基于EPMM的软件过程结构合理性研究

代 飞[1,2] 李 彤[1,2] 谢仲文[1,2] 莫 启[1] 金运志[1]
(云南大学软件学院 昆明 650091)[1] (云南大学云南省软件工程重点实验室 昆明 650091)[2]

摘 要 为了确保软件演化过程的正确性,有必要对软件演化所涉及的软件过程进行结构合理性研究,以提高软件演化的质量和效率、缩短软件演化的周期。针对EPMM建模产生的软件演化过程模型,从过程层的角度,提出结构合理性定义,并设计相应的判断算法。结果表明,结构合理性的检验有助于提高软件演化过程的质量。

关键词 软件演化过程,软件演化,软件过程,性质合理性,Petri网,元模型
中图法分类号 TP311 **文献标识码** A

Research on Structure Soundness of Software Processes Based on EPMM

DAI Fei[1,2] LI Tong[1,2] XIE Zhong-wen[1,2] MO Qi[1] JIN Yun-zhi[1]
(School of Software, Yunnan University, Kunming 650091, China)[1]
(Key Laboratory for Software Engineering of Yunnan Province, Yunnan University, Kunming 650091, China)[2]

Abstract In order to ensure the correctness of software evolution processes, to check the structure soundness of the software processes under which the corresponding software is evolving is necessary. It is used to improve the quality and efficiency of software evolution and shorten the time of software evolution. According to the software evolution process models modeled by software evolution process meta-model(EPMM), the structure soundness was defined from the point of view of software processes. Moreover, the corresponding algorithms were designed. The result shows that to check the structure soundness helps to improve the quality of software evolution processes.

Keywords Software evolution processes, Software evolution, Software processes, Structure soundness, Petri nets, Meta model

1 引言

随着计算环境从以CPU为中心的单机环境向以互联网为核心的网络环境发展,软件演化的频率和复杂性越来越高。其核心问题是软件如何随着用户需求和外部环境的不断变化而进行灵活的改变。在软件演化领域,雷曼等在一系列研究工作的基础上,总结提出了8条著名的软件演化定律,称之为雷曼软件演化定律[1]。

通过软件开发实践,人们逐步认识到软件过程可以规范软件开发行为,进而提高软件产品的质量。ISO/IEC 12207将软件过程定义为活动的集合,活动视作任务的集合,任务则把输入变为输出[2]。在软件过程领域,Osterweil提出了一个被广泛接受的观点"软件过程也是软件",认为应像编程软件一样来建模软件过程[3]。

软件演化过程是软件演化和软件过程的交叉学科,指软件演化中涉及的一系列过程。它为软件演化构造了一个管理框架,是实施软件演化的工作流程。文献[4]认为:软件演化过程可以有效管理软件演化,提高软件演化的效率。在此基础上,通过实践逐步发现:一个结构合理的软件演化过程可以确保过程结构不发生异常,是满足过程定义正确性的最低要求,能有效促进软件演化的成功实施;反之将导致软件演化的失败。因此,对软件演化过程的结构合理性进行研究,有助于提高软件演化过程的质量,进而提高软件演化的质量。

2 软件演化过程元模型

软件过程建模是通过特定的方法对软件过程进行抽象、表示和分析,以增加对软件过程理解的活动[5]。建模产生的软件过程模型是软件过程的抽象描述,可以规范软件开发和维护的行为,是实施软件过程的依据;软件过程是软件过程模型的动态执行。

为了形式建模软件演化过程,本文作者之一基于基本Petri网、扩展面向对象技术和Hoare逻辑,提出了软件演化过程元模型EPMM[6]。自上向下,EPMM分为4层:全局层、过程层、活动层和任务层。各层的形式定义如下:

到稿日期:2012-10-19 返修日期:2012-12-21 本文受国家自然科学基金(60963007,61262024,61262025),云南省自然科学基金(2012FD005,2012FB118,2012FB119),云南省软件工程重点实验室开放基金(2010KS01,2011SE04,2012SE307,2012SE101)资助。

代 飞(1982-),男,博士,讲师,CCF会员,主要研究方向为软件过程、软件工程,E-mail:flydai.cn@gmail.com;李 彤(1963-),男,博士,博士生导师,主要研究方向为软件过程、软件工程;谢仲文(1982-),男,博士,讲师,主要研究方向为软件工程;莫 启(1986-),男,博士生,主要研究方向为软件工程;金运志(1989-),男,硕士生,主要研究方向为软件工程。

定义 1[6] 任务是一个四元组 $t=(\{Q_1\},\{Q_2\},M_t,M_0)$，其中，

(1)Q_1 和 Q_2 是一阶谓词公式：$\{Q_1\}$称为前断言，用于定义任务 t 执行前的状态；$\{Q_2\}$称为后断言，用于定义任务 t 执行后的状态；

(2)$A(F)=(\{Q_1\},\{Q_2\})$是一个 2-断言，用于定义任务 t 的功能；

(3)M_t 是接收消息的集合，当任务 t 获得 M_i 中的一个或多个消息时，任务 t 被执行；

(4)M_0 是发送消息的集合，$\forall m\in M_0, m=(r,b)$，表示当任务 t 被执行时，t 发送一个消息 m 到 r，b 称为消息体，包含许多参数。

定义 2[6] 活动是一个四元组 $a=(I,O,L,B)$，其中，

(1)I,O 和 L 分别称为活动的输入数据结构、输出数据结构和局部数据结构。

(2)B 称为活动体，既可以是一个软件过程 p，也可以是任务 $t_{Main},t_1,t_2,\cdots,t_n$ 的集合，这些任务或软件过程在数据结构 I,O 和 L 上执行。任务 t_{Main} 是活动中首先获得消息 execution 并且执行的特殊任务。

(3)活动的定义是一个类的定义，称为活动类。当活动被执行时，活动对象就被创建。

定义 3[6] 软件过程系统是一个四元组 $\Omega=(C,A;F,M)$，其中，

(1)(C,A,F)是一个没有孤立元素的网，即 $A\cup C\neq\emptyset$；

(2)C 是一个条件的有限集，$\forall c\in C$ 称为一个条件；

(3)A 是一个活动的有限集，$\forall a\in A$ 称为一个活动；元素 a 的发生称为 a 被执行或者 a 点火；

(4)$M\subseteq 2^C$ 称为 Ω 的实例类，其中 2^C 表示 C 的幂集；

(5)$\forall a\in A,\exists m\in M$，以致 a 在 m 可以发生。

定义 4[6] 令 $\Omega=(C,A;F,M)$是一个软件过程系统，设 $M_0\in M(M_0\subseteq C)$是 Ω 的一个实例，$\Sigma=(C,A;F,M_0)$。M_0 称为 Σ 的初始情态，元素 $d\in M_0$ 称为一个托肯，Σ 称为一个软件过程。

定义 5[6] 全局模型是一个二元组 $g=(P,E)$，其中，

(1)P 是一个软件过程的集合；

(2)$E\subseteq P\times P$ 是一个偏序的二元关系，称为 P 的嵌入关系；$E=\{(p,p')\mid p,p'\in P\wedge p'$嵌入到 p 中$\}$，p'称为 p 的子过程。

由上述定义可知，由 EPMM 建模产生的软件演化过程模型也应包含 4 层，自上而下分别是：全局层、过程层、活动层和任务层。其中，过程层建模产生的软件过程本质上是一个基本 Petri 网，条件集 C 中的元素只有“有托肯”和“无托肯”两种状态。根据 Petri 网的点火规则，建模产生的软件过程可以仿真执行。

为了直观表示，通常使用圆圈表示软件过程中的条件，使用方框表示软件过程中的活动。

例如，过程工程师使用 EPMM 建模产生的软件过程 p_1，如图 1 所示。

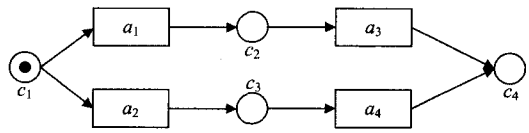

图 1 一个软件过程

p_1 的形式定义：

$C=\{c_1,c_2,c_3,c_4\}$

$A=\{a_1,a_2,a_3,a_4\}$

$F=\{(c_1,a_1),(c_1,a_2),(a_1,c_2),(a_2,c_3),(c_2,a_3),(c_3,a_4),(a_3,c_4),(a_4,c_4)\}$

$M_0=\{c_1\}$

其中 $N=\{C,A;F\}$，称为 p_1 的结构。

但是，由 EPMM 建模产生的软件演化过程模型是否正确呢？这是摆在建模者面前必须回答的一个问题。因为错误的软件演化过程模型将导致软件演化的失败。我们认为：合理的软件演化过程模型是满足过程定义正确性的最低要求，对于提高软件演化的质量和效率、缩短软件演化的周期，具有重要意义。

针对上述问题，本文作者之一在文献[7]中依据 EPMM 对任务和活动的形式化定义，对任务和活动的合理性进行了研究。本文将依据 EPMM 对软件过程的形式化定义，从结构方面着手，对软件过程(相对于活动和任务)的合理性进行研究。

3 结构合理性的定义

结构合理性用于检验软件过程在结构和语法方面是否满足某些限制条件，与程序设计语言的语法检验类似，可以确保过程结构不发生异常。从建模者和使用者的角度看，一个结构合理(参见定义 6)的软件过程，既要遵循基本 Petri 网的定义，又要满足 EPMM 的语法要求，还要避免结构缺陷所带来的死锁。

具体而言，从基本 Petri 网的角度，结构合理性定义应只考虑网的结构，不涉及任何情态；从语法的角度，结构合理性定义应与 EPMM 的建模方法、限制条件和语法保持一致；从仿真执行的角度，结构合理性定义应确保建模产生的软件过程无结构死锁。

定义 6 设 $\Sigma=(C,A;F,M_0)$是一个由 EPMM 建模产生的软件过程，其中 M_0 是初始情态。软件过程的结构合理性 S 是指当且仅当满足以下 4 个条件：

(1)$\forall c\in C,{}^\cdot c\cap c^\cdot=\emptyset$；

(2)有且只有 $\exists c_1,c_2\in C$，且 $in(c_1)=0$ 和 $out(c_2)=0$，则称 c_1 为开始条件，c_2 为结束条件；

(3)$\forall x\in C\cup A$，不存在 $in(x)=0$ 和 $out(x)=0$；

(4)$\forall C_1\subseteq C$，不存在 ${}^\cdot C_1\subseteq C_1{}^\cdot$[8]。

第一个条件是从基本 Petri 网的角度，定义软件过程需要满足的限制条件：没有伴随条件。其中，伴随条件参见定义 7，${}^\cdot c$ 表示条件 c 的前集，参见定义 8；$c^\cdot$ 表示条件 c 的后集，参见定义 9。

对于其它 Petri 网系统，如库所变迁网和有色 Petri 网系

统，伴随库所的出现可以理解为化学反应中的催化剂。但是，对基本 Petri 网而言，伴随条件的出现会使相关的活动不能点火，由此导致死锁。例如，图 2 所示的软件过程中，条件 c_2 是伴随条件，它的出现使得活动 a_1 无法发生点火。

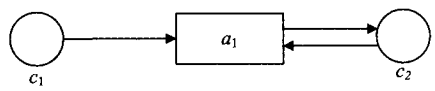

图 2　伴随条件

第二个条件是从 EPMM 的语法角度，定义软件过程需要满足的限制条件：有且只有一个开始条件和一个结束条件。其中，$in(c_1)$和 $out(c_2)$参见定义 10。

此限制条件与 EPMM 要求建模产生的软件演化过程模型具有单入口性和单出口性一致[6]。例如，图 3 所示的软件过程中，c_1 和 c_2 分别是开始条件和结束条件。

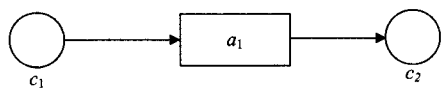

图 3　初始块

第三个条件是从建模角度，定义软件过程需要满足的限制条件：不存在孤立点。

例如，图 4 所示的软件过程中，存在与其他节点毫无联系的孤立点：c_7 和 a_8，这些节点对软件开发和维护毫无指导意义。因而，在软件过程建模中应避免出现。

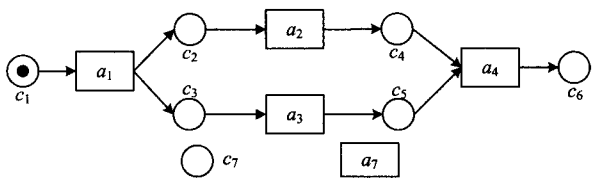

图 4　孤立活动和条件

第四个条件是从过程实施的角度，定义软件过程需满足的限制条件：不存在结构死锁。其中，$^{\cdot}C_1$ 和 $C_1{}^{\cdot}$ 表示条件集的前集和后集，分别参见定义 11 和定义 12。

在并发系统的运行过程中，为保证系统的运行安全可靠，必须有效解决死锁问题。而解决死锁的首要任务是解决与初始标识无关的 Petri 网死锁问题，即结构死锁。因此，软件过程建模应避免出现结构死锁。

例如，图 5 所示的软件过程，无论初始情态怎么指定，执行过程都将发生死锁。因为，令 $C_1=\{c_1,c_2\}\subseteq C$，由于 $^{\cdot}\{c_1,c_2\}=^{\cdot}c_1\cup{}^{\cdot}c_2=\{a_1\}\cup\{a_2\}=\{a_1,a_2\}$，$\{c_1,c_2\}^{\cdot}=c_1^{\cdot}\cup c_2^{\cdot}=\{a_2,a_3\}\cup\{a_1\}=\{a_1,a_2,a_3\}$，满足 $^{\cdot}\{c_1,c_2\}\subseteq\{c_1,c_2\}^{\cdot}$，所以$\{c_1,c_2\}$是软件过程的结构死锁。

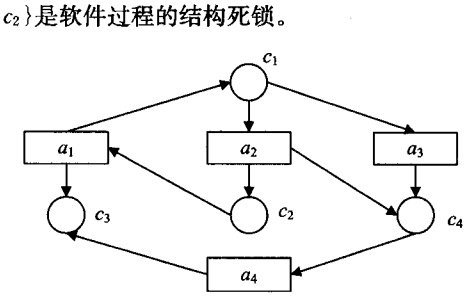

图 5　结构死锁

定义 7　设 $\Sigma=(C,A;F,M_0)$是一个由 EPMM 建模产生的软件过程，其中 M_0 是初始情态，若 $\exists c\in C,{}^{\cdot}c\cap c^{\cdot}\neq\varnothing$，则称 c 是伴随条件。

定义 8　设 $\Sigma=(C,A;F,M_0)$是一个由 EPMM 建模产生的软件过程，其中 M_0 是初始情态，若 $c\in C,{}^{\cdot}c=\{a\mid(a,c)\in F\}$，则称 $^{\cdot}c$ 为条件 c 的前集。

定义 9　设 $\Sigma=(C,A;F,M_0)$是一个由 EPMM 建模产生的软件过程，其中 M_0 是初始情态，若 $c\in C,c^{\cdot}=\{a\mid(c,a)\in F\}$，则称 $c^{\cdot}$ 为条件 c 的后集。

定义 10　设 $\Sigma=(C,A;F,M_0)$是一个由 EPMM 建模产生的软件过程，其中 M_0 是初始情态，$\forall c\in C$，结点 c 的入度 $in(c)=|\{(x,c)\mid(x,c)\in F,x\in A\}|$，结点 c 的出度 $out(c)=|\{(c,x)\mid(c,x)\in F,x\in A\}|$。

定义 11　设 $\Sigma=(C,A;F,M_0)$是一个由 EPMM 建模产生的软件过程，其中 M_0 是初始情态，若 $C_1\subseteq C$ 则 $^{\cdot}C_1=\{\cup{}^{\cdot}c\mid c\in C\}$，则称 $^{\cdot}C_1$ 为条件集的前集。

定义 12　设 $\Sigma=(C,A;F,M_0)$是一个由 EPMM 建模产生的软件过程，其中 M_0 是初始情态，若 $C_1\subseteq C$ 则 $C_1{}^{\cdot}=\{\cup c^{\cdot}\mid c\in C\}$，则称 $C_1{}^{\cdot}$ 为条件集的后集。

4　检验算法

任给一个由 EPMM 建模产生的软件过程 $\Sigma=(C,A;F,M_0)$，由定义 6 可知，判定 Σ 是否满足结构合理性，需要检验 4 个条件：(1)无伴随条件；(2)有且只有一个输入条件和输出条件；(3)无孤立节点；(4)无结构死锁。检验算法见算法 1。

算法 1(结构合理的检验算法)　判定一个 $\Sigma=(C,A;F,M_0)$是否满足定义 6，如果满足，则输出 Σ 是结构合理的。

输入：$\Sigma=(C,A;F,M_0)$；

输出：如果 Σ 满足定义 6，则输出“Σ 是结构合理的”；如果 Σ 不满足，则输出“Σ 结构不合理”。

```
BEGIN
  1. Call 算法 2 判断 Σ 是否满足无伴随条件；
  2. Call 算法 4 判断 Σ 是否满足只有一个开始条件和结束条件；
  3. Call 算法 5 判断 Σ 是否满足无孤立节点；
  4. Call 算法 6 判断 Σ 是否满足无结构死锁；
  5. IF 上述 4 个条件都满足 THEN
     5.1 输出“Σ 是结构合理的”
END ELSE
     5.2 输出“Σ 结构不合理”
  END
END
```

算法 2(伴随条件的检验算法)　判定一个软件过程 $\Sigma=(C,A;F,M_0)$是否具有伴随条件。

输入：$\Sigma=(C,A;F,M_0)$；

输出：如果 Σ 满足定义 6 中的条件 1，则输出“Σ 满足条件 1”；如果 Σ 不满足，则输出“Σ 不满足条件 1”。

```
BEGIN
  1. WHILE 在集合 C 中还存在未被标识的节点
  BEGIN
    1.1 从集合 C 中任选一个未被访问的节点 c 并标识它；
```

1.2 Call 算法 3 计算节点 c 的前集和后集；

1.3 IF $\cdot c \cap c^{\cdot} \neq \phi$ THEN

1.3.1 输出“Σ 不满足条件 1”

END

2. 输出“Σ 满足条件 1”

END

算法 3(条件的前集和后集的计算算法)　计算条件节点 c 的前集和后集。

输入：流关系集 F 和条件 c；

输出：条件节点 c 的前集和后集。

BEGIN

1. $\cdot c = \emptyset$；

2. $c^{\cdot} = \emptyset$；

3. WHILE 在集合 F 中还存在未被标识的节点；

BEGIN

3.1 从集合 F 中任选一个未被标识的节点 f=(x,y)并标识它；

3.2 IF f 的第一个元素 x∈A AND f 的第二个元素 y==c THEN

3.2.1 $\cdot c = \cdot c \cup x$

END

3.3 IF f 的第一个元素 x==c AND f 的第二个元素 y∈A THEN

3.3.1 $c^{\cdot} = c^{\cdot} \cup y$；

END

END

4. 输出“$\cdot c$ 和 $c^{\cdot}$”

END

算法 4(开始条件和结束条件的检验算法)　判定一个软件过程 $\Sigma=(C,A;F,M_0)$ 是否具有一个开始条件和结束条件。

输入：Σ=(C,A;F,M_0)；

输出：如果 Σ 满足定义 6 中的条件 2，则输出“Σ 满足条件 2”；如果 Σ 不满足，则输出“Σ 不满足条件 2”。

BEGIN

1. $number_1 = number_2 = 0$；

2. $flag_1 = flag_2 =$ FALSE；

3. WHILE 在集合 C 中还存在未被标识的节点

BEGIN

3.1 从集合 C 中任选一个未被标识的节点 c 并标识它；

3.2 IF in(c)==0 AND out(c)≠0 THEN

3.2.1 $flag_1 =$ TRUE；

3.2.2 $number_1 = number_1 ++$；

END

3.3 IF in(c)≠0 AND out(c)==0 THEN

3.3.1 $flag_2 =$ TRUE；

3.3.2 $number_2 = number_2 ++$；

END

3.4 IF $flag_1$ AND $flag_2$ AND $number_1$==1 AND $number_2$==1 THEN

3.4.1 输出“Σ 满足条件 2”

END

END / * WHILE 循环结束 * /

4. 输出“Σ 不满足条件 2”

END

算法 5(孤立节点的检验算法)　判定一个软件过程 $\Sigma=(C,A;F,M_0)$ 是否具有孤立节点。

输入：Σ=(C,A;F,M_0)；

输出：如果 Σ 满足定义 6 中的条件 3，则输出“Σ 满足条件 3”；如果 Σ 不满足，则输出“Σ 不满足条件 3”。

BEGIN

1. WHILE 在集合 C 中还存在未被标识的节点

BEGIN

1.1 从集合 C 中任选一个未被标识的节点 c 并标识它；

1.2 IF in(c)==0 AND out(c)==0 THEN

算法终止，输出“Σ 不满足条件 3”

END

END / * WHILE 循环结束 * /

2. WHILE 在集合 A 中还存在未被标识的节点

BEGIN

2.1 从集合 A 中任选一个未被标识的节点 a 并标识它；

2.2 IF in(a)==0 AND out(a)==0 THEN

2.2.1 算法终止，输出“Σ 不满足条件 3”

END

END / * WHILE 循环结束 * /

3. 输出“Σ 满足条件 3”

END

算法 6(结构死锁的检验算法)　判定一个软件过程 $\Sigma=(C,A;F,M_0)$ 是否具有结构死锁。

输入：Σ=(C,A;F,M_0)；

输出：如果 Σ 满足定义 6 中的条件 4，则输出“Σ 满足条件 4”；如果 Σ 不满足，则输出“Σ 不满足条件 4”。

BEGIN

1. 计算条件集 C 的幂集 2^C；

2. WHILE 在幂集 2^C 中还存在未被标识的节点

BEGIN

2.1 从幂集 2^C 中任选一个未被标识的节点 C 并标识它；

2.2 Call 算法 7 计算集合 C 的前集和后集；

2.3 IF $\cdot C \subseteq C^{\cdot}$ THEN

2.3.1 输出“Σ 不满足条件 4”

END

END

3. 输出“Σ 满足条件 4”

END

算法 7(条件集的前集和后集计算算法)　计算条件集 C_1 的前集和后集。

输入：流关系集 F 和条件集 C_1；

输出：C_1 的前集和后集。

BEGIN

1. $\cdot C_1 = \emptyset$；

2. $C_1^{\cdot} = \emptyset$；

3. WHILE 在条件集 C_1 中还存在未被访问的节点

BEGIN

3.1 从集合 C_1 中任选一个未被访问的节点 c 并标识它；

3.2 Call 算法 3 计算节点 c 的前集和后集；

3.3 $\cdot C_1 = \cdot C_1 \cup \cdot c$；

3.4 $C_1^{\cdot} = C_1^{\cdot} \cup c^{\cdot}$

END

4. 输出“$\cdot C_1$ 和 $C_1^{\cdot}$”

END

5 应用

使用 EPMM 对软件演化涉及的一个设计演化过程进行建模，得到的软件过程如图 6 所示。

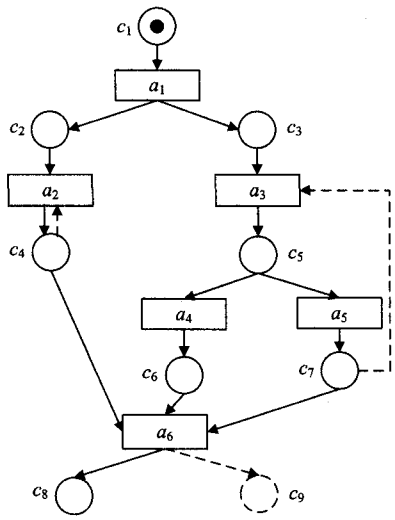

图 6 一个设计演化过程

其中，条件和活动的含义如下：

c_1：设计完成；a_1：设计变化请求；

c_2：请求评审 1；c_3：请求评审 2；

a_2：风险分析；a_3：设计划分；

c_4：风险评审；c_5：设计划分评审；

a_4：设计 1；a_5：设计 2；

c_6：设计 1 评审；c_7：设计 2 评审；

a_6：设计确认；c_8：反馈 1；

c_9：反馈 2。

使用算法 1 对设计演化过程进行检验，不难发现存在以下问题：

(1)结束条件不唯一。图 6 所示设计演化过程中，c_1 满足 $in(c_1)=0$，为开始条件；但 $out(c_8)=0$ 和 $out(c_9)=0$，故而结束条件不唯一。

(2)存在伴随条件。图 6 所示设计演化过程中，c_4 满足 $\cdot c_4 \cap c_4^{\cdot} = a_2$，为伴随条件。由于 EPMM 基于基本 Petri 网定义，故而伴随条件的出现，将使得 a_2 无法点火。

(3)存在结构死锁。图 6 所示设计演化过程中，$C_1=\{c_5, c_7\} \subseteq C$，由于 $\cdot\{c_5,c_7\} = \cdot c_5 \cup \cdot c_7 = \{a_3\} \cup \{a_5\} = \{a_3, a_5\}$，$\{c_5,c_7\}^{\cdot} = c_5^{\cdot} \cup c_7^{\cdot} = \{a_4, a_5\} \cup \{a_3, a_6\} = \{a_4, a_5, a_3, a_6\}$，满足 $\cdot\{c_5,c_7\} \subseteq \{c_5,c_7\}^{\cdot}$，故而存在结构死锁。

针对问题(1)，建模者需删除 c_9 及其对应的流关系即可，如图 6 中虚线部分所示；针对问题(2)，建模者只需删除 c_4 和 a_2 间的一条流关系即可，如图 6 中虚线部分所示；针对问题(3)，建模者只需删除 c_7 和 a_3 间的一条流关系即可，如图 6 中虚线部分所示。

结束语 在工作流和业务过程管理领域，工作流网的合理性从 3 个方面定义：弱终止性(无死锁和活锁)、恰当终止性(无悬而未决的状态)、无死活动(任何活动都有机会发生)，以确保过程模型的正确性[9,10]。为进一步刻画工作流网的数据流，过程模型 WFD-net[11] 又提出了几类需要避免的数据流错误。

在软件过程领域，软件过程的合理性却很少提及。本文依据 EPMM 对软件过程的形式化定义，提出了软件过程的结构合理性定义，并设计了相应的检验算法。与参考文献[7]一起，构成了相对完整的软件演化过程模型的合理性检验体系(包括任务、活动和软件过程)，有利于提高软件演化过程模型的质量，进而提高软件演化的质量和效率，缩短软件演化的周期。

但是，结构合理性只能从静态角度确保过程结构不发生异常，无法确保过程逻辑的正确性。为此，下一步研究工作拟从动态角度着手，对软件过程的性质合理性进行研究。

参 考 文 献

[1] Lehman M M. Laws of software evolution revisited[C]// Proceedings of the 5th European Workshop on Software Process Technology. London, UK: Springer-Verlag, 1997: 108-124

[2] ISO, IEC. ISO/IEC 12207 standard for information technology-software life cycle processes[S]. 1998

[3] Osterweil L J. Software processes are software too [C]// Proceedings of the 9th international conference on software engineering. Los Alamitos, CA, USA: ACM Press, 1987: 2-13

[4] 王青，李娟. 互联网对软件演化的挑战[J]. 中国计算机学会通讯，2009，5(12)：27-37

[5] 李明树，杨秋松，翟健. 软件过程建模方法研究[J]. 软件学报，2009，20(3)：524-545

[6] Li T. An approach to modeling software evolution processes [M]. Berlin: Springer-Verlag, 2008

[7] 谢仲文，李彤，秦江龙. 基于 EPMM 的任务和活动的规范化研究[J]. 计算机应用与软件，2010，27(5)：20-23

[8] 吴哲辉. Petri 网导论[M]. 北京：机械工业出版社，2006

[9] Van der Aalst W. The application of Petri nets to workflow management [J]. The Journal of Circuits, Systems and Computers, 1998, 8(1): 21-66

[10] Van der Aalst W, van Hee K. Workflow Management: Models, Methods and System [M]. The MIT Press, 2002

[11] Trčka N, Van der Aalst W, Sidorova N. Data-Flow anti-patterns: Discovering data-flow errors in workflows [C]// Proceedings of the 21th International conference on Advanced Information Systems Engineering. Berlin, German: Springer-Verlag, 2009: 425-439