一种软件需求变化追踪方法

王映辉[1,2], 王立福[2], 张世琨[2], 王琼芳[1]
(1 陕西师范大学计算机学院, 陕西西安 710062; 2 北京大学软件研究所, 北京 100871)

摘 要: 有效的软件变化追踪方法是实施软件演化的关键. 基于需求信息传播与建模、需求变化信息传播路径和需求变化信息跟踪方法三个层面, 阐述了软件变化跟踪的整体过程框架; 给出了基于变化起源跟踪矩阵、变化对象跟踪矩阵、变化构件跟踪矩阵、以及可达矩阵的功能变化传播记录方法; 凭借矩阵运算, 描述了功能变化跟踪的具体实现, 并给出了实施变化所付出代价高低的初步判定方法. 对软件维护甚至软件演化研究具有一定的借鉴意义.

关键词: 场景; 用况; 变化传播; 软件演化; 变化追踪
中图分类号: TP311 13 文献标识码: A 文章编号: 0372-2112 (2006) 08-1428-05

A Tracing Approach of Software Requirement Change

WANG Ying-hui[1,2], WANG Li-fu[2], ZHANG Shi-kun[2], WANG Qiong-fang[1]
(1. School of Computer Science, Shaanxi Normal University, Xi'an, Shaanxi 710062, China;
2. Institute of Software, Peking University, Beijing 100871, China)

Abstract It's a key of software evolution to achieve available method of software change-trace. A whole trace-process framework of requirement change is drawn based on three tiers which are propagation & modeling, propagation path, and trace approach of requirement change. Record method about propagation of function-requirement change is addressed based on trace-matrix of change-source, of change-object and of change component. Implementation of change-trace based matrix calculation is described, and a simple cost decision method of software change-implementation is brought forward. It is advantage to implement software maintenance as well as software evolution.

Key words scenario; use case; change propagation; software evolution; change-trace

1 引言

构造性和演化性是软件的两个基本特性[1]. 仅软件演化来说, 对软件变化的把握和控制是其关键. 引起软件变化的原因是多方面的, 如基础设施的改变、功能需求的增减、高性能算法的发现、技术环境因素的变化等等. 所以, 对软件变化进行理解和控制显得比较困难.

在诸多用户需求中, 功能需求是最为核心的需求, 其他需求都是围绕功能需求引出的[2,3]. 因此, 在诸多引起软件变化的原因中, 功能需求变化是最主要的, 对它的追踪和捕捉应是把握需求变化的关键.

在功能需求的捕捉与建模中, 基于场景 (scenario)的用况 (use case)得到了广泛应用, 已成了用户功能需求建模的有效手段. "用况驱动"软件开发[4]是 RUP(Rational Unified Process)的主要特征之一, 其包括在进行需求分析时, 用况可以指导系统功能的定义; 在进行设计时, 用况集中体现了软件的主要行为结构, 是 SA (Software Architecture)建模的主要依据; 在进行系统测试时, 用况能较好地指导测试用例的开发; 在进行系统维护时, 用况有利于对现有 SA 的分析, 并支持系统的不断演化.

综观软件变化研究的历史, 早在 1978年人们就开始了软件变化传播的研究[5]. 在 20世纪 90年代, 人们对软件变化甚至演化作了一定的探讨, 诸如 Bohner给出了软件变化分析的过程框架[6,7], Baxter等人提出了通过将设计信息延伸到维护过程中来掌握软件的变化[8], Chaumun等人提出的类层次影响计算方法[9], Zhou对面向对象系统中变化的影响类型、途径、程度和范围问题做了比较详尽地研究[10]. Wang等对软件的静态演化和动态演化进行

收稿日期: 2005-09-02; 修回日期: 2006-05-20
基金项目: 国家 863高技术研究发展计划 (No. 2001AA113171); 国家 973重点基础研究发展规划 (No. 2002CB312006); 国家博士后基金 (No. 20040350251)

了分析,并给出了统一的软件演化描述模型[11],并对软件功能需求变化传播机理及其性质做了研究. 目前, 对软件变化的研究日益增多, 但是如何具体跟踪软件变化的研究相对比较薄弱,而这正是实施软件变化甚至研究软件演化的关键.

2 相关概念

(1)场景与用况

在软件开发中,用户的需求有许多, 诸如技术需求、运行环境需求、功能需求等. 其中功能需求是用户需求中最基本的需求, 也是最核心的需求, 其他需求都是围绕功能需求提出的. 也就是说, 如何记录和跟踪功能需求及其变化是解决需求变化的关键. 功能需求的变化跟踪研究是本文的重点.

场景是对所期望的系统中某个使用情况的简短描述. 而用况是对一组相关联的场景的封装, 也是对使用者与系统之间交互的一种描述. 用况通过场景来捕捉用户功能需求,场景的变化自然而然承载和再现了用户功能需求的变化. 场景为用户功能需求捕捉提供了良好的机制.

从用况使用者角度来看, 用况描述了不同条件下系统对使用人员的请求所做出的响应[2]. 从用况本身来看, 它是对用户功能需要的一种捕捉和建模手段. 用况由一系列的行为(action)组成, 在不同的条件下, 系统会执行不同的行为系列.

(2)功能需求建模

用况是目前用户功能需求获取和建模比较理想的工具,它立足于项目相关人员易于理解的角度对用户的功能需求进行描述和记载. 虽然说, 用况建模不能穷尽所有的用户需求, 但却为功能需求提供了很好的支持, 并得到了广泛地应用. 因而, 通过用况的变化来记载和传递用户的功能需求的变化是一种非常自然的想法. 具体来说, 用户需求的变化必然会体现在用况中场景的变化上. 本文是在场景这一层面上来考察需求变化的起源, 并跟踪和把握这种变化的. 如何基于场景撰写有效用况来对用户功能需求建模的细节, 可参阅文献[2, 12].

(3)用况对象识别

一般情况下, 可以将用况相关的实体或事物简单地当作实现用况的对象,如“处理定单”用况中的“定单”. 但是, 由于用况与对象属于需求分析和系统设计这两个不同层面的概念, 用况的建立与对象没有必然的联系, 所以, 许多实现用况的对象不能从用况中直接获取. 然而一种可行的方法是结合面向对象分析技术[13,14], 将用况中的动作序列按照内聚性进行分类和划分, 并作为设计中所获得的对象的行为属性, 从而建立用况和对象之间的联系.

用况模型能很好地捕获系统的动态结构, 而对象模型更有利于描述系统的静态结构. 用况模型为对象模型提供所需信息, 对象模型来实现用况模型所描述的功能, 两者的结合为问题的完整地分析和系统行为的全面描述提供了有效支持.

3 需求变化跟踪的整体描述

为了方便下文的描述, 本节从变化传播途径和变化跟踪的整体框架两个方面给出需求变化跟踪的整体框架描述. 其中, 变化传播途径展示了需求变化信息传播所经过的实体及其先后关系; 而变化跟踪的整体框架表明了需求信息传播与建模层、需求变化信息传播路径层和需求变化信息跟踪方法层之间的关系.

3.1 变化传播途径

在软件生命周期中, 软件变化起源于用户需求的变化,基于面向对象技术, 这种变化是通过场景、用况、对象、构件和 SA 进行传播的. 为了能更清晰地表达笔者的思想, 先给出一个实例系统(如图 1), 假设: 该系统基于 6个场景 s_1、s_2、s_3、s_4、s_5 和 s_6,并在这些场景的基础上形成了 4个用况 u_1、u_2、u_3 和 u_4 来捕捉和记载用户的需求,而实现用况的对象共 5个: o_1、o_2、o_3、o_4 和 o_5, 由对象组合成的构件共包括 3个: c_1、c_2 和 c_3, 该系统的体系结构为 SA 1. 图 1表达了该实例系统在需求变化信息传播途径中场景、用况、对象、构件和 SA 之间的关系.

由图 1可以看出, 对软件的变化跟踪, 可通过捕捉用况中变化的场景、追踪变化场景所引起的变化对象、标识包含变化对象的构件, 最后圈定 SA 中构件变化影响范围等活动来实现.

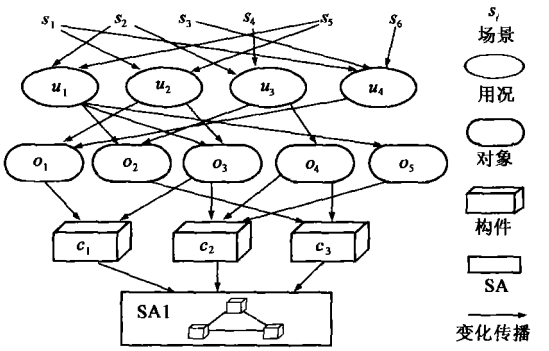

图 1 变化信息传播途径示例

3.2 变化跟踪的整体框架

跟踪与实现的整体框架如图 2所示. 从纵向上看, 图 2包含了三个层面. 其中, 第一个是需求信息传播与建模层面, 表示了软件需求信息的记载与实现是通过用户需求模型(包含场景的用况模型)、到系统分析模型(面向对象的对象模型)、直到 SA 设计模型的变化信息传递与建模来完成的; 第二个是需求变化信息传播路径层面, 表示了软件需求的变化是通过变化的场景、到变化的用况、再到变化的对象、直到变化的构件来传递的; 第三个是需求变化信息跟踪方法层面, 表示了变化信息的追踪实现是依赖于变化起源跟踪矩阵、变化对象跟踪矩阵和变化构

件跟踪矩阵、以及通过这些矩阵的运算来完成的. 另外, 进一步可以看出, 第二、三层面是以第一层面为基础的.

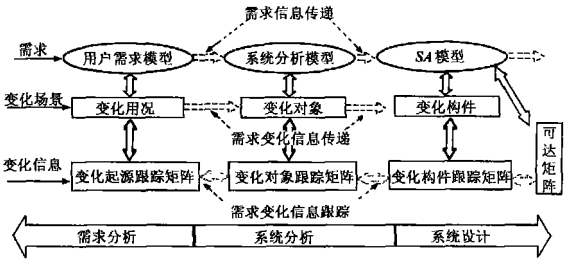

图 2 变化跟踪实现的整体过程

此外, 从横向上看, 以上三个层面贯穿了软件生命周期的三个关键过程: 用户需求分析过程、系统分析过程和系统设计过程.

4 变化追踪实现

4.1 变化信息记载

本节先给出变化信息的起源, 以及在传播过程中变化信息的记载工具, 即各种变化跟踪矩阵.

用户功能需求的变化可以通过场景来承载, 而用况是若干个场景的有效封装. 首先建立场景与用况之间的关系矩阵, 即变化起源跟踪矩阵.

设某一软件系统, 其场景集合 $S=\{s_1, s_2, \cdots, s_n\}$, 用况集合 $U=\{u_1, u_2, \cdots, u_n\}$, 定义 $\boldsymbol{M}_s=(m_{s(i,j)})$, 其中 $m_{s(i,j)}$ 表示用况 u_i 与场景 s_j 之间的包含关系 ($1\leqslant i\leqslant m$, $1\leqslant j\leqslant n$), 并且 $m_{s(i,j)}=\begin{cases}1, & \text{当 } u_i \text{ 包含 } s_j \text{ 时;}\\ 0, & \text{当 } u_i \text{ 不包含 } s_j \text{ 时;}\end{cases}$ 则称 $\boldsymbol{M}_s$ 为变化起源跟踪矩阵.

例如, 根据图 1, 系统的场景集合为 $S=\{s_1, s_2, s_3, s_4, s_5, s_6\}$, 用况集合 $U=\{u_1, u_2, u_3, u_4\}$, 且 u_1 包含场景 s_2 和 s_5, u_2 包含场景 s_1 和 s_5, u_3 包含场景 s_2 和 s_4, u_4 包含场景 s_1、s_3 和 s_6, 则建立如图 3所示的关系阵列. 在变化起源跟踪矩阵 $\boldsymbol{M}_s$ 中, 如第 1行第 2列中的“1”表示用况 u_1 包含场景 s_2, 而第 1行第 4列表示用况 u_1 不包含场景 s_4.

由于用户功能需求的变化直接体现在场景上. 所以, 可以通过矩阵 $\boldsymbol{M}_s$ 非常方便地定位变化所影响到的用况. 如场景 s_3 变化了, 用况 u_4 一定会发生变化. 下面给出变化对象跟踪矩阵.

	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

进而形成变化起源跟踪矩阵 $\boldsymbol{M}_s=\begin{bmatrix}0&1&0&0&1&0\\1&0&0&0&1&0\\0&1&0&1&0&0\\1&0&1&0&0&1\end{bmatrix}$

图 3 用况和场景之间的关系阵列示例

设软件系统, 其用况集合 $U=\{u_1, u_2, \cdots, u_n\}$, 对象集合 $O=\{o_1, o_2, \cdots, o_k\}$, 定义 $\boldsymbol{M}_u=(m_{u(i,j)})$, 其中 $m_{u(i,j)}$ 表示对象 o_i 与用况 u_j 之间的实现关系 ($1\leqslant i\leqslant n$, $1\leqslant j\leqslant k$), 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 $\boldsymbol{M}_u$ 为变化对象跟踪矩阵.

在面向对象系统中, 用况描述的用户功能最终是通过一个对象或多个对象之间的协作来完成的. 所以, 一个用况可以与若干个对象相关联.

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, 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. 则建立如图 4所示的关系阵列:

	u_1	u_2	u_3	u_4
o_1	0	1	0	1
o_2	1	0	1	0
o_3	1	1	0	0
o_4	0	0	1	0
o_5	1	0	0	0

进而形成变化对象跟踪矩阵 $\boldsymbol{M}_u=\begin{bmatrix}0&1&0&1\\1&0&1&0\\1&1&0&0\\1&0&0&0\end{bmatrix}$

图 4 用况和对象之间的关系阵列示例

同样, 由矩阵 $\boldsymbol{M}_u$ 可方便地判定实现变化用况的对象. 下面给出变化构件跟踪矩阵.

设软件系统, 其对象集合 $O=\{o_1, o_2, \cdots, o_k\}$, 构件集合 $C=\{c_1, c_2, \cdots, c_l\}$, 定义 $\boldsymbol{M}_c=(m_{c(i,j)})$, 其中 $m_{c(i,j)}$ 表示构件 c_i 包含对象 o_j ($1\leqslant i\leqslant k$, $1\leqslant j\leqslant l$), 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 $\boldsymbol{M}_c$ 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. 则建立如图 5所示的关系阵列:

	o_1	o_2	o_3	o_4	o_5
c_1	1	0	0	1	0
c_2	0	0	1	0	1
c_3	0	1	0	1	0

进而形成变化构件跟踪矩阵 $\boldsymbol{M}_c=\begin{bmatrix}1&0&0&1&0\\0&0&1&0&1\\0&1&0&1&0\end{bmatrix}$

图 5 对象和构件之间的关系阵列示例

同样, 由矩阵 $\boldsymbol{M}_c$ 非常容易判定变化对象所影响的构件. 下面继续讨论软件体系结构 (SA)中构件之间的关系.

根据文献[11]对 SA网络的定义, 可得 SA邻接矩阵:

$\boldsymbol{M}_R=(C_{ij})$, 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

$C_{ij}=\begin{cases}1, & \text{当 } C_i \text{ 与 } C_j \text{ 之间直接通过连接件连接时;}\\ 0, & \text{当 } C_i \text{ 与 } C_j \text{ 之间不存在直接连接件连接时;}\end{cases}$

其中, 称 C_i 和 C_j 分别为行表头构件和列表头构件.

另外, 设 SA中构件组成的集合 $X=\{C_1, C_2, \cdots, C_P\}$, 则邻接矩阵 $\boldsymbol{M}_R$ 对应的关系: $R\subseteq X^2$.

于是, 关系 R的传递闭包 $R^+=R\cup R^2\cup\ldots\cup R^P$, 则

R^+ 对应的矩阵 $M_{R^+}=M_R \vee M_R^2 \vee \cdots \vee M_R^P=\bigvee_{k=1}^{P} M_R^k$, 其中 $M_{R_i}^i=M_{R_{i-1}}^{i-1} \vee M_R$, (i= 2, 3, …, P); 则称 M_{R^+} 为 SA 的可达矩阵, 简称可达矩阵.

此外, $M_{R^+}=(C_{kl})=\begin{cases}1, \text{当 } C_k \text{ 与 } C_l \text{ 之间可达时;}\\ 0, \text{当 } C_k \text{ 与 } C_l \text{ 之间不可达时;}\end{cases}$ (1≤k, l≤P, P 为 SA 中构件总数).

在 SA 可达矩阵 M_{R^+} 中, $\forall C_j$ (列表头构件), 如果 $\exists C_{ij}=1$, 则自 C_j 到 C_i (行表头构件)之间可达; 否则不可达.

由可达矩阵可以方便地圈定一个构件变化所影响的其他构件.

例如 (续上例), 假设列 c_1 到行 c_2 可达, 列 c_2 到行 c_3 可达, 列 c_3 到行 c_1 和 c_2 分别可达. 则建立 c_1、c_2 和 c_3 之间的邻接阵列如图 6所示.

	c_1	c_2	c_3
c_1	0	0	1
c_2	1	0	1
c_3	0	1	0

进而形成 SA 邻接矩阵 $\boldsymbol{M}_{cc}=\begin{bmatrix}0&0&1\\1&0&1\\0&1&0\end{bmatrix}$

图 6　构件之间的关系阵列示例

4.2　变化跟踪实现步骤

本节在以上各种变化跟踪矩阵的基础上, 继续讨论变化跟踪的具体实现步骤.

step 1　确定功能需求变化所影响的用况

由于场景可直接承载功能需求变化, 所以根据变化起源跟踪矩阵 $\boldsymbol{M}_s$ 可直接确定功能变化所影响的用况.

此外, 矩阵 $\boldsymbol{M}_s$ 中第 j 列元素之和表示场景 s_j 变化直接影响到的用况总数, 第 i 行元素之和表示直接引起用况 u_i 发生变化的可能场景数.

step 2　确定功能需求变化所影响的对象

首先, 计算 $\boldsymbol{M}_{su}=\boldsymbol{M}_u \times \boldsymbol{M}_s=(m_{su(i,j)})$, 其中 $m_{su(i,j)}=\sum_{x=1}^{n}(m_{u(i,x)} \times m_{s(x,j)})$, $m_{u(i,x)}$ 和 $m_{s(x,j)}$ 分别为 $\boldsymbol{M}_u$ 和 $\boldsymbol{M}_s$ 的元素, i= 1, 2, …, k, j= 1, 2, …, m.

其次, 再判断功能需求变化所影响的对象. 当矩阵 $\boldsymbol{M}_{su}$ 中元素值为 $m_{su(i,j)} \neq 0$ 时, 则场景 s_i 变化必将引起对象 o_j 的变化. $\boldsymbol{M}_{su}$ 中第 j 列非 0元素个数表示场景 s_i 变化所直接影响的对象数, 第 i 行非 0元素个数表示直接引起对象 o_i 改变的用况数.

例如 (续上例), 由于:

$$\boldsymbol{M}_{su}=\boldsymbol{M}_u \times \boldsymbol{M}_s=\begin{bmatrix}0&1&0&1\\1&0&1&0\\1&1&0&0\\0&0&1&0\\1&0&0&0\end{bmatrix}\times\begin{bmatrix}0&1&0&0&1&0\\1&0&0&0&1&0\\0&1&0&1&0&0\\1&0&1&0&0&1\end{bmatrix}$$

$$=\begin{bmatrix}2&0&1&0&1&1\\0&2&0&1&1&0\\1&1&0&0&2&0\\0&1&0&1&0&0\\0&1&0&0&1&0\end{bmatrix}$$

则对于矩阵 $\boldsymbol{M}_{su}$ 中的第 1列来说, 场景 s_1 的变化必将引起 o_1 和 o_3 的变化 (其他列类同), 对第 1行来说, 引起对象 o_1 变化的可能场景为 s_1、s_3、s_5 和 s_6 (其他行类同).

此外, 根据矩阵 $\boldsymbol{M}_{su}$ 中元素之值来判定每条变化 (即每个变化的场景)信息传递的途径和范围. 例如, 上例中矩阵 $\boldsymbol{M}_{su}$ 的第 2行第 2列的元素值为 2, 则表示变化信息自用况 s_2 传递到对象 o_2, 存在 2条路径 $s_2 \rightarrow u_1 \rightarrow o_2$ 和 $s_2 \rightarrow u_3 \rightarrow o_2$. 由于功能需求变化自场景传递到对象是以用况为桥梁的, 所以此值越大, 表示变化信息影响的范围越大, 实施这种变化付出的代价越高.

step 3　确定功能需求变化所影响的构件

首先, 计算 $\boldsymbol{M}_{su\text{-}c}=\boldsymbol{M}_c \times \boldsymbol{M}_{su}=\boldsymbol{M}_c \times (\boldsymbol{M}_u \times \boldsymbol{M}_s)=(m_{su\text{-}c(i,j)})$, 其中 $m_{su\text{-}c(i,j)}=\sum_{x=1}^{k}(m_{c(i,x)} \times m_{su(x,j)})$, $m_{c(i,x)}$ 和 $m_{su(x,j)}$ 分别为 $\boldsymbol{M}_c$ 和 $\boldsymbol{M}_{su}$ 的元素 (i= 1, 2, …, j= 1, 2, …, m).

其次, 判断功能变化所影响的构件. 当矩阵 $\boldsymbol{M}_{c\text{-}su}$ 中元素值为 $m_{c\text{-}su(i,j)} \neq 0$ 时, 则场景 s_j 变化必将引起构件 c_i 的变化. $\boldsymbol{M}_{c\text{-}su}$ 中第 j 列非 0元素个数表示场景 s_j 变化直接影响的构件数, 第 i 行非 0元素个数表示直接引起改变构件 c_i 的场景数.

例如 (续上例), 由于:

$$\boldsymbol{M}_{su\text{-}c}=\boldsymbol{M}_c \times \boldsymbol{M}_{su}$$

$$=\begin{bmatrix}1&0&0&1&0\\0&0&1&0&1\\0&1&0&1&0\end{bmatrix}\times\begin{bmatrix}2&0&1&0&1&1\\0&2&0&1&1&0\\1&1&0&0&2&0\\0&1&0&1&0&0\\0&1&0&0&1&0\end{bmatrix}$$

$$=\begin{bmatrix}2&1&1&1&1&1\\1&2&0&0&3&0\\0&3&0&2&1&0\end{bmatrix}$$

则对于矩阵 $\boldsymbol{M}_{su\text{-}c}$ 中的第 1列来说, 场景 s_1 的变化必将引起 c_1 和 c_2 的变化 (其他列类同), 对第 1行来说, 引起对象 o_2 变化的可能场景为 s_1、s_2 和 s_5 (其他行类同).

此外, 同样根据矩阵 $\boldsymbol{M}_{su}$ 中元素值来判定每条变化信息传递的途径和范围. 例如, 上例中矩阵 $\boldsymbol{M}_{su\text{-}c}$ 的第 3行第 2列的元素值为 3, 则表示变化信息自场景 s_2 传递到构件 c_3, 存在 3条路径 $s_2 \rightarrow u_1 \rightarrow o_2 \rightarrow c_3$、$s_2 \rightarrow u_3 \rightarrow o_2 \rightarrow c_3$ 和 $s_2 \rightarrow u_3 \rightarrow o_4 \rightarrow c_3$. 传递的途径越多, 变化影响的范围越大, 实施该变化付出的代价越高.

step 4　确定构件之间的变化影响

首先计算 M_{cc} 的可达矩阵 $\boldsymbol{M}_{R^+}=\bigvee_{x=1}^{l} M^{(x)}$, 然后确定变化构件所影响到的其他构件, 并对 SA 中的构件进行删除、增

加、修改、合并与分解、及其关系的重组等. 具体方法可参阅文献[11],在此不再赘述.

总之,通过以上四个步骤,可将用户功能需求变化追踪到 SA 中;进而利用可达矩阵再来圈定最终被影响的全部构件,为实施软件功能变化提供了有效支持.

5 结束语

软件变化捕捉与跟踪是实现软件演化的关键,也是软件维护的核心. 要把握软件演化,研究软件变化追踪的有效方法是首要的. 用多种关系矩阵来记录用户功能需求变化传递中的信息,用矩阵运算实现变化信息的过程追踪,并利用可达矩阵辅助确定 SA 中变化构件所波及和影响的其他构件,本文给出了具体的跟踪方法和实现过程. 对实施软件维护和研究软件演化具有一定的借鉴和指导意义.

文中阐述的变化跟踪方法虽然是基于用户功能需求变化而获得的,但其思想同样适应其他需求变化(诸如技术变化、环境变化和非功能变化等). 要注意的是:由于基于场景的用况为用户功能需求捕捉提供了良好手段,这为本文的功能需求变化信息的捕捉奠定了基础. 而对其他需求变化来说,除了要寻求另外有效的变化起源建模技术之外,完全可以参考本文的思路对其进行过程的跟踪.

后续的工作主要包括借鉴本文的思想:探求基于场景的软件质量需求变化分析和捕捉技术;获取软件变化实施难易程度量化评估方法和计算模型;针对软件维护,结合软件体系结构研制具有实用价值的软件变化跟踪工具.

致谢 非常感谢北京大学杨芙清院士提供良好的博士后研究条件.

参考文献:

[1] Liu Y, Zhang SK, Wang LF, Yang FQ. Component-based software frameworks and role extension form[J]. Journal of Software, 2003, 14(8): 1364-1370. (in Chinese)

[2] Alistair C. Writing Effective Use Cases[M]. Addison-Wesley, 2001.

[3] Geri S, Jason PW. Applying Use Cases(2nd Ed.)[M]. Addison-Wesley, 2001.

[4] Jacobson I, Booch G, Rumbaugh J. The Unified Software Development Process[M]. Addison-Wesley, 1999.

[5] Yau SS, Collofello JS, McGregor TM. Ripple effect analysis of software maintenance[A]. Proc of the Computer Software and Applications Conf(COMPSAC'78)[C]. Washington, DC USA: IEEE Computer Society Press, 1978. 60-65.

[6] Bohner SA. Impact analysis in the software change process: A year 2000 perspective[A]. Proc. of the 1996 Int. Conf. on Software Maintenance[C]. Washington, DC USA: IEEE Computer Society Press, 1996. 42-51.

[7] Bohner SA. Software change impacts an evolving perspective[A]. Proc of the Int'l Conf of Software Maintenance (ICSM'2002)[C]. Washington, DC USA: IEEE Computer Society Press, 2002. 263-272.

[8] Baxter ID, Pidgeon CW. Software change through design maintenance[A]. Proc of the Int'l Conf of Software Maintenance[C]. Washington, DC USA: IEEE Computer Society Press, 1997. 250-259.

[9] Chaumun MA, Kabaili H, Keller RK, Lustman F. A change impact model for changeability assessment in object-oriented software systems[A]. Proc of the 3rd European Conf. on Software Maintenance and Reengineering[C]. Washington, DC USA: IEEE Computer Society Press, 1999. 130-138.

[10] Zhou Xin. Research on Object-Oriented Software Change Impact Analysis(Ph. D. thesis)[D]. Beijing: Peking University, 2003. (in Chinese)

[11] 王映辉, 王立福. 软件体系结构演化模型[J]. 电子学报, 2005, 33(5): 1381-1386.
Wang Ying-hui, Wang Li-fu. Research about model and ripple effect analysis of software architecture evolution[J]. Acta Electronica Sinica, 2005, 33(8): 1381-1386. (in Chinese)

[12] Frank A, Granville M. Advanced Use Case Modeling[M]. Addison-Wesley, 2001.

[13] Booch, Grady. Object-oriented analysis and design with application, 2nd ed.[M]. Addsion-Wesley, 1994.

[14] Rumbaugh, James, Michael B, et al. Object-Oriented Modeling and Design[M]. Prentice Hall, Upper Saddle River, 1991.

作者简介:

王映辉 男,1967年出生,博士,教授. 分别于1989年7月、1999年7月和2002年7月本科、硕士和博士毕业,并获得学士、硕士和博士学位,且于2005年9月博士后出站于北京大学软件工程国家工程研究中心. 现于陕西师范大学计算机科学与技术学院从事教学和科研工作,目前主要研究方向为软件演化、计算机视觉和GIS可视化. E-mail: wyhmail@snnu.edu.cn

王立福 男,1945年生,博士,教授,博士生导师. 研究方向为软件工程、软件测试、软件体系结构和信息安全.