软件的传统生命周期开发法与原型化开发法

北京航天大学软件工程研究所　　郭　江

摘要:系统开发生命周期(SDLC)的概念已经成了计划、实施、和控制系统开发的一个重要概念。但这种开发方法还有许多缺点,许多不能令人满意的系统仍在开发之中。原型化方法虽然不是一个完美的解决方法,但已经有力地补充和改善了传统的软件开发方法。本文回顾了这两个软件开发方法以及各自的局限性,并讨论了怎样将这两种方法结合起来以便有一个更为有效的软件开发过程。

一、引言

在复杂的商业软件系统开发中,预先确定用户的需求是相当困难的,因为用户通常在看到一个系统的工作模型之前并不确切知到底需要什么。这样就提出了原型化方法,即一种开发和评价系统工作原型的软件开发方法,这种方法就是逐步用来帮助设计者迅速而准确地定义系统需求和功能。因而可以把原型化方法作为系统开发生命周期(SDLC)中的一个有效的组成部分,本文描述了传统的系统开发生命周期法和原型化方法的限制,并阐述了怎样将这两种方法结合起来以改善系统的开发过程。

二、传统的 SDLC 模型

传统的软件项目开发过程是线性地通过了一个系统的开发生命周期(SDLC)。即沿着一系列定义好的、文档化的、顺序的阶段来进行。SDLC 概念的出现,主要是作为项目的管理工具用来计划,实施和控制系统项目的开发。尽管已经开发了各种 SDLC,但典型的 SDLC 主要由下面的六步构成:

<1>分析
<2>需求说明
<3>设计
<4>实现
<5>安装和测试
<6>维护和进一步的开发

这种传统的模型是由文档驱动的。即,每个阶段通常都产生一个可用于下一个阶段的产品(通常是一个文档)。这个过程是以完整的系统分析开始,这个分析涉及到用户、开发者、以及管理者对系统项目的范围及可行性的全面评价。一个典型的系统分析通常产生出系统项目的简短描述、代价/效益分析、基本预算和计划。

在需求说明阶段,用户和分析员的文档是完整、准确而又详细的,是用户需求的集合。典型的说明包括了系统功能的详细描述以及有关的非功能性需求,包括系统的运行、可靠性以及物理限制。这些用户需求构成了以后的生命周期设计和实现等阶段的基础。

设计阶段的目标是去产生系统的一个详细的逻辑定义和物理定义,以便以之为基础来开发程序。设计必须为系统描述所有的数据处理结构和数据元素。在考虑系统的用户接口、数据库、系统输出、以及外部接口等细节时都要把它们文档化以便记录。还要考虑的是开发系统的硬件以及将来的运行环境。

通常,实现是在需求和设计文档完成之后才开始的。在实现阶段需要为系统开发、购买程序,然后进行测试。如果系统说明是完整的,而设计又是高质量的,那么实际的实现就相对比较简单了。但是,实际的情况是系统设计经常是有漏洞的或者是不完全的。这样,重要的设计决策就必须在程序开发期间完成了。这个问题是传统的 SDLC 方法的一个严重缺陷。如果在实现期间不能确定并纠正系统说明和设计中的错误和漏洞,那么它们就可能随着进入系统中。

在前面的阶段完成之后,系统进入生产阶段,这个阶

段涉及到了转换现存的系统、培训用户、填充数据库(即往其中输入数据)和计算机文件、以及分发文档等工作。这时应执行一个生产系统测试以验证系统确实满足用户的期望和需求。最后,需要对系统进行维护以确保其运行和操作的正确性。

三、传统的 SDLC 模型的局限

传统的 SDLC 文档驱动方法已经给软件开发过程产生了一些非常重要的原则。许多系统在这些 SDLC 原则的指导下成功地开发出来了。但是信息系统(IS)的开发者和用户意识到:尽管已经确立了很好的开发技术,但还是产生了许多不令人满意或有缺点甚至是没有价值的系统。尽管系统用户参与了需求定义而且还查看了系统设计,但他们还是经常对完成了的系统感到失望。

由于用传统的文档驱动的开发方法产生的系统不仅需要大量人力和高昂的代价,而且还有失败的风险,因此信息系统的开发者已经意识到传统的 SDLC 有着严重的缺陷和局限性。实际上,由于传统 SDLC 的局限性,使用文档驱动的开发方法来产生既经济有效而又复杂的系统是非常困难的。对许多面向用户的应用(如决策支持系统和数据库应用)来说,需求通常是不清楚的,因而传统的预先确定需求的方法几乎没有什么帮助。实际上,在开发出系统之前,要预先定义出详细的需求和系统设计不仅困难和费时费钱,而且通常也是不可能的。另外,让用户去理解书面的说明也不是和用户进行沟通的最有效方式。

四、原型化方法的生命周期模型

原型化方法对复杂的工程项目而言早已成了一个流行的设计工具,而且已经成为传统的软件开发方法的一个有力的补充工具。原型,即系统的工作模型,是在最后系统实际完成之前建立的。由于能在系统完全实现之前显示系统的某些特征,因而原型化方法给用户提供了一个在系统开发过程的早期就与系统进行交互的机会,并且能帮助用户理解和澄清他们所需要的系统功能。而且这种方法也有助于系统设计者来迅速测试他们的设计。

在原型化方法中,系统需求和说明的定义是一个反复重复的过程;需求和说明随着用户和开发者与原型的交互而被推敲修改。由于用户在所有阶段都参与了开发,因此原型可能潜移默化地增加了用户的自信心,增加了对开发系统的满意程度,增加了最终系统接受和成功的可能性。

原型开发生命周期模型由下面六个步骤组成:

<1>确定初始的用户需求

<2>开发初始系统设计

<3>开发、评价、和修订原型

<4>系统实现

<5>安装和测试系统

<6>维护和进一步开发系统

原型化生命周期和传统方法的主要差别在于系统需求和说明的开发。和传统方法一样,用户和系统分析员一起来定义初始系统需求和说明。但是,他们不能企图确定所有用户需求和系统说明的各个细节。他们应用一系列原型来沟通交流,一起反复推敲来完成需求和说明。

从初始需求和说明开始,分析员就应该使用第四代语言(4GL)、应用生成器、或特定的原型化工具来构造系统模型,即原型。由用户来评价原型的特征和性能。如果原型不能满足用户的期望,分析员和用户就必须重作需求说明,并对原型进行改进以满足需求。分析员和用户连续不断地开发出系列原型直到用户认为原型准确地反映了他们的需要为止。

确认后的原型就成为随后的生产系统设计和实现的工作需求说明。这个原型取代了在传统 SDLC 早期阶段所产生的详细说明文档。原型本身既可以成为生产系统全面开发的基础,又可以在生产系统用另一种程序设计语言来实现时抛弃掉。以后的生命周期阶段就象传统的方法一样..通过实现、安装、测试、维护以及进一步的开发等五个阶段。

五、软件原型化方法的问题

原型化方法不应是用来代替传统的 SDLC 的;而且应把它作为一种补充的方法,它仅能在某些阶段改善生产率。尽管原型化方法在简化系统开发过程以及形成系统开发过程流水线等方面有很大的潜为;但是它的使用

应象其它开发工具一样,必须全面理解和严格管理。由于原型化方法的动态本质(有时是明显不可预测的),必然会涉及到许多可能的风险。这些风险可以通过严格的计划和项目管理,以及全面考虑生命周期的各个因素来加以避免。

下面是与软件原型化方法相关的、几个比较共同的风险:

- 不适当、不完全、不充分的分析和设计
- 不现实的实现期望
- 不愿意抛弃原型
- 与用户有关的一些问题

这些问题的解决方法将在下面的几个部分中逐个加以讨论

1.分析和设计的不充分

软件原型化方法的最大陷井是:原型仅是系统分析和设计的一个部分。用户、分析员、以及管理者必须意识到原型仅是 SDLC 方法的一个部分。原型化方法是用来补充生命周期实践和存在的结构技术的,而不是用来代替文档化的系统分析和设计。

即使是对原型来说,仍然需要一些系统设计。分析员必须意识到在开发任何原型之前完成一些基本的设计决策。但在实际开发过程中,许多系统开发人员在没有任何实际设计或说明的情况下就开始了对软件系统的原型化。

尽管使用原型化方法总可以减少形式分析和预先的说明,但不能完全消除它们。如果对商业问题和系统需求没有基本的理解就对系统项目进行原型化,那么可能会在开发的早期就产生不令用户满意的、解决问题方向不正确的系统。

随着较快产生结果的需要,要求设计能灵活地适应不可预见的系统变化。但是如果在进行原型化之前不进行全面的考虑,就可能没有充分理解原来的问题,因此可能忽视其它可以选择的解决办法。由于原型的主要特点是具体而又真实的,因此工作系统原型可能会抑制分析员和用户去认识问题的本质,寻找新的解决办法。由于可能很容易开发出能工作但不完全的设计,可能因此会导致用户对特定的设计或实现作出不成熟的认可。

总之,不完全、不正确、或考虑不成熟的设计减少了对后面原型的怀疑。这样就限制了系统的进一步开发和发展,最终结果是代价很高的修改和维护。最后又需要反复试验来确定问题领域的实质和用户的需求。

为了避免这些风险,可以在编码之前仔细检查系统的逻辑设计。而且严格开发系统的逻辑模型也有助于确保系统实现中的灵活性。另外还必须将物理设计的细节推迟到解决了基本的逻辑设计问题之后进行处理。在设计过程中,还应将一些熟悉的设计工具诸如:数据模型、实体关系图、数据字典、数据流图等和结构化分析和设计技术结合起来使用,并且应当考虑系统与现存系统的体系结构、接口、数据库、功能等有关兼容性、集成性、使用一致性方面的问题。为了便于以后的集成,要仔细考虑、设计与外部系统相关联的接口,并要将它们编制成文档。物理设计的细节也应推迟到这些逻辑设计问题解决之后进行。

最后,文档不存在或不充分就不能认为系统设计是完全的。在用原型化方法来迅速开发系统时,可能没有用户需求或重要的设计决策记录就完成了开发过程。在许多最坏的情况下,代码本身成了系统的唯一文档形式。这些情况是不能允许的。

2.不现实的实现期望

早期的工作模型不可能成比例地扩大成可全面接受的生产性系统,因此工作原型不应该必须成为系统灵活性的证明。原型的实现可能对小型数据库、少量的用户、或小规模的问题是可以接受的,但是当数据库或问题规模扩大到现实或期望的生产水平就可能行不通了。

一个在部分领域或小规模问题上明显成功的原型可能会导致歧途,因为在原型中所使用的方法可能当用在较大的问题上时完全行不通。而对系统需求所作的似乎简单的修改或增加就可能产生难以预料的和严重的副作用,操作参数中的微小改变可能导致设计和实现的重新进行。

另外,用原型化工具或第四代语言开发的原型通常不能象它们的第三代语言对应的部分那样来实现,因而需要传统的程序设计语言来重新实现。但是,用第三代语言(COBOL,PL/1)来重新实现软件系统可能就会引入一些熟悉的问题,即很高的开发和维护代价,缺少灵活性,不便于修改等。由于不能保证用传统语言重新实现就一定能改善系统的执行性能,因此必须慎重权衡重新实现的效益和代价。不仅如此,应用系统的寿命期、经济

效益、实际转换的代价、预计的维修费用等都应作为作重新实现决策时要考虑的重要因素。

为了减少系统执行方面不满足需求的风险,现实的执行需求必须在选择合适的原型化工具的早期确定。一些应用系统有至关重要的执行需求,而另一些则没有。通常有大量操作的应用要求有较高的执行效率;而不经常使用的报表生成器或只有少量操作的应用一般说来不需要有特别高的执行效率。

将适当的原型化工具和技术与任务相匹配是很重要的。一般每个工具只适合特定类型的系统开发,而最终的系统执行可能是千差万别的。例如,一些应用生成器需要用高质量的机器执行来建立高效的应用,这样分析员就能用这些软件包开发出比用 COBOL 开发的更有效的应用;而另一些应用生成器则可以生成 COBOJ 或其它第三代语言代码,在需要的情况下,可将之进行优化以便执行。

3.失去控制的项目

目前考虑到的原型化项目的计划、预算、管理、以及控制等方面的知识还很不够,因而如果不合理地使用适当的管理原则,开发人员就可能失去对系统范围、复杂性等的开发控制。这时,可以使用为传统的 SDLC 定义的特定阶段、里程碑、以及可交付的文档来指导项目管理。但是,在原型化项目系统时,最终的系统需求和物理设计在最初经常是不知道的,原型的数目和复杂性也是不知道的,因此对反复重复的原型化过程和管理也就成了一个较为共同的问题。系统开发人员和用户可能在演示和开发之间反复循环而没有明显的限制,因而何时结束原型化过程的处理的决策通常是个主观的问题,是在过程末尾而不是在开头协商的。

为了限制循环失控的可能性,就需要用户感到到原型已满足他们的期望时去标志原型化过程的结束。用户应该预测要交付的系统将要有的界面和功能,并且必须接受这样的界面和功能,对交付的系统不应有任何诧异的表示。而开发人员则应交付由原型所表示的系统。在用户标志了原型化过程的结束之后,就开始了实际生产系统的开发过程。这时管理者和开发人员应谨慎遵循生命周期的各个阶段以便使所交付的系统是按照所需要的来实际执行的。在实际系统开发完成之后,对需求和说明的改变不仅代价高昂而且非常困难,因而用户必须负担后期改变的影响的责任。

为了避免项目的失控,原型的代价应预先进行估计并且应对处理过程采取原则化的方法。一些合适的技术可用于代价估计及人员和技术需求计划,而且还可以使用项目时间基线控制方法。项目管理对大项目的原型化尤其重要,因而也应建立项目控制和里程碑。此外,管理人员、用户和原型开发人员应该为项目的范围、个人职责、计划、以及预算等准备一个概略协议。

有效的人员管理也将有助于减少不令用户满意的项目的风险。管理者应当意识到要想成功地将项目进行原型化就需要有特殊技能的人员。项目组的成员不仅要有强的分析、设计和沟通能力,而且还要有原型化方法及工具的知识。

4.不愿抛弃原型

在设计和演示循环迭代几次之后,用户可能很容易把原型看作是可以工作的、完成了的系统。但是用户把原型作为生产系统实际上是个风险。尽管原型有高质量的外部接口,给用户一种完成了的假象,但实际上原型可能缺少完成的生产系统的许多重要特征和功能。企业不原意抛弃掉花费了人力和物力的原型是可以理解的,但原型仅是用于演示概念或开发需求的,它们通常不能解决关键的设计问题,如网络和分布式处理、存储限制、安全、错误恢复以及合理的内部控制。

不抛弃不能工作的原型就需要保留它们,或把它们放到生产之中,这就会增加项目失败的可能性。在任何原型化项目开始之前,原型开发人员必须明确原型化过程对用户来说仅是作为他们工作协议的一个部分。原型也可以用来为用户解决一些重要的问题;例如,它们可以用于培训、加快最后的转换、用于有限生产性使用等。

5.用户的一些问题

在原型化项目的过程中,最终系统能否成功的一个很大因素已从职业开发人员转移到了用户。用户必须密切介入系统原型的开发过程,他们通常是信息的主要来源。用户必须严肃地承担起他们的责任,并全面参与开发过程。用户的技能和兴趣都是非常需要的。

用户的需求决策在原型项目中与传统软件开发过程中同样关键。用户一般比系统开发人员更理解他们自己的需求,而且他们通常还能较好地描述和定义他们所需主的东西。但遗憾的是,他们的最大热情通常受到时间

以及其它职责的限制。例如,那些最合格的高级用户也可能就是那些最没有时间来投入到原型化项目过程中的人员。

另外,这些系统的用户可能对他们在系统开发中所扮演的角色没有准备。他们通常缺少业务方面开阔的视野和系统技术方面的知识。尽管他们了解他们的工作,但却可能并不理解系统设计的基本原则。因此,就可能用原型来阐述的仅是最明显和最表面的需求,而当前的原型及其所有不足都可能引入系统之中。开发人员必须认识到这种风险,并且通过让用户参加项目的开发以及取得他们的承诺来消除这种风险。较为理想的情况是:系统设计技能的分析员将他们自己投入到用户环境之中,以便帮助用户确定重要的业务环节和基本问题。

六、小结

原型化方法已经成为补充传统的文档驱动的SDLC模型的一种有力工具。原型化方法将许多系统特征提供给了用户,现在许多企业将原型化方法集成到了自己的开发实践之中。但原型化方法仅是明确用户需求以及解决设计问题的一个局部的生命周期方法。

应当注意的是:成功的软件原型并不是系统开发成功的保证。要使原型化成为一种有效开发方法就需要管理者、开发人员、以及用户去理解这个过程并认识到它的风险。这些风险可以通过遵循适当的开发原则、慎重对待生命周期的各个阶段、以及严格进行项目的管理来减少。

参考文献:

1.J.M.Carey and J.D.Currey,"The Prototyping Conundrum,"Datamation 35(June 1,1989),PP29-23

2.G. Tillman,"Prototyping for the Ringht Results,"Datamation 35(April 1,1989),PP42-45

3.Klimgler,D.E."The Commandments of Prototypimg,"Journal of Information Systems Managements(Summer 1988),PP66-72

4.Randy S. Weiberg."Prototyping and the Systems Development Life Cycle,"Journal of Information Systems Management 2(Spring1991), P47-53

▲