一种基于场景的软件维护性需求分析方法

叶 飞 朱小冬 王毅刚
(石家庄军械工程学院装备指挥与管理系 石家庄 050003)

摘 要 软件维护性是软件质量的重要属性之一,它在很大程度上决定了软件的后期维护成本。设计赋予软件良好的维护性是软件需求、设计和开发经常忽视的一个问题,也是一个难题。提出了一种基于场景的软件维护性需求分析方法,即通过对软件将来可能发生的维护场景进行分析以获取软件可维护性需求。

关键词 软件维护性,软件维护,场景,需求分析
中图法分类号 TP311.5 **文献标识码** A

New Method of Software Maintainability Requirement Analysis Based on Maintenance Scenario Analysis

YE Fei ZHU Xiao-dong WANG Yi-gang
(Department of Management Engineering, Ordnance Engineering College, Shijiazhuang 050003, China)

Abstract Software maintainability is one of important software quality attributes. Poor software maintainability may conduce to great software maintenance cost. Software maintainability requirement analysis and design have often been ignored during the software development process. A scenario based software maintainability requirement analysis method was proposed. Software maintainability requirements can be acquired through software maintenance scenarios analysis.

Keywords Software maintainability, Software maintenance, Scenario, Requirement analysis

虽然软件维护性是软件的一种外部属性,只有通过实际的软件维护才能表现出来,但它也是设计赋予软件的一种固有特性,即软件维护性是设计出来的。软件维护性是通过软件体系结构设计、编码等设计开发环节来实现的,如何在软件设计时获取清晰、一致的维护性需求是软件维护性实现的前提。目前软件需求分析和设计时还没有专门针对软件维护性的需求分析方法,绝大部分还是一些泛泛的维护性要求,如"注释要多"、"模块化要好"等,这些要求在具体落实到软件设计和开发时可操作性差。本文针对软件维护性需求分析获取困难的问题,研究了一种基于维护场景的软件维护性需求分析方法,以获取利于可操作性强的软件维护性需求。

1 相关研究

关于软件维护性需求的研究还比较少。一些学者对软件维护性需求进行了定性描述,如 Jan Bosch[1] 阐述了他关于维护性需求的经验:在软件需求规范中被提及的相当少。一个典型的需求描述的例子是:"系统的维护性应该尽可能的好。" Bass, Clements and Kazman[2] 把维护性需求区分为两种不同的种类:一般需求和特殊需求。一般需求是如何提出好的软件工程原理,以获得高的软件维护性。特殊需求是如何使一组特定的更改变得更容易。

Chung L 的 NFR 框架中[3],软件维护性作为软件重要的非功能属性之一,也可以作为一个"软目标"在 NFR 框架里进行描述,但是 NFR 框架是针对整个非功能需求的建模方法,它比较适合于多种非功能需求之间的综合权衡分析。

还有学者或组织从定量的角度对软件维护性需求进行了研究,OPEN(面向对象过程、环境以及符号的简称)[4] 过程框架定义的维护性需求为面向开发者的、指定必需的可维护性的质量要求。因为可维护性要求很难被定量化,明确期望的目标可能比实际的要求更重要。Peters 和 Pedrycz[5] 提议在软件要求分析中用目标值来量化一个系统的可维护性。

在进行软件维护性需求建模时,必须考虑软件部署后可能发生的软件维护,并分析这些维护对系统的影响,从而获取全面的软件维护性需求信息。而上述研究成果还不能够为软件开发者提供一种对软件维护性需求进行清晰、一致定义的方法。本文引入场景技术,以提供给开发者一种操作性较好的维护性需求分析方法。

2 基于场景的软件维护性需求分析方法基本原理

2.1 软件维护性需求

软件维护性需求到底是什么?怎样表示?这是进行软件维护性需求分析时首先要解决的问题。良好的软件维护性需求必须有很好的设计和实现的可操作性。结合相关文献研究[4],我们定义软件维护性需求为:面向开发者,对软件维护性具有可操作性、清晰和一致的质量要求。一个明确的软件维护性需求信息应该包括以下几部分内容:维护性属性信息、实现信息、约束信息等。根据以上分析,构建了基于3个模板的软件维护性需求模型[6],这3个模板分别是:维护性属性、约

到稿日期:2012-03-09 返修日期:2012-08-04 本文受国防预研项目资助。
叶 飞(1979-),男,博士,讲师,主要研究方向为软件保障、装备保障,E-mail:feiye2005@gmail.com;朱小冬(1965-),男,硕士,教授,主要研究方向为软件保障、装备保障;王毅刚(1975-),男,博士,讲师,主要研究方向为软件保障、装备保障。

束及实现操作模板。它们能够清楚地定义软件维护性需求及实现该需求的措施，以及对维护性需求的各种约束。定义的软件维护性需求元模型如图 1 所示。

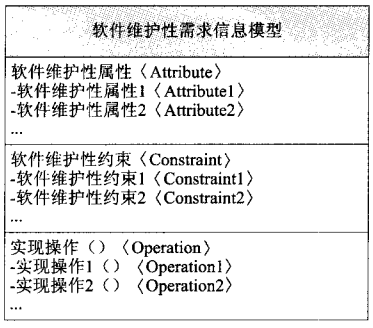

图 1　软件维护性需求信息模型

2.2　基于场景的软件维护性需求分析框架

基于场景的软件维护性需求分析的基本原理如图 2 所示。在软件需求分析和设计阶段研究获取将来可能的维护场景，并分析这些维护场景对软件设计模型的影响，包括影响到的组件及其影响程度；针对这些影响的组件，分析软件设计模型应该具有的软件可维护性需求及应该采取的措施。

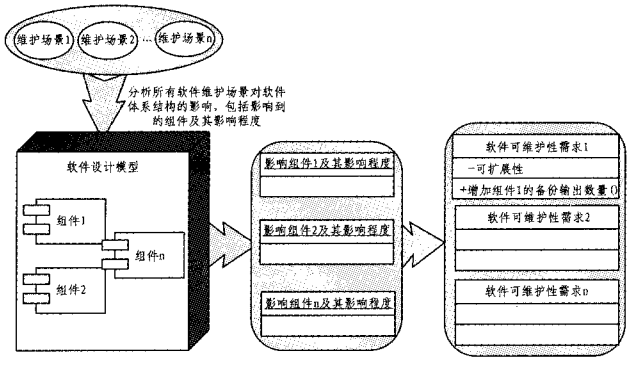

图 2　基于场景的软件维护性需求分析的基本原理

3　软件维护场景及其表示

软件维护场景是对将来软件部署后可能发生的软件维护活动的描述。一个完整的软件维护场景应该包括如下几部分内容：(1)软件维护场景实施的目标(Maintenance Object)；(2)每个维护场景所包含的所有维护活动(Maintenance Action)，维护活动的信息包括：实施更改的类型(如增加、修改、删除等)以及更改的作用对象(构件)；(3)软件维护场景对软件系统体系结构的影响信息(Maintenance Impact)，即软件维护场景能够引起软件系统哪些构件的变化。基于以上分析，软件维护场景可表示为：

Software Maintenance Scenario = 〈ScenarioID, Maintenance Object Set, Maintenance Action Set, Maintenance Impact Set〉

其中，Object Set=$\{o_1, o_2, \cdots, o_n\}$是一个非空有限集合，表示软件维护场景需要实现的目标集，Action Set=$\{a_1, a_2, \cdots, a_k\}$是实现 Object 所需要执行的软件维护活动集，Maintenance Impact Set=$\{c_1, c_2, \cdots, c_m\}$，表示该软件维护场景所影响的构件集。

在软件体系结构层次的软件维护活动主要考虑对构件的更改(*Change_C*)，更改活动主要包括构件增加(*Add_C*)、构件删除(*Del_C*)和构件修改(*Modify_C*) 3 类。

由此，对一个构件的更改活动可以描述为：

Component Maintenance Action = (*Action_ID*, *Change_C*, *changed_Component*, *Change Impact*)

其中，*Action_ID* 是更改活动的编号，*Change_C* 表示实施的更改活动，*changed_Component* 表示 *Change_C* 所作用的对象构件，*Change Impact* 表示本次更改活动对软件体系结构的影响。

4　基于场景的维护性需求分析过程

主要包括两个步骤：首先是分析并获取软件维护场景，并对其进行精确表示；然后是针对各个软件维护场景分析其对软件体系结构的影响，获取软件体系结构的可维护性需求。

4.1　软件维护场景分析获取及表示

一般的维护场景可通过直接和风险承担者(用户、软件设计和开发者、项目管理者、维护者等)交谈得出他们的期望。获得的维护场景集合用 S 来表示，则 $S=\{s_1, s_2, \cdots, s_n\}$，其中 s_i 是各个维护场景。根据上节软件维护场景的表示方法，其表示主要通过两个步骤来完成：(1)分析软件维护场景，获取其全部维护活动；(2)分析各维护活动对体系结构的影响。

4.1.1　软件维护活动获取

软件维护活动的获取是通过对软件维护场景引起的变化跟踪来实现的。如图 3 所示，变化跟踪包括 3 个步骤：①根据维护场景，分析这些维护场景所影响到的用例；②根据变化的用例，确定需要对实现用例的对象进行怎样的更改；③根据变化的对象，确定需要对哪些构件实施更改，这样就可以得到软件体系结构级的所有的软件维护活动。要实现对这些变化信息的跟踪，必须首先实现变化信息的记录，即将每个软件维护场景引起的用例变化、进而引起的对象变化、构件变化信息进行记录。

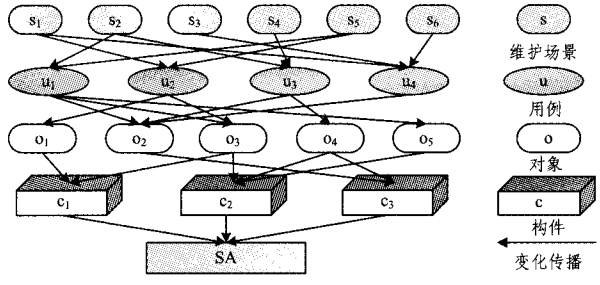

图 3　软件变化传播途径

(1)变化跟踪信息传递

变化信息的记录通过各种变化跟踪矩阵来实现[7]。用户功能需求的变化可以通过维护场景来承载，而用例是若干个功能需求的有效封装。首先建立维护场景与用例之间的关系矩阵，即变化起源跟踪矩阵，如图 4 所示。

	S_1	S_2	S_3	S_4	S_5	S_6
u_1	0	1	0	0	1	0
u_2	1	0	0	0	1	0
u_3	0	1	0	1	0	0
u_4	1	0	1	0	0	1

图 4　用例和维护场景之间的关系阵列示例

进而形成变化起源跟踪矩阵：

$$M_s=\begin{matrix}0&1&0&0&1&0\\1&0&0&0&1&0\\0&1&0&1&0&0\\1&0&1&0&0&1\end{matrix}$$

$m_{s(i,j)}=\begin{cases}1, & \text{当 } u_i \text{ 包含 } s_j \text{ 时}\\0, & \text{当 } u_i \text{ 不包含 } s_j \text{ 时}\end{cases}$，则称 M_s 为变化起源跟踪矩阵。

定义 $M_u=(m_{u(i,j)})$，其中 $m_{u(i,j)}$ 表示对象 o_i 与用例 u_j 之间的实现关系（$1\leqslant i\leqslant p$，$1\leqslant j\leqslant l$），并且 $m_{u(i,j)}=\begin{cases}1, & \text{当 } o_i \text{ 与 } u_j \text{ 的实现有关时}\\0, & \text{当 } o_i \text{ 与 } u_j \text{ 的实现无关时}\end{cases}$，则称 M_u 为变化对象跟踪矩阵。

例如(续上例)，假设系统包含的对象为 o_1，o_2，o_3，o_4 和 o_5，其中 o_5 和 o_6 是新增的对象，如果 u_1 由对象 o_2，o_3 和 o_5 实现，u_2 由对象 o_1 和 o_3 实现，u_3 由 o_2 和 o_4 实现，u_4 由 o_1 实现，则：

$$M_u=\begin{bmatrix}0&1&0&1\\1&0&1&0\\1&1&0&0\\0&0&1&0\\1&0&0&0\end{bmatrix}$$

定义 $M_c=(m_{c(i,j)})$，其中 $m_{c(i,j)}$ 表示构件 c_i 包含对象 o_j（$1\leqslant i\leqslant o, 1\leqslant j\leqslant p$），并且 $m_{c(i,j)}=\begin{cases}1, & \text{当 } c_i \text{ 包含 } o_j \text{ 时}\\0, & \text{当 } c_i \text{ 不包含 } o_j \text{ 时}\end{cases}$，则称 M_c 为变化构件跟踪矩阵。

例如(续上例)，假设系统包含的构件为 c_1，c_2 和 c_3，如果 c_1 包含 o_1 和 o_4，c_2 包含 o_3 和 o_5，c_3 包含 o_2 和 o_4。则：

$$M_c=\begin{bmatrix}1&0&0&1&0\\0&0&1&0&1\\0&1&0&1&0\end{bmatrix}$$

(2) 变化跟踪实现及软件维护活动的获取

在变化信息记录的基础上，可以通过变化矩阵的运算来实现变化跟踪，在此基础上确定每个维护场景所影响到的构件和需要实施的维护活动(如增加、修改、删除等)。

$$M_{s\text{-}u\text{-}c}=M_c\times M_{s\text{-}u}=M_c\times(M_u\times M_s)=(m_{s\text{-}u\text{-}c(i,j)})$$

式中，$m_{s\text{-}u\text{-}c(i,j)}=\sum_{x=1}^{k}(m_{c(i,x)}\times m_{s\text{-}u(x,j)})$，$m_{c(i,x)}$ 和 $m_{s\text{-}u(x,j)}$（$i=1,2,\cdots,l, j=1,2,\cdots,m$）分别为 M_c 和 $M_{s\text{-}u}$ 的元素。

其次，判断功能变化所影响的构件，当矩阵 $M_{c\text{-}s\text{-}u}$ 中元素值 $m_{s\text{-}u(x,j)}\neq 0$ 时，则维护场景 s_j 必将引起构件 c_i 的变化。$M_{c\text{-}s\text{-}u}$ 中第 j 列非 0 元素个数表示维护场景 s_j 直接影响的构件数，第 i 行非 0 元素个数表示直接引起改变构件 c_i 的维护场景数。

例如(续上例)，由于：

$$M_{s\text{-}u}=M_c\times M_s=\begin{bmatrix}0&1&0&1\\1&0&1&0\\1&1&0&0\\0&0&1&0\\1&0&0&0\end{bmatrix}\times\begin{bmatrix}0&1&0&0&1&0\\1&0&0&0&1&0\\0&1&0&1&0&0\\1&0&1&0&0&1\end{bmatrix}=\begin{bmatrix}2&0&1&0&1&1\\0&2&0&1&1&0\\1&1&0&0&2&0\\0&1&0&1&0&0\\1&0&1&0&0&1\end{bmatrix}$$

$$M_{s\text{-}u\text{-}c}=M_c\times M_{s\text{-}u}=\begin{bmatrix}1&0&0&1&0\\0&0&1&0&1\\0&1&0&1&0\end{bmatrix}\times\begin{bmatrix}2&0&1&0&1&1\\0&2&0&1&1&1\\1&1&0&0&2&0\\0&1&0&1&0&0\\0&1&0&0&1&0\end{bmatrix}=\begin{bmatrix}2&1&1&1&1&1\\1&2&0&0&3&0\\0&3&0&2&1&0\end{bmatrix}$$

则对于矩阵 $M_{s\text{-}u\text{-}c}$ 中的第 1 列来说，维护场景 s_1 必将引起 c_1 和 c_2 的变化(其他列类同)；对第 2 行来说，引起构件 c_2 变化的可能维护场景为 s_1，s_2 和 s_5(其他行类同)。

这样通过变化跟踪矩阵的计算可以最终确定实现每一个维护场景所直接影响到的构件，从而得到维护场景 s_i 的维护活动集，其中包括了增加构件、修改构件和删除构件等活动。

4.1.2 软件维护活动对软件体系结构的影响程度获取

构件之间存在很多的相互关系，对某一个构件的更改会影响到与之相关的构件，可能也需要对这些构件进行更改，这又称为软件维护的波及效应。通过软件波及效应分析可初步确定实现某个软件维护场景对软件体系结构影响程度的大小。

获取软件维护场景对软件体系结构的影响程度分两步：①构建软件系统体系结构的可达矩阵，通过可达矩阵，可以分析软件构件之间的可达性；②在构建 SA 可达矩阵的基础上，根据每个软件维护场景的维护活动集，确定各维护活动的构件的变化对 SA 的影响程度大小。

(1) SA 可达矩阵构建

设 SA 的构件关系矩阵 $M_R=(C_{ij})$，其中 C_{ij} 表示构件 C_i 与构件 C_j 之间的连接件，$1\leqslant i,j\leqslant P$($P$ 为 SA 中构件总数)，并且 $C_{ij}=\begin{cases}1, & \text{当 } C_i \text{ 与 } C_j \text{ 之间直接通过连接件连接时}\\0, & \text{当 } C_i \text{ 与 } C_j \text{ 之间不存在直接连接件连接时}\end{cases}$；设 M_{R^*} 为 SA 的可达矩阵，则：

$$M_{R^*}=(C_{kl})=\begin{cases}1, & \text{当 } C_k \text{ 与 } C_l \text{ 之间可达时}\\0, & \text{当 } C_k \text{ 与 } C_l \text{ 之间不可达时}\end{cases}$$

式中，$1\leqslant k,l\leqslant P$，$P$ 为 SA 中构件总数。

$$M_R=\begin{bmatrix}0&1&1&1&0\\0&0&1&0&0\\0&1&0&0&0\\0&0&1&0&0\\1&0&0&0&0\end{bmatrix}\quad M_{R^*}=\begin{bmatrix}0&1&1&1&0\\0&1&1&0&0\\0&1&1&0&0\\0&1&1&0&0\\1&1&1&1&0\end{bmatrix}$$

如果第 j 个构件被修改，则会影响到可达矩阵中第 j 列中所有为 1 的行的构件，即上述可达矩阵中，如果 C_1 修改，则只会影响到 C_5（$C_{51}=1$），C_2 修改，则会影响到所有的构件。在这里我们定义第 j 个构件的修改影响度($Impact_j$)如下：

$$Impact_j=\sum_{i=1}^{n}C_{ij}$$

修改影响度表示了第 j 个构件对系统的影响程度，$Impact_j$ 越大，表示该构件影响到的构件越多，对系统的影响程度越大。

(2)分析各个软件维护场景的维护活动对 SA 的影响

这里主要考虑 3 种软件维护活动：删除构件、增加构件、修改构件，具体分析如下：

• 删除构件 C_i，如果可达矩阵的第 i 列全为 0，则可直接删除 C_i，而不会对其他构件造成影响(但会影响 SA 的结构)。如果第 i 列至少存在一个 1，则删除 C_i 对 SA 的结构和整个软件的功能及性能都会产生影响，其影响的相对大小为第 i 列元素之和。

• 增加构件 C_i，首先要提供与之有直接交互关系的构件，进而形成新的可达矩阵，最后在 SA 范围内判定被影响的全部构件。其影响的相对大小是新的可达矩阵的第 i 列元素之和。

• 修改构件 C_i，修改包括对自身结构的调整和功能的增减，可能会导致 SA 在结构上或语义上的变化，需要针对新 SA 的可达矩阵进行影响分析。

在以上的构件更改分析的基础上对各个软件维护场景的维护活动进行分析，则可以得到各个软件维护场景对软件体系结构的影响程度，获取算法如下：

```
GetChangeImpact()
输入：维护场景集(S)各个维护活动集(Ai)及 SA 的可达矩阵
输出：软件各个维护场景(si)对 SA 的影响程度
Begin
n=软件维护场景个数
For i=1 to n //初始化各维护场景对 SA 的 Impact
  si. Impact=0
End for
For i=1 to n do
{
  mi=第 i 个维护场景的维护活动个数
  for j=1 to mi do
  {
  Impactij= 第 i 个维护场景的第 j 个维护活动对软件体系结构的影响度(可以通过上面的分析方法来获得)
  Changed_Componentij={第 k 个构件|如果可达矩阵的第 j 列第 k 行为 1}
  }
  si. Impac=∑impactij
}
End GetChangeImpact
```

通过上述两个步骤就可以获得维护场景的完整表示。

4.2 软件维护性需求获取

软件维护性需求是在对已经获取的软件维护场景的详细分析的基础上获取的，其基本思路是：针对各个维护场景 s_i 对软件体系结构的影响程度(s_i. *Impact*)，给定一个影响程度阈值(*Threshold*)，如果 s_i. *Impact*>*Threshold*，则表示该维护场景的实施将严重影响软件体系结构的稳定性，要严格限制此类维护事件的发生。为此，在体系结构设计的时候，需要针对该维护场景更改体系结构设计以适应这种变化，这就是一个维护性需求。具体的维护性需求获取方法如下：

```
GetMaintainabilityRSet()
输入：软件维护场景集 S={s1,s2,…,sn}
输出：维护性需求模型集 Maintainability Requirements Set(MRS)=
      {MiRj}
Begin
    MRS=Φ  //初始化维护性需求集
    Threshold=T //定义一个维护影响度阈值
    j=1
    for i=1 to n
    {
      If si. Impact>Threshold
      {
        获取该维护场景的所有维护活动所影响到的构件；
        分析减少影响的措施，即获得维护性需求模板的操作方法；
        分析该操作方法对其它属性的影响；
        MRS=MRS ∪ MiRj
        j=j+1
      }
    }
End GetMaintainabilityRSet
```

通过以上的算法分析，可以获取针对这些维护场景的所有软件维护性需求集合。

5 实例研究

实例系统是作者参与开发的基于软件黑匣子的集成软件故障诊断系统。在工程化开发过程中，运用提出的软件维护性需求分析方法进行了应用研究。实例系统主要构件图如图 5 所示，SourceAmalysis 包的类图如图 6 所示。

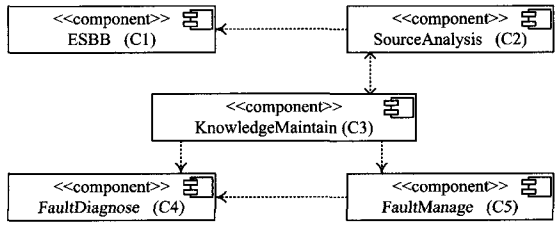

图 5 实例系统主要构件图

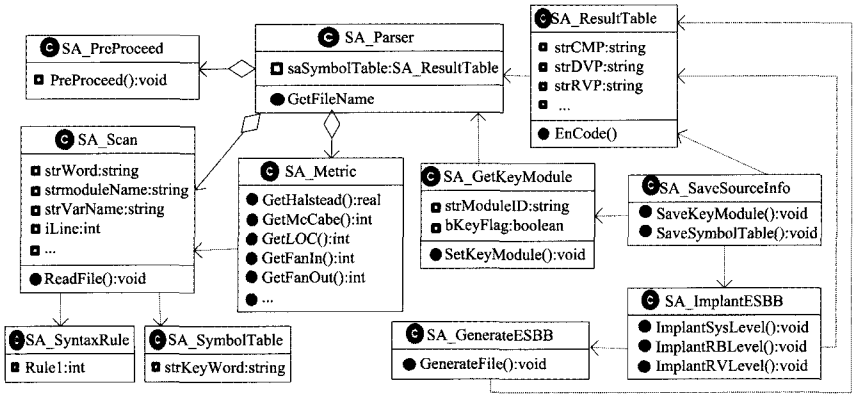

图 6 SourceAnalysis 包的类图

(1)获取初始的软件维护场景

通过与各个风险承担者的沟通，按照各个风险承担者(角色)划分，获得的初始维护场景如表1所列。

表1 初始的软件维护场景

序号	角色	维护场景描述	权重
s_1	用户	增加对多种语言源程序故障诊断的支持	0.3
s_2		添加实时诊断功能	0.10
S_3	系统管理员	增加日志管理	0.05
…		…	

(2)软件维护场景表示

以维护场景 s_1 为例来说明本文方法的具体应用。第1个维护场景是用户提出的，其目的是使系统支持更多语言的源代码分析和故障诊断(原来只支持C和C++语言)，现在要增加对Java、VB等语言的支持功能。通过变更传播分析(s_1→用例确定源程序分析器→SourceAnalysis模块)可以看出，要实现维护场景 s_1，需要对构件SourceAnalysis进行更改，由图5系统的组件图可以得到它的邻接关系矩阵 M，并求出它的可达矩阵 M^+。

$$M=\begin{bmatrix}0&1&0&0&0\\0&0&1&0&0\\0&1&0&0&0\\0&0&1&0&1\\0&0&1&0&0\end{bmatrix}\quad M^+=\begin{bmatrix}1&1&1&0&0\\0&1&1&0&0\\0&1&1&0&0\\0&1&1&1&1\\0&1&1&0&1\end{bmatrix}$$

由可达矩阵 M^+ 可以看出，对构件SourceAnalysis(C_2)和KnowledgeMaintain(C_3)的更改将影响到其它所有的组件。通过上述分析，我们可以将维护场景 s_1 表示为：

(增加多种语言的支持，Action＝(Change_C，SourceAnalysis component，(C_1，C_2，C_3，C_4，C_5))，5)

用三模板[6]来表示由场景 s_1 构建的维护性需求模型如下：

```
attribute Expandability parent maintainability
{
  component system,SourceAnalysis component
  children Modularity,Flexibility
  contribution oneX
}
  Constraint good_ Expandability
{
  constraints
  Modularity [strong ]
  Flexibility [strong ]
…
  priorities
  Modularity [high]
  Flexibility [high]
}
Operation add SA_LanguageSelect Class
{
  affected Modularity
  implemented Flexibility
  effect
  Modularity [+3]
}
```

结束语 本文针对当前软件维护性需求获取困难、缺乏维护性分析设计方法的难题，提出了一种基于软件维护场景的维护性需求分析方法，该方法能够帮助软件需求分析和设计人员在软件开发早期把握软件维护性需求，进而设计赋予软件良好的维护性。实例应用表明，该方法能够为开发人员提供一种较好的维护性需求分析方法，我们将在下一步工作中对方法进行深入实践和不断改进。

参考文献

[1] Bosch J. Design & Use of Software Architectures: Addison Wesley[M]. 2000

[2] Bass L, Clements P, Kazman R. Software Architecture in Practice: Addison-Wesley[M]. 1998

[3] Mylopoulos J, Chung L, Nixon B A. Representing and using nonfunctional requirements: a process-oriented approach [J]. IEEE Transactions on Software Engineering, 1992, 18(6): 483-497

[4] Firesmith. OPEN Process Framework (OPF) [OL] http://www.donald-firesmith.com, 2004

[5] Peters J F, Pedrycz W. Software Engineering, an Engineering Approach[M]. Wiley, 2000

[6] 叶飞，朱小冬，王毅刚. 基于XML的软件非功能需求建模研究[J]. 微计算机信息，2008，24(3)：250-252

[7] 王映辉，张世琨，刘瑜，等. 基于可达矩阵的软件体系结构演化波及效应分析[J]. 软件学报，2004，15(8)：1107-1115

(上接第165页)

[10] Grottke M, Nikora A, Trivedi K. An Empirical Investigation of Fault Types in Space Mission System Software[C]// IEEE/IFIP International Conference on Dependable Systems and Networks. Chicago, USA, 2010: 447-456

[11] Dohi T, Goseva-Popstojanova K, Trivedi K S. Statistical nonparametric algorithms to estimate the optimal software rejuvenation schedule[C]// Proceedings of the Pacific Rim International Symposium on Dependable Computing (PRDC2000). Los Angeles, USA, 2000: 77-84

[12] Pfening A, Garg S, Puliafito A, et al. Optimal software rejuvenation for toleration software failures[C]// Proceedings of the 18th International Symposium on Performance Evluation. Lausanne, Switzerland, 1996: 491-506

[13] Garg S, Huang Y, Kintala C, et al. Time and load based software rejuvenation: Policy, evaluation and optimality[C]// Proceedings of the 1st Fault Tolerance Symposium, FTS-95. Madras, India, 1995: 22-25

[14] Andrzejak A, Silva L. Robust and adaptive modeling of software aging[OL]. http://citeseerx.ist.psu.edu/viewdoc/summary, 2011-07

[15] Bozdogan H. Model selection and Akaike's information criterion (AIC): the general theory and its analytical extension [J]. Psychometrika, 1987, 52: 345-570

[16] Controneo D, Natella R, Pietrantuono R, et al. Software aging analysis of the Linux Operating System[C]// 2010 IEEE 21st International Symposium on Software Reliability Engineering. San Jose CA, USA, 2010: 71-80