完成概念验证；完成类图；开发人员能给出项目估算(可分为精确、人月和无法估算)；基于用况考虑了所有风险(可分为高风险、可能的风险和不可能的风险)，并制订了相应的对策和计划；对用况标出优先级(可分为必须先实现、短期内实现和长期实现)。

3.2.3 *构造阶段* 识别出剩余的用况。此阶段的每一次迭代开发都针对用况进行分析、设计、编码(如对类、属性、范围、函数原型和继承的声明等)、测试和集成过程，所得到产品满足项目需求的一个子集。由于在精化阶段已经完成软件设计，此时各项目组可以并行开发。

在代码完成后，要保证其符合某些标准和设计规则，并要进行质量检查。对于新出现的变化，要通过逆向工具把代码转换为模型，对模型进行修改，再重新产生代码，以保证软件与模型同步。

此阶段要建立类图、交互图和配置图；如一个类具有复杂的生命周期，可绘制状态图；如算法特别复杂，可绘制活动图。

3.2.4 *移交阶段* 完成最后的软件产品和最后的验收测试，并完成用户文档以及准备对用户培训等。

4 评价与展望

因为被 OMG 采纳的 UML 只是一种建模语言，并不包含过程指导，针对于此，UML 的缔造者们给出了一种以 UML 作为建模语言进行软件开发的过程指导 USDP，但其内容不是 UML 固有的组成部分。当然 USDP 也可以作为其它建模语言的过程指导。该软件开发过程是 UML 的三位作者在研制 UML 时一直在头脑中思考的，因而很切合 UML 的特点。

由于 USDP 是集众家之所长而产生的，且其主要执笔人 I. Jacobson，G. Booch 和 J. Rumbaugh 在面向对象领域影响很大，它又被 OMG 所推荐，加之它是针对当前最为流行的建模语言之一 UML 而制订的，USDP 一直为软件工程领域所关注。

USDP 是以用况为驱动的、体系结构为中心的、迭代增量的开发过程。以用况为驱动可使产品开发的各个阶段都可以回溯到用户的真正需求，将核心工作流结合为一体。以体系结构为中心可控制系统的开发、复用和演化，如纸上谈兵的分析和设计有一些缺点，而可执行的系统体系结构可让工作者们提供反馈的演示版。迭代增量的开发过程可控制开发的风险。

尽管现在该过程已经得到了一定的应用，但是 USDP 能否为面向对象领域的多数软件开发人员所采用，尚需经过实践的考验；至于能否统一当前的各种软件开发过程，现在还不能断言，主要原因有如下几点：

1)在 USDP 中要对各用况进行完整的功能描述，对用况进行细化，这就要在一定的程度上涉及系统的内部功能，例如用况细化时要区分用况中的接口的事物、现实事物和控制事物。而在 UML 中对用况的定义是：用况是对一个行为序列的描述(包括变体)，系统执行该序列要为参与者产生一个可观察到的结果。即该定义描述了参与者和系统之间的交互，按其用意对用况的描述并不深入到系统的内部。另外对用况的细化要进行到什么程度?若把握不住，就会以逐步求精的方式把所有的分析结果都在用况中表现出来，这与传统的功能分解法建立的需求分析模型没有什么太大的区别。

2)过程定义了谁在什么时间做什么，以及如何达到特定的目标。一个有效的过程要为合乎质量要求的软件开发提供有效的指导，以降低风险，增加可预见性。无论使用什么样的建模语言，所有与软件开发相关的人员都应遵循这样的过程作为指导，而以往的过程还做不到这一点。统一软件开发过程是由 Ericsson 方法到 Objectory 过程，再到 Rational Objectory 过程，进而到 Rational 统一过程[4](Ratioanal Unified Process，RUP)演变而来的。1998年6月，Rational 发布了 RUP 5.0，其中的很多内容在 USDP 中第一次得以公布。USDP 是针对 UML 而提出来的，在上述方面取得了长足的进步，但要取代其它软件开发过程，还有待于实践的检验。

3)UML 已被 OMG 采纳，在面向对象领域影响面很大。由于 UML 使多种方法走向统一，其功能较为丰富，表达力很强。但随之其复杂性也大大增加[5]，影响了很多用户对它的掌握与使用。因为 USDP 是针对 UML 制订的，UML 的发展状况势必会影响到 USDP。尽管 USDP 也可以作为其它建模语言的开发过程，但这样的应用并不常见。

4)USDP 是可裁剪，但如何裁剪，USDP 给出的指导过于简洁。正像作者所指出的那样，裁剪时要考虑系统的大小、系统应用的领域、系统的复杂性、开发人员的经验与技能，以及管理者的水平等众多因素。这意味着应用 USDP 不是一件简单的事情。

无论 USDP 能否统一当前的各种软件开发过程，USDP 确实是一种值得学习的软件开发过程框架。它毕竟是作者们权衡了三十年软件开发实践的产物，包含了数百个用户多年的现场经验以及 Rational 公司多年来的研究成果，并已被较为广泛地使用。它较为详细地阐述了一系列的过程，具体地表现出了一个完整的软件开发周期，在这方面它超越了其它的面向对象的分析和设计方法。相信随着面向对象和构件技术及方法的发展，USDP 会更加成熟，成为主流的软件开发过程。

参 考 文 献

1 Jacobson I，Booch G，Rumbaugh J. The Unified Software Development Process. Addison-Wesley Object Technology Series，1999

2 UML documentation version 1.3. http://www.rational/uml/index.jsp

3 Booch G，Jacobson I，Rumbaugh J. The Unified Modeling Language User Guide. Addison-Wesley，1999

4 Rational Unified Process. http://www.rational/uml/index.jsp

5 邵维忠，梅宏. 统一建模语言 UML 述评. 计算机研究与发展，1999，36(4)：385～394