敏捷建模对统一过程的改造实践

作者：金洁羽

1.绪论

优良的软件过程可以为开发高质量的软件提供有效的实践方案。随着软件开发复杂性的日益增强及软件行业规范化管理经验的逐渐积累，通过依托某种软件过程在开发中产生恰当制品、科学有效地控制开发节奏与步骤，从而合理调配与使用开发资源、满足开发愿景，已经成为提高软件生产率、确保如期按质地提交软件成果的重要途径，而这一观念也正被越来越多的软件开发公司所认同和采用。

2.软件过程和敏捷建模

经典软件工程理论所阐述的软件过程，特别是瀑布模型，由于其较强的理想化环境条件假设，所以如果在实际软件开发项目中采用，往往会与实际情况产生较大脱节，使得实施效果大打折扣，故而往往只以具有理论参考意义的面孔出现。而当前在业界，既有完整理论，又有较强可操作性的软件过程，则以统一过程（theUnifiedProcess,简称 UP）和极限编程（theeXtremeProgramming,简称 XP）最为令人瞩目。

2.1 统一过程与极限编程及其思想观念

2.1.1UP 与 XP 的特征

统一过程作为一种重量级的指导性的软件过程，其基本特征体现在用例驱动、以架构为中心、迭代和增量开发等几方面。根据对统一过程生命周期的不同划分方法，统一过程还分为企业统一过程（EUP）、Rational统一过程（RUP）等不同子类型。而极限编程则不同于统一过程，它在本质上属于更为敏捷的一种软件过程，并由一系列简单却互为补充的实践所组成。

2.1.2 统一过程的优势

虽然比较 XP 而言，UP 更为传统一些，其过程控制灵活度也相对较弱，但由于这种软件过程非常适用于复杂、需求多变、开发难度大的情况，同时也可以根据项目特点进行适当裁剪，所以仍然被许多软件企业所广泛采用。特别是，对于一些在软件过程控制方面依据 UP 原则已经形成较为固定模式、同时又注重以各阶段的指定制品控制开发节奏的公司而言，如何在保持UP基本方法的前提下,提高UP项目开发的敏捷性则是一个非常现实而重要的课题。

2.2 以敏捷建模改造统一过程的可行性

2.1.3 敏捷建模与统一过程在实践中的内在联系

事实上，敏捷建模（AgileModeling，简称 AM）所倡导的一系列实践中有许多做法与 UP 的实践是不谋而合的，有的还加强或改进了 UP 的某些实践。比如：

UP 和 AM 都强调“项目关系人积极参与”的重要意义，甚至允许项目关系人参与建模。

UP 和 AM 都强调“用代码验证”、“并行创建多个模型”。即使是 UP，在必要情况下也可调整决定同时执行源于不同规程的活动。

UP 和 AM 都遵循“增量建模”的实践，但 AM 通过减小增量的幅度，改善了 UP 的迭代实践。

AM“有危害时才更新模型”的实践改进了 UP 及时同步各阶段不同制品的要求。

AM“使用合适的制品”的实践改进了 UP 对 UML 建模制品的过分依赖。

AM“集体所有”的实践改进了 UP 项目中有关配置管理的观念，通过营造开放、交流的团队文化，使所有项目成员都能访问和修改各自想处理的制品，包括模型和文档。

AM“使用最简单的工具”的实践拓展了 UP 只注重使用 CASE 工具的局限。

2.1.4 实践敏捷建模的主要原则

与 UP、XP 相比，AM 本身则是一种基于实践的、不完整的、有序与混乱并存的软件过程。通常，软件的开发可将 UP、XP 等作为基础软件过程，用 AM 增强这些更加完整的软件过程。

AM 的概念吸收了敏捷开发联盟（AgileSoftwareDevelopmentAlliance）所倡导的若干原则（限于篇幅，这些条款将不在文中详述），并形成了自己的一系列原则与实践。纵观 AM 论坛发起人 ScottW.Ambler 所提出的涉及 AM 的 11 条核心原则和 13 条核心实践，笔者认为，体现 AM 精髓的原则主要集中在如下几个方面：

（1）明确最终目标：即软件本身才是应当确保的工作目标；

（2）快速迭代反馈：明确各阶段问题焦点，快速修订前一阶段过程中的制品，并推回后一阶段；

（3）多种模型建模：即要勇于突破传统软件过程所规定的建模工具的约束，根据效果特点采用适用的建模模型描述软件；

(4)简化工作过程:任何阶段都要以最简单的解决方案来达至工作目标,不要追求形式、不要过度构建软件。

虽然按照 Ambler 的观点，必须完全执行所有 11 条核心原则和 13 条核心实践才能算得上是在敏捷建模，但笔者认为，由于 AM 本身就是一种不完整的软件过程，况且 AM 本身必定会在行业实践中进一步得到充实，所以可以认为，在目前阶段，实践上面总结出来的四条原则即可基本反映出 AM 的精髓。

3.太原同城系统开发中的敏捷建模实践

在上述方针的指导下，可以利用敏捷建模对统一过程施行改造。以下将以山西省太原市人民银行同城票据清算系统 v2.0（以下简称“太原同城系统”）为例，说明应用 AM 原则与实践在改造一个 UP 开发项目过程中的一些体会。

3.1 太原同城清算系统介绍

3.1.1 开发背景

太原同城系统是在金融体制改革的形势下，由北京市泰通电子技术公司承担开发的，在太原市辖区范围内建设的一个连接该市各个商业银行和与人民银行清算中心的票据实时清算系统。通过实时处理贷记、借记票据交换业务，可以改变传统的票据手工交换方式，以电子流代替票据流，使资金可即时抵用，为各商业银行提供统一、快捷、安全、可靠的资金清算渠道，也为人民银行提供了监控资金流量与流向的现代化手段。

3.1.2 系统体系结构

太原同城系统是一个基于交易中间件、复合平台、多语言联合开发的三层 C/S 结构系统。其三层架构分别为：

(1)数据层：运行在人行结算中心的数据库服务器上，OS为TurboLinuxEnterprise8.0，DBMS为Oracle9iEnterpriseforlinux。

(2) 中间层：运行在人行结算中心的应用服务器上，OS为TurboLinuxEnterprise8.0，用C++语言实现了核心商业服务，开发工具选用BorlandC++BuilderX1.0Enterprise。

(3) 表示层：运行在各商业银行的 PC 前端机（SCOUNIX5.05 平台，270 台）及人行结算中心的 PC 前端 机(Win2000/RedflagLinux4.0 平台，5 台)上。商行前端程序用 C 语言实现；人行前端程序用 JAVA 语言实现，开发工具选用 BorlandJBuilder7Enterprise。

上述三个层次分别安装了东方通科技公司的消息中间件 TongEASY4.5forLinux/SCOUNIX/Windows，整个系统的事务处理功能即由它保证。TongEASY 作为一个基于三层C/S体系结构的中间件，其构成的交易管理平台提供了交易管理、负载均衡、应用调度等功能。其包含的通讯管理模块还提供了可靠的数据传输、网络监控、流量控制等功能。

3.2 太原同城系统开发中的敏捷建模实践

本人作为太原同城系统的项目经理，按照上文所述观点，将敏捷实践中“明确最终目标、快速迭代反馈、多种模型建模、简化工作过程”这四方面的内容与实际开发的过程控制进行了结合。以下是对其中这几方面实践内容的总结。

3.2.1 太原同城系统的快速迭代与增量式开发实践

这一实践体现了“快速?????”UP。按照Rational公司对UP的定义，软件开发的生命周期被划分为初始、细化、构造、交付四个阶段，每个阶段结束于定义良好的里程碑——即某些关键决策必须做出的时间点。为了更好地控制变更、减少开发风险，太原同城系统的开发

遵循了RUP所规定的从一个迭代过程到下一个迭代过程的递增式增长，从而形成了最终系统。具体而言，这些迭代过程包括：

（1）第一次迭代（历时一个半月）：

跨越 UP 所定义的初始阶段。该次迭代实现的 UP 规程主要包括：初步业务建模、捕获业务需求、建立开发环境等。通过该次迭代，完成了架构方案的确定，实现了部分子系统中最关键的业务用例的分析设计与编码、测试－－比如票据发送、票据接收、票据文件生成等。同时，通过测试比较，项目组明确了系统的体系结构，即：立足于基于消息中间件的三层 C/S 结构系统。此外，还根据上述体系结构的特点，确定了开发语言与开发工具。值得一提的是，该次迭代的初衷并非单纯针对太原同城系统，但太原同城系统事实上成为了此次迭代过程的第一个受益者。

（2）第二次迭代（历时两个半月）：

跨越了 UP 所定义的细化阶段和构造阶段。该次迭代实现的 UP 规程包括业务建模、需求分析、体系结构设计、编码实现，它所包括的数次小的迭代过程分别侧重于分析、设计和编码实现。首先，通过本次迭代，基本完成了对所有概要级别用例、用户级别用例及大部分子功能级别用例的分析，同时，还完成了类设计、基本数据静态结构设计与数据动态流向设计。系统架构的整体搭建也在此阶段完成。之后，在上述架构的基础上实现了相应的编码工作。

（3）第三次迭代（历时两个半月）：

跨越了 UP 所定义的构造阶段和移交阶段。该次迭代实现的 UP 规程包括补充业务建模、补充需求分析、补充编码实现、系统测试、运行和支持等。通过该次迭代，完成了对各级别用例的修改、补充，补充完善了各功能模块的代码，完成了系统测试和部署准备。

当然，系统最终迭代次数的变更结果取决于太原同城系统的实际部署效果和部署后用户的意见反馈。

同时，系统在进行某一阶段的建模工作时，还注意尽量避免在该阶段只专注于一个制品的制作。比如，在用例建模阶段，还同时进行了用户界面原型的设计工作；在数据建模阶段，还将设计会议包括进来。

3.2.2 太原同城系统的多种模型建模实践

太原同城系统的过程控制实践很注重改造系统启动时组织内部因沿袭 UP 思维方式而可能存在的不符合 AM 原则的文化障碍。比如，根据 AM“多种模型建模”的精神，太原同城系统的建模过程就没有受限于 UML 规定的若干建模制品。

虽然UML定义了一系列的建模制品，但很显然，完整创建其定义的所有建模制品是没有必要的。另一方面，UML建模制品至少在分析需求时并不完备，也存在盲点。比如，UML无法满足用户界面建模需求、UML的活动图无法描述一个信息存储的位置信息等等。因此，在太原同城系统的开发中对UML制品根据需要进行了取舍。针对UML建模制品的不足，还补充了其它一些制品，包括一些UML问世之前就存在了的建模制品（如数据流图），甚至包括一些有

独创性的建模制品（如数据流向设计）。下面是太原同城系统分析设计过程中一些制品的制作情况描述：

⑴编写有效用例，透彻描述和分解需求规格说明书中提出的各项功能需求。

比如，对于“中心日初始化”这一功能而言，我们需要清楚地说明人行结算中心在每个工作日业务开始时，日初始化功能涉及到的主执行者、目标、前置条件、触发条件、主要场景、意外情况处理等细节情况。其有效用例主要部分摘要如下：

“中心日初始化”处理用例

n 主执行者：人行结算中心系统管理员

n 语境中的目标：人行结算中心系统能开始处理税票信息。

n 级别：用户目标。

n 项目相关人员和利益：

1. 人行结算中心操作员

2. 人行结算中心业务领导

3. 人行结算中心

n 前置条件：人行结算中心操作员、系统管理员或业务领导登录成功。

n 触发事件：人行结算中心系统工作人员登录成功后开设新工作日。

n 成功保证：人行结算中心新工作日开设成功，并且工作状态进入日间状态。

n 最小保证：人行结算中心系统工作人员登录验证。

n 主成功场景：

1. 人行结算中心前一工作日日终判断。

2. 人行结算中心本工作日日初注册：人行结算中心系统获取新的工作日日期；人行结算中心该工作日由日初状态进入日间状态。

n 扩展：

*a. 任意时刻，系统崩溃：（略）

*b. 任意时刻，网络连通出现故障：（略）

1a.结算中心前一工作日日终尚未完成：

继续进行结算中心前一工作日日终处理，直到日终成功。

1b.结算中心前一工作日日终已经完成。

继续进行新一工作日的日初处理。

2.手工输入的新工作日日期不符合要求：提示修改工作日日期，直到合格。

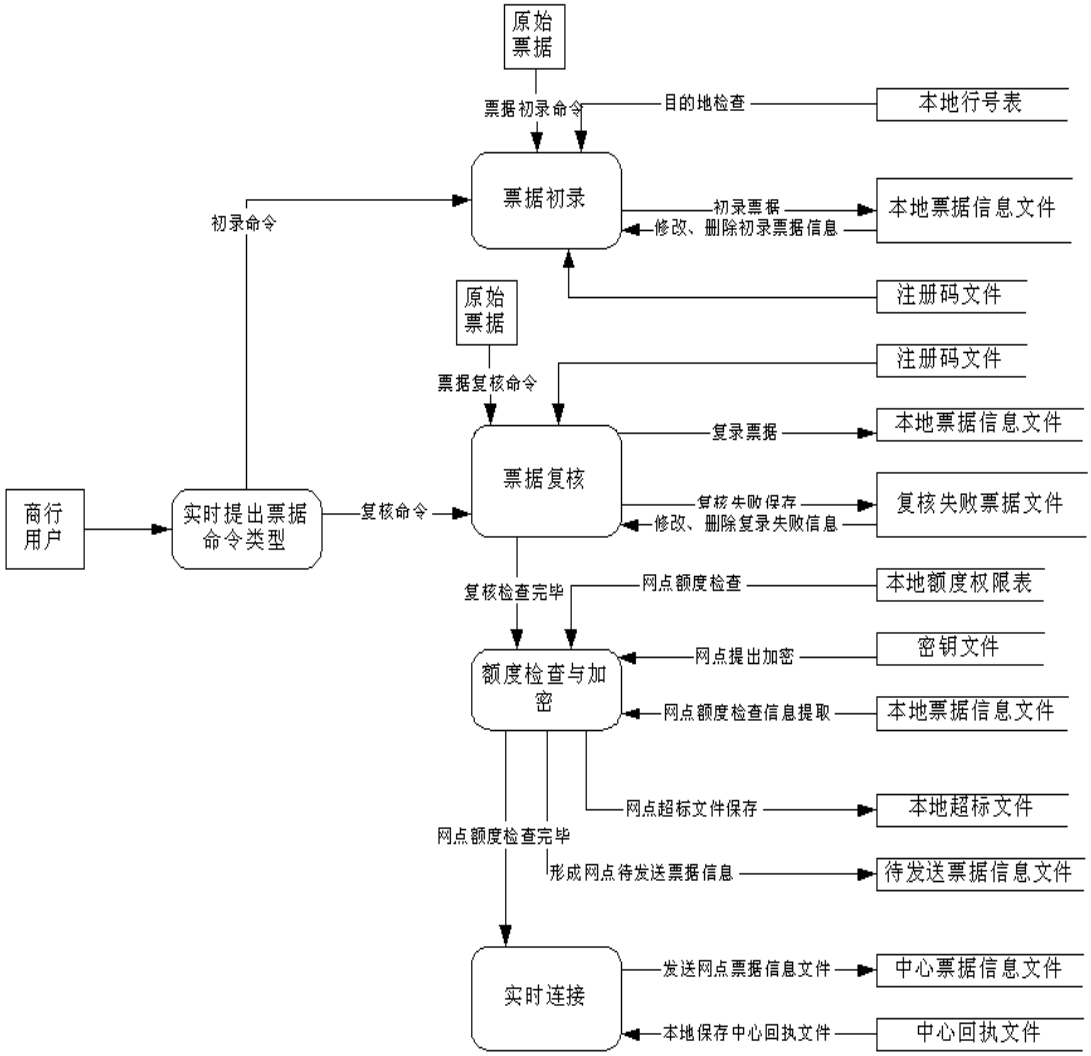
n 使用频率：每个工作日一次

太原同城系统中包括上述用例在内所有用例的设计，其设计风格都与AlistairCockburn 在《WritingEffectiveUseCases》一书中推荐的风格相同。其中，下横线部分是准备继续扩展描述的下一级用例，为省篇幅本文从略。

这里，应当突破UP声称的对用例的认识局限，意识到用例并不足以驱动所有事情。事实上，用例只是全部需求，甚至只是全部功能需求的一部分，对诸如用户界面需求、商务规则、非功能需求的描述，仅靠系统用例是不够的，有必要以基本界面原型、界面流程图、CRC卡等加以补充。

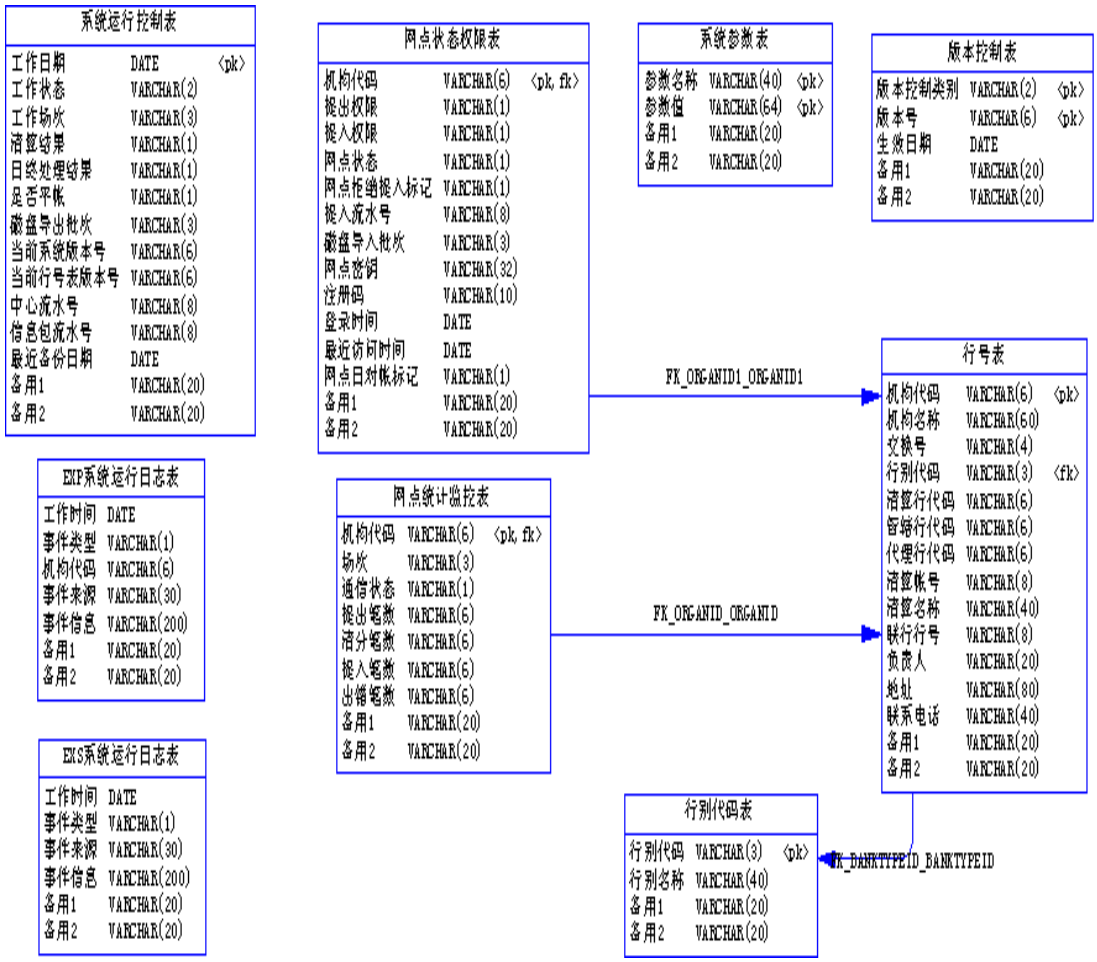
⑵针对“概要级别用例”、“用户级别用例”及“子功能级别用例”使用数据流图进行逐层分析，了解各用例所涉及的数据源点与汇点之间的关系。

通过数据流分析这种传统的分析手段，我们可以汇总出数据字典，并作为静态数据结构设计的依据。下面的数据流图实例就是太原同城系统中针对用户级别用例－－“实时提出”所做的分析。依托数据流图，“实时提出”功能所涉及到的各个数据存储、各个数据源点和终点、各个加工都一目了然。



⑶根据数据流图进行数据建模。

在 UML 中，数据模型并没有很好的建模制品进行表达，但我们有必要提供数据模型制品，并为后面诸如数据库表静态结构设计、UML 类图设计等工作做准备。以下展示的，就是人行结算中心数据建模工作的一小部分结果。



项目组为完成该项设计，专门召集了数据建模会议。会议方式是项目经理组织全体项目成员集体讨论，集思广益。最终讨论结果以 PowerDesigner 为数据建模工具当场记录和修改，主要涉及库表结构定义、各数据项定义、表间关系定义、主外键定义等内容。不论是在当时的建模会议上，还是以后进行修改时，PowerDesigner 在对数据建模结果进行快速修正和记录方面都体现了强大的功能。

⑷参照静态数据结构、数据流图、用例描述获得数据流向设计图。

这一独创的制品具有与 UML 时序图异曲同工的效果。但如此表达更符合程序员的编码习惯，更有利于实现从伪代码到真实代码的转化。至于设计范围，主要依据用例描述来确定。以下是“中心日初始化”这一子功能级别用例的数据流向设计实例：

“中心日初始化”数据流向

查询系统运行控制表

if 有记录

{

定位到工作日期最晚的一条记录

if 工作状态<>0<日终>

{

退出日初始化处理

}

手工选择工作日期××××-××-××

判断该日的记录是否已经存在

if 存在

{

提示并退出日初处理

}

判断该日是否小于库中的最大日期

if 小于

{

提示并退出日初处理

}

在系统运行控制表写入一条记录（手工选择工作日期××××-××-××工作状态 1<日初>；工作场次 1；是否平帐 0<未清算>）

清空业务库表：网点日提出票据表、网点日提入票据表、网点日提入错误票据表、网点磁盘日待提入票据表、网点磁盘日提入登记表、网点日清算表、清算行日清算表、管辖行日清算表、清算帐户日清算表

更新网点状态权限表所有记录的网点日对帐标记为 0<未对帐>，提入流水号为 1

更新网点统计监控表所有记录的提出笔数、清分笔数、提入笔数、出错笔数为 0；场次为 1；通信状态为 0<断开>

行号表是否需更新：

从版本控制表中查询>当前行号表版本号的记录，如果有记录，依次判断该记录的生效日期是否到期

if 有到期记录

{

备份当前行号表到备份行号表

更新行号表

更新网点状态权限表、网点统计监控表

修改系统运行控制表的当前行号表版本号

}

在系统运行控制表中修改记录（工作日期××××-××-××工作状态 2<日间>；工作场次 1；是否平帐 0<未清算>）

}

else

{

手工选择工作日期××××-××-××

在系统运行控制表写入一条记录（手工选择工作日期××××-××-××工作状态 2<日间>；工作场次 1；是否平帐 0<未清算>）

}

在数据流向设计中，总的设计方式是参照流程顺序和时间顺序。其中，横线部分是此制品涉及到的数据库表。对相应库表的重要改动内容也同时记录在旁边，状态字段的内容改动其含义记录在“<>”中。

可见，通过上述各环节的分析设计工作，系统的制品已大大突破了 UML 所定义的制品范围。虽然这其中所完成的基本界面原型设计、界面流程图等制品尚未加以罗列，事实上其存在与制作对于系统的透彻分析是很有必要的。

3.2.3 太原同城系统的简单化和目标明确化实践

当然，太原同城系统的开发实践还注意了“简化工作过程”与“明确最终目标”这两项AM精神的贯彻。比如，建造制品时对UML明确定义的部分制品的舍弃、在类图的制作过程中仅着重核心类的建模、在各阶段建模过程中注意点到为止，使制品量与其对开发的指导价值

相匹配……等等，都体现了简单化的用意。而所有这一切，其实都是为完成最终目标－－目标软件本身而服务的。

笔者认为，这两方面的实践虽然相对更加哲学化、抽象化一些，但在开发过程控制中依旧要始终加以留意，使之真正起到指导思想的作用。

4. 总结

综上所述，对于统一过程而言，敏捷建模是在保持统一过程原有基本优点的前提下，加强这一重量级软件过程敏捷程度的有效途径。相信，会有越来越多的统一过程软件开发实践将受益于对敏捷建模精髓的汲取，而太原同城系统开发的成功实践就是其中一个很好的例子。