DOI: 10.13656/j.cnki.gxkx.1999.01.011
广西科学 Guangxi Sciences 1999, 6 (1): 37~ 40

DBPM: 基于决定的过程模型*
DBPM: A Decision-based Process Model

唐培和 刘 浩 韦爱忠 刘连芳**
Tang Peihe Liu Hao Wei Aizhong Liu Lianfang

(广西工学院计算中心 柳州市东环路 545005)
(Computer Center, Guangxi Institute of Technology, Donghuanlu, Liuzhou, Guangxi, 545005)

摘要 以决定为基础, 讨论软件开发与维护的过程模型 (DBPM), 并给出了过程模型 DBPM 的规范描述.

关键词 软件开发 软件维护 过程模型
中图法分类号 TP 311.52

Abstract This paper discusses the process model of software, and gives the formal description of DBPM. This model can serve as a mean for the development of the tools and environments to the model.

Key words software development, software maintenance, process model

近年来, 人们普遍认识到, 软件和其他工业产品一样, 其生产率和质量的高低与生产产品的过程有关, 改进软件过程的质量, 可最终获得高质量的软件产品. 因此, 人们又把注意力逐渐转移到开发与维护软件的过程本身—— 软件过程 (Software Process) 上来. 甚至认为 "软件过程也是软件"[1]. 软件过程的研究日益受到重视, 并在过程模型、建模方法和建模语言等方面取得了许多研究成果[4,5].

软件过程涉及到软件开发、维护和演化整个过程的所有活动. 在这些活动中, 人们要不断地做出各种决定 (Decisions), 它们是弄清最终的实现与最初的需求相联系的关键, 因此, 决定在软件过程中是至关重要的. 本文以决定为基础, 讨论软件的过程模型, 以及模型中各要素之间的关系.

1 基于决定的过程模型 DBPM

在软件系统的开发过程中, 一般来讲, 人们无法立刻集中精力于实现的细节上, 即使能够一次编出程序, 一次性的程序设计也有许多弊端. 因此, 人们认识到这些问题后, 提出了逐步求精的思想 (自顶向下的程序设计), 乃至更广泛意义下的 "软件工程". 逐步求精的方法提倡把软件开发的整个过程划分为许多小的步骤, 从模糊的非形式需求描述开始, 到精确定义的形式描述, 再到包含所有实现细节的算法和程序. 在这个逐步精化的过程中, 人们需要不断地做出设计决定 (Design Decisions), 每一次的设计决定使得实现更精确, 更能够有效地执行, 最终得到一个包括所有细节的可有效执行的程序. 因此, 精化过程的中心点, 是如何作出设计决定[2].

另一方面, 软件维护是一件十分困难而又繁重的工作, 费用相当昂贵. 它首先要理解被维护的软件系统, 然后才能根据用户的需求进行改造, 包括增加功能, 提高性能, 以及适应新的软硬件环境等. 因此, 软件维护在某种意义下可看作是软件开发过程的进一步延续.

在软件维护过程中, 不可避免地要做出许多维护决定 (Maintenance Decisions). 每做出一次维护决定, 被维护的系统向维护目的更进一步. 维护过程中的决定对于软件系统的再维护是十分重要的, 它们与软件开发过程中的设计决定具有同等的地位和意义. 因此, 基于决定的过程模型既要包含开发过程中的设计决定, 又要包含维护决定, 使软件的开发与维护过程在过程模型中有机地结合起来.

软件开发与维护过程实际上就是不断地认识问题和解决问题的过程, 针对不同的问题, 需要做出不同的决定, 以便确定合理的解决问题的方案, 并由此获得解决每一个问题的代码. 根据这样的思路, 我们可以得到一个初步的基于决定的软件过程模型

1998-01-16收稿.
* 广西区科委青年科学基金项目资助 (桂科青 9452008).
** 广西计算中心, 南宁市星湖路, 530022 (Computer Center of Guangxi, Xinghulu, Nanning, Guangxi, 530022).

DBPM 如图 1所示。

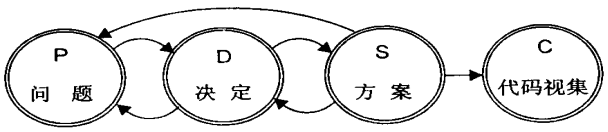

图 1 过程模型 DBPM

Fig. 1 A decision-based process model

在这里, P为待解决的问题 (Problem); D为做出的决定 (Decision), 既可以是设计决定, 也可以是维护决定; S为解决问题的方案 (Solution); C为对应的代码视集 (Code View)。带箭头的连线为因果关系。

根据软件生命周期的特点,我们通常把决定分为设计决定与维护决定两大类。本文按照决定的不同性质, 将其划分为三种不同类型的决定: 一种是分解决定 (Decomposition Decisions, 简称 DDs); 另一种是精化决定 (Refinement Decisions, 简称 RDs); 还有一种为选择决定 (Selection Decisions, 简称 SDs)。

分解决定 (DDs) 系统工程学告诉我们: 求解一个复杂的大问题, 可按子系统分解原理, 逐级分解成多个具有相对独立性的子问题,直到最终的子问题可以分别求解为止。不过问题如何分解, 按什么原则分解, 就需要针对具体问题, 在设计与维护过程中做出合理的决定。这种分解问题的决定, 我们称之为分解决定 (Decomposition Decisions)。例如, 设待解的问题为 P, P的求解可分解为若干个子问题的求解, 即:

$$P = D(p_1, p_2, \cdots, p_n).$$

在这里, $p_1, p_2, \cdots, p_n$是由 P分解派生出来的,比 P的规模更小或更容易求解的子问题。D描述了 P与 $p_1, p_2, \cdots, p_n$之间的关系,或者说描述了 P的分解策略。这种分解策略,就是设计与维护过程中做出的分解决定。问题分解必须合理,且层次分明; 此外, 还须注意,所有 $p_i (i = 1, 2, \cdots, n)$的解的综合恰好能构成 P的解。

精化决定 (RDs) 对于那些不需要进一步分解的问题或者某个问题的求解方案,有时需要进一步求精 (细化), 直到它们变成一组已被认识或管理上可控的变量集为止,用它们来构成该求解问题的变量体系。因此, 如何对问题或方案求精, 精化到什么程度, 需要我们在设计与维护过程中做出精化决定 (Refinement Decisions)。即:

$$p \xrightarrow{RD} p' \text{ 或者 } s \xrightarrow{RD} s'.$$

在这里, p'、s'分别为 p和 s的精化结果。问题求精的关键, 是要保证求精前后问题的性质保持不变。同样道理,方案的精化必需保证精化前后方案所产生的结果不变。

选择决定 (SDs) 对于某一个具体的问题, 可能有多种分解办法,不同的分解决定产生不同的分解结果; 另外, 对于同一个问题, 也可能存在多种求解方案, 不同求解方案的前提与效果不同。因此, 针对某一个问题, 选择哪一种决定或方案更合理呢? 这就需要我们实事求是地做出选择决定 (Selection Decisions)。典型地:

$$S \xrightarrow{RD} s',$$

其中, $S = \{s_i \mid i = 1, 2, \cdots, n\}$, $s' \in S$.

选择决定的前提是必须有多个可供选择的方案, 且每个方案都能达到预期的目的。

根据以上分析,可进一步得到更详细的基于决定的软件过程模型 如图 2所示。在此模型中, 从问题到决定, 再到相应的代码视集, 形成了一条链, 称之为变化链。通过变化链记录了软件开发与维护活动的实际进程。变化链不仅总能反映当前状态, 还能反映历史; 此外, 当系统发生变化时, 通过修改状态属性, 非决定性方案可以容易地转变为决定性方案。此外, 该模型对问题、决定和方案的描述, 可产生软件开发与维护过程中的各种文档。

可见, 在软件开发与维护过程中, 人们要做出许多决定。这些决定是开发与维护过程中的重要信息, 它们是弄清最终的实现如何与最初的需求相联系的关键。因此, 决定是开发与维护过程中的核心问题。

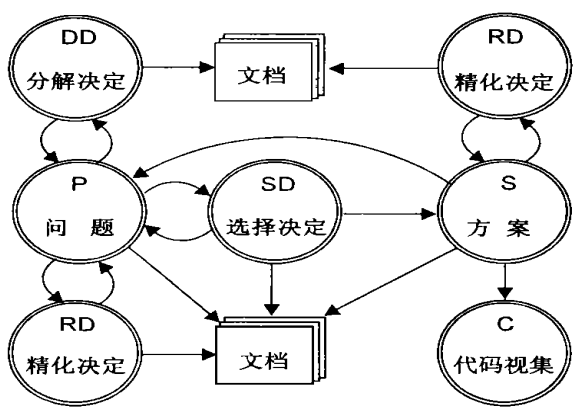

图 2 改良的过程模型 DBPM

Fig. 2 An improved decision-based process model

2 模型的规范描述

近年来,软件工程界的专家学者一直提倡对软件过程要做明确、规范的记录。Peter Freeman阐明了对设计过程做合理的记录所具有的意义, 并指出, 如果拥有这样的环境, 它能记录、跟踪、组织并利用软件过程信息, 那么软件过程本身也将变成有用的财

富[3]。

基于决定的软件过程模型包括问题、决定、方案和代码视集四种基本对象,模型的规范描述是设计支持该模型的环境与工具的基础。软件过程涉及的范围较宽,它不仅和软件的开发与维护的环境有关,还跟设计者的经验、所使用的开发方法及其表示、问题域以及软件规模有很大关系。因此,模型的规范描述也只能涉及一般性的问题。

2.1 问题 (Problems)

过程模型中对问题的描述涉及待解决的问题需要我们"做什么",并不要求我们回答"怎么做"。这种描述可以是形式化、半形式化(如结构化或图形)或者非形式化的。能否做形式化或半形式化的记录与描述,和软件过程的环境与工具密切相关。

每个问题具有唯一的标识。问题的描述可用5种不同的属性来表示,其中属性 Description确定问题的语义;属性 Status指明问题是否已被处理,或者将被放弃; Result-from和 Subject-of为关系属性,描述问题与决定之间的关系;通过属性 Reference,可描述更多的相关信息,如参考文献、会议备忘录等。

Problems= Pid→ Problem
Problem:: Description× Reference× Result-from × Subject-of× Status
Description:: STRING
Reference:: STRING
Status= PROCESSED| IGNORED| PENDING
Result-from= DDid| SDid| RDid| NULL
Subject-of= DDid| SDid| RDid
Pid, DDid, SDid, RDid:: TOKEN

在这里,须指出的是仅对原始问题,属性 Result-from的值为空(NULL)。

2.2 决定 (Decisions)

程序设计与维护过程是做出合理的决定的过程。基于决定的过程模型把决定划分为分解决定(DDs),选择决定(SDs)和精化决定(RDs)。因此,决定的描述也应分别讨论。

分解决定(DDs) 分解决定作用于特定的问题,分解的结果是得到一系列的子问题。对分解决定的记录与描述,应侧重于分解的方法、原因与理由。

每个分解决定具有唯一的标识。分解决定具有5种不同的属性,属性 Subject为分解决定的作用对象;属性 Description描述分解方法;属性 Justification描述这种分解的原因与理由; Result记录分解决定的结果;以及属性 Status记录该分解决定的状态。

DDs= DDid→ DD
DD:: Subject× Description× Justification× Result× Status
Subject= Pid
Description:: STRING
Justification:: STRING
Result= Pid
Status= ACTIVE| SLEEPING
DDid:: TOKEN

选择决定(SDs) 根据明确的约束关系,选择权或者某种理由,软件设计与维护是在可供选择的方案中有选择地做出决定。选择哪一种方案以及为什么做出这种选择是记录与描述选择决定的关键。

在这里,属性 Alternatives记录可供选择的方案; Select描述最后的选择结果。其它属性与分解决定基本相同。

SDs= SDid→ SD
SD:: Subject× Description× Justification× Alternatives× Select× Status
Subject= Pid
Description:: STRING
Justification:: STRING
Alternatives= Sid
Select= Sid
Status= ACTIVE| SLEEPING
SDid, Sid:: TOKEN

精化决定(RDs) 精化决定与问题或方案的进一步求精有关。做出这种决定的理由以及如何精化对于精化决定的记录与描述是不可缺少的。

RDs= RDid→ RD
RD:: Subject× Description× Justification× Result × Status
Subject= Pid| Sid
Description:: STRING
Justification:: STRING
Result= Pid| Sid
Status= ACTIVE| SLEEPING
RDid, Pid, Sid:: TOKEN

2.3 方案 (Solutions)

方案是解决问题的思路和方法。方案与其对应的代码相关联,根据特定的方案可求出指定问题的解。

每个方案具有唯一的标识。属性 Subject指明该方案针对哪一个问题而言;属性 Decision表示该方案是由哪一个决定所产生的;方案的详细描述及为什么采用这种方案可通过属性 Description与 Justification来描述。属性 CodeView记录与本方案对应的代码视

集，Status为状态属性。

Solutions= Sid→ Solution

Solution :: Subject× Decision× Description× Justification× CodeView× Status

Subject= Pid

Decision= SDid| RDid

Description :: STRING

Justification :: STRING

CodeView= Cid

Status= ACTIVE| SLEEPING

Sid, Cid :: TOKEN

2.4 代码视集 (**Code Views**)

代码视集就是基于特定软硬件环境、解决指定问题的源程序代码。和传统意义下的代码片段 (程度段) 不同，它是一种代码片集。

代码视集是根据问题来划分的，既与功能性需求有关，也与非功能性需求有关；而传统的程序段 (程序块) 是根据功能来划分的。因此前者的代码行可以是连续的，也可能是离散的；而后者的代码行通常是连续的。并且，两个不同的代码视集的交集可能是非空的。

CodeViews= Cid→ CodeView

CodeView :: Subject× Codes× Status

Subject= Sid

Codes= Lid→ Line

Line :: STRING

Status= ACTIVE| SLEEPING

Cid, Sid, Lid :: TOKEN

3 结语

传统的软件开发技术不太注重对于设计过程中所作出的决定这个关键问题的描述和记录。这些设计与维护信息的丢失导致了许多问题，如因此所带来的难以理解程序是如何实现了所需的功能及维护和证明中的重重困难。本文不仅模型化软件过程，且对过程信息的记录与描述给出了规范的描述。

软件过程模型是过程的抽象描述，对过程的描述可以是形式的、半形式化的或非形式的。一个过程模型通常表达了一定的抽象层次和看待过程的一种特定观点。基于该模型的软件工具 (SDMA) 已基本完成，基于决定的思想在 SDMA中得到了较好的体现。

建立过程模型的首要目的是为了更好地理解过程，精确地表示过程，从而使过程的参与者可以有效地交流、合作；其次则是试图自动地控制、指导和帮助参与过程中的人和工具，并可使一些开发活动可以自动地重复进行；最后，过程模型为过程的修改和评估提供了良好的基础。

参考文献

1 Osterweiol L. Software processes are softwares too, In: Proceedings of the 9th International Conference on Software Engineering, Los Alamitos, CA: IEEE Computer Soc Press, 1987, 2~ 13.

2 缪 旭，唐稚松. 基于设计决定的逐步求精方法及环境. 软件学报，1990, 1 (3): 15~ 27.

3 Guillermo Arango. A tool shell for tracking design decision. IEEE Software, March 1991, 75~ 82.

4 李 健，邵维忠，杨芙清. 软件过程建模方法分类概述. 计算机应用与软件，1996, 13 (2): 1~ 8.

5 柳军飞，唐稚松. 软件过程建模语言研究. 软件学报，1996, 7 (8): 449~ 457.

(责任编辑：黎贞崇)

生物化学中最后一个巨大前沿—— 糖生物学

1988年牛津大学 R. A. Pwek教授首先提出“glycobiology”概念，糖生物学以生物大分子组成的部分糖链为对象，研究其作为“信息分子”在多细胞生物的高层次生命中的特殊功能，被称为生物化学中最后一个巨大前沿。现已确认糖链参与生物体受精、发育、分化、免疫、神经系统信号的识别与调控；在微生物与动物、微生物与植物的相互作用中担任重要角色：在生物老化、癌症过程中也涉及糖链的参与。糖生物学与糖工程研究在迅速兴起，下面是国内外科学家十分关注的前沿课题：(1) 神经糖生物学：糖链结构与神经细胞信号识别与通讯；(2) 糖病理学：糖脂和糖蛋白在合成、分解过程中各种酶的先天缺陷病理情况下，糖链的变化，以及糖糖合成调控及其功能等；(3) 糖免疫学：寡糖在免疫机制中的作用，糖基化改变与疾病关系，新型糖治疗方法和策略；(4) 糖的结构生物学：糖链结构、糖链合成调控及其功能；(5) 糖工程及药物开发；(6) 糖类空间结构的计算机模拟。

(摘自 中国科学院 1999年《科学发展报告》，P252，题目为本刊按)