基于特征迹的软件演化研究

黄万艮[1,2],陈松乔[2]
HUANG Wan-gen[1,2],CHEN Song-qiao[2]

1.湖南大学 会计学院 信息管理系,长沙 410079
2.中南大学 信息科学与工程学院,长沙 410083
1.Information Department of Accounting College,Hunan University,Changsha 410079,China
2.College of Information Science and Engineering,Central South University,Changsha 410083,China
E-mail:hwgencn@hotmail.com

HUANG Wan-gen,CHEN Song-qiao.Study on software evolution based on feature traces.Computer Engineering and Applications,2007,43(20):3-6.

Abstract: Require variations are the essential cause of Software Evolution (SE),the gap between problem domain and solution domain is the difficult root of SE success.Requires and feature traces supporting requires are associated with system interfaces,SE is defined based on them.SE is classfied into function evolution,non-function evolution and eviroment evolution according to different kinds of requires,and it is discussed how to evolve softwares.According to the dependency relations among private messages,local messages and public messages,the effect of SE is analyzed quantitatively,some characters of evolvable software are gained.SE based on requires and feature traces is propitious to evolution location,evolution management and evolution verification.
Key words: message dependency matrix;feature traces;require variations;software evolution

摘 要:需求变化是导致软件演化的根本原因,而问题空间与解空之间存在的缺口是软件难以成功演化的根源。通过系统接口,把变化的需求与支持它的特征迹联系起来,定义了基于需求和特征迹的软件演化。根据需求的不同,把软件演化分为功能演化、非功能演化和环境演化,并对软件演化方法进行了讨论。根据特征迹的私有消息、局部消息和公有消息之间的关系,对演化影响进行了定量分析,得到了可演化软件的一些性质。基于特征迹的软件演化,有利于演化定位、演化管理和演化的验证.
关键词:消息依赖矩阵;特征迹;需求变化;软件演化
文章编号:1002-8331(2007)20-0003-04 **文献标识码**:A **中图分类号**:TP311

传统的软件开发,总是尽快开发出原型系统,并通过原型系统的运行得到用户的反馈,进一步明确系统的需求,改进系统的设计,这对提高软件质量、保证软件件系统的开发成功具有重要意义[13];一个软件系统开发完毕正式投入使用之后,如果需求发生变化,或者要将该系统移植到另一个环境运行(环境需求变化),就需要对软件进行修改,包括静态修改和动态修改。因此,无论是软件开发,还是软件使用,都涉及软件的变化,只有不再使用的软件才保持不变。这里把软件为满足新的需求而发生变化称为软件演化,显然,需求变化是导致软件演化的根本原因。

研究者把整个软件工程分成问题空间和解空间。用户总是关心系统功能(直接与问题空间相关),以确定需求是否得到满足;软件开发者关心软件产品的创建和维护(直接与解空间相关)。开发、分发和演化成功软件困难的根源是问题空间和解空间存在缺口[7]。首先,需求通常是以系统特征的观点给出的,定位需要改变的代码要花费大量的时间;其次,如果不清楚系统特征之间的相互作用,改变可能导致有害的副作用[11]。虽然Albert[10]等已指出构件之间的关系及其消息的依赖关系,对于改变时定位是关键的,但是存在的形式化消息不能很好地支持软件演化[12]。

为了满足新的用户需求,不断改变软件,使软件越来越复杂,从而导致软件质量持续下降,最终导致软件老化。由于经济和技术两个方面的原因,变化导致隐含假定的违反,验证技术也必须对整个软件进行验证,即使只有很小的变化,这导致费用正比于整个软件系统。为了使软件易于扩展,开发者倾向于使用提供功能多的构件[16],使得软件实际上只使用构件提供的众多功能中的少数几个, 而依赖的却是整个构件所依赖的功能,这不仅使基于构件的软件变得复杂,也使软件演化更困难。

软件演化不仅是软件生命周期中的一个完整的组成部分,而且开发可演化软件已经成为在IT界具有最高优先级的问题。研究软件演化的目的,是为了能够预见软件的演化,开发可演化的软件[1],方便演化的定位、形式化说明、证明、确认和管理,减少软件演化费用。为了达到演化研究目的,本文给出了消息依赖矩阵和特征迹。通过系统接口,把需求与支持需求的特征迹联系起来,对演化影响进行了定量分析,并讨论了私有消息、局部消息和公有消息之间的依赖关系,得到了可演化软件

基金项目:国家自然科学基金(the National Natural Science Foundation of China under Grant No.90104028)。
作者简介:黄万艮,博士生,主要研究方向:软件工程,信息安全;陈松乔,教授、博士生导师,主要研究方向:软件工程等。

的一些性质。

本文组织如下:第1章介绍了相关概念,第2章讨论了消息依赖矩阵;第3章对软件演化进行了探讨,第4章对全文进行了总结。

1 相关概念

1.1 消息与构件

定义1 一个消息可以定义为一个四元组,即 $message=<source,destination,type,data>$。其中:*source* 表示发送消息的构件;*destination* 表示接收消息的构件;*type* 表示消息的类型;*data* 表示构件之间交换的数据,包括服务名称和参数等,具体内容与消息类型有关。

虽然构件还没有统一的定义,但为了讨论方便,根据赵会群[15]和张世琨[17],给出如下定义:

定义2 构件是一个数据单元或一个计算单元,由构件接口集合和构件实现模块组成。一个接口可以定义为一个六元组,即 $Interface=<RMsg,SMsg,Beha,Cons,NFunc,Ev>$。其中:*RMsg* 表示构件接口接收的消息集合;*SMsg* 表示构件接口发送的消息集合;*Beha* 描述构件接口行为语义;*Cons* 说明对构件行为的约束;*NFunc* 说明构件的非功能特性;*Ev* 定义构件对环境的依赖。

1.2 特征与特征迹

特征是系统的一个已经实现的功能需求,是一个由用户触发的系统的可观察行为[11]。特征迹是由于特征被触发后引发的一个执行轨迹,由一些方法调用构成[11]。用 *FTr* 表示系统的一条特征迹,用 *s.FTrs* 表示系统 *s* 的所有特征迹。在基于消息的系统中,每一条特征迹由系统接收的一条消息激发,并且每一条特征迹可以表示成为一个消息序列。

1.3 消息依赖关系

定义3 $\forall rm\in c.RMsg$,如果构件 *c* 接收消息 *rm* 后激发功能 *rm.f* 的执行,而 *rm.f* 在执行过程中产生消息 $sm\in c.SMsg$,则称 *rm* 直接依赖于 *sm*,记为 $rm\rightarrow sm$。

显然,*rm* 直接依赖的消息可能有多条,用 $c.rm.SMsg::=\{msg|rm\rightarrow msg\}$表示 *rm* 直接依赖的消息集合。设 *c.rm.SMsg* 包含如下消息:$msg_1,msg_2,\cdots,msg_m$,其中下标表示依赖次序,即 *rm* 发出 msg_i 后,才发送 $msg_{i+1},1\leqslant i\leqslant m-1$。用 $order(rm\rightarrow msg_k)::=\{k\}$表示 msg_k 是 *rm* 发出的第 *k* 条消息。当 *rm* 多次发出同一消息 *msgj* 时,$order(rm\rightarrow msg_j)$是一个包含多个整数的集合。例如,如果 *rm* 发出的第1条和第5条是同一消息 msg_j,则 $order(rm\rightarrow msg_j)::=\{1,5\}$。为了讨论方便,如果 *rm* 循环发送同一消息 msg_j,则只在 $order(rm\rightarrow msg_j)$中记录最早的一次。

定义4 设 *R* 为 *c.RMsg* 到 *c.SMsg* 的一个二元关系,且:

(1)$R\subseteq c.RecMsg\times c.SMsg$;

(2)$<rm,sm>\in R$,当且仅当 $rm\rightarrow sm$。

如果上述条件被满足,则称 *R* 为构件 *c* 的一个消息直接依赖关系。

定义5 设 *R* 为构件 *c* 的一个消息直接依赖关系,矩阵 $DM_R(c)=[r_{ij}]$为原子构件 *c* 的消息直接依赖矩阵,其中 r_{ij} 表示构件 *c* 接收的消息 rm_i 与其发送的消息 sm_j 之间的直接依赖次序集合,并且:

$$r_{ij}=\begin{cases}order(rm_i\rightarrow sm_j),\text{如果}<rm_i,sm_j>\in R\\ \varnothing,\text{否则}\end{cases}$$

2 软件演化

2.1 需求与特征迹的关系模型

定义6 系统 *s* 的需求 *Req* 可以定义成为一个三元组,即 $s.Req=<s.FReq,s.NFReq,s.EvReq>$,其中 *s.FReq*、*s.NFReq* 和 *s.EvReq* 分别表示系统 *s* 的功能需求、非功能需求和对环境的需求。

功能需求可以使用特征来描述,特征通过系统接口来触发,而被触发的特征对应一条特征迹。为了明确需求变化和系统改变之间的关系,通过系统接口把功能需求与特征迹联系起来,而特征迹可以把一些消息及其所属构件联系起来,这有助于软件维护者理解代码改变和扩展的原因[2]。每一条特征迹都必须满足非功能需求和环境需求。图1利用UML中的记号描述了需求、系统接口、特征迹、消息和构件之间的关系。

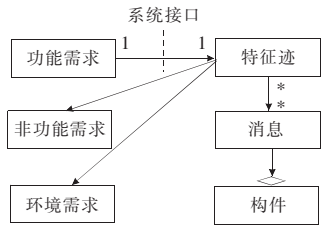

图1 需求与特征迹的关系模型

2.2 软件演化

如果系统 *s* 的特征迹 *FTrs* 满足其需求 *Req*,记为 $s.FTrs\vdash s.Req$。根据软件演化框架[14],利用特征迹,下面给出软件演化定义。

定义7 设 s_1 和 s_2 两个系统,$s_1.Cons$ 表示系统 s_1 的约束,$\triangle Req$ 表示增加的需求,$s_1.FTrs\vdash s_1.Req\wedge s_2.FTrs\vdash s_2.\text{Req}$,且:

(1)$s_2.Req=s_1.Req+\triangle Req$;

(2)$s_2.FTrs=s_1.FTrs+\triangle FTrs$;

(3)$s_2.NFReq\Rightarrow s_1.NFReq$;

(4)$s_2.Cons\Rightarrow s_1.\text{Cons}$。

如果上述条件被满足,则称 s_2 是 s_1 的一个演化,记为 $s_1\propto s_2$。

软件演化不仅意味着系统功能得到增强和非功能特性得到改善,而且应具备“追溯性”、“一致性”和“完整性”[5]。“追溯性”不仅有利于软件演化的管理,而且有利于软件演化定位。$s_2.FReq-s_1.FReq$ 为软件演化后增加的功能需求,$\forall r\in s_2.FReq-s_1.FReq,\exists!FTr_2\in s_2.FTrs\cdot s_2.FTr_2\vdash r,FTr_2$ 必定与变化密切相关。“一致性”和“完整性”是演化的必要条件。“一致性”表现为 s_1 的功能在 s_2 中保持不变,并且 $\triangle FTrs$ 不违反 $s_1.Cons$。完整性表现为演化前后系统约束条件和非功能属性没有被破坏。

s.FReq 可以作为用户对系统功能的要求,如果 $s.FTrs\vdash s.FReq$,则 *s.FReq* 和 *s.FTrs* 是一致的,而 *s.FTrs* 只描述 *s.Beha* 的一部分;*s.NFReq* 可以作为用户对系统非功能特性的要求,而 *s.Nfunc* 是系统已经具备的非功能特性,如果 $s.FTrs\vdash s.NFReq$,则 *s.NFReq* 和 *s.Nfunc* 是一致的;*s.EvReq* 可以作为用户对系统环境的要求,而 *s.Ev* 是系统已经适应的环境,如果 $s.FTrs\vdash s.EvReq$,则 *s.EvReq* 和 *s.Ev* 是一致的。

2.3 演化分类

根据需求类型的不同,把演化分为功能演化、非功能演化和环境演化。

定义8 设 s_1 和 s_2 两个系统,$s_1.FTrs\vdash s_1.Req\wedge s_2.FTrs\vdash$

$s_2.Req$,如果:

(1)若 $s_2.FReq=s_1.FReq+\triangle FReq \wedge s_2.FTrs=s_1.FTrs+\triangle FTrs \wedge \triangle FTrs \vdash s_1.Cons \wedge s_1.NFReq=s_2.NFReq \wedge s_1.EvReq=s_2.$ $EvReq$,则称 s_2 为 s_1 的功能演化,记为 $s_1 \propto s_2(F)$。

(2)若 $s_1.FReq=s_2.FReq \wedge s_1.EvReq=s_2.EvReq \wedge s_2.NFReq=s_1.NFReq+\triangle NFReq$,则称 s_2 为 s_1 的非功能演化,记为 $s_1 \propto s_2(NF)$。

(3)若 $s_1.FReq=s_2.FReq \wedge s_1.NFReq=s_2.NFReq \wedge s_2.EvReq=s_1.EvReq+\triangle EvReq$,则称 s_2 为 s_1 的功能演化,记为 $s_1 \propto s_2(Ev)$。

(4)若如果 $s_1 \propto s_2(F)$,$s_2 \propto s_3(NF)$,$s_3 \propto s_4(Ev)$,则称 s_4 是 s_1 的一个演化,记为 $s_1 \propto s_4$。

2.4 软件演化方法

根据软件工程原理,进行软件开发时,首先根据功能需求确定软件体系结构,即确定构件之间的依赖关系与约束,然后根据非功能需求,或者对软件体系结构进行调整,或者把相关构件替换成具有更好非功能特性的构件,或者改变消息传播的非功能特性。

进行演化时,首先根据功能需求确定特征迹,根据特征迹确定演化位置,实施功能演化;然后进行非功能演化和环境演化,并根据特征迹对演化进行验证。如果只有功能演化(可能导致系统的非功能特性发生变化),所以需要验证 $s_2.FTrs=s_1.FTrs+\triangle FTrs \wedge \triangle FTrs \vdash s_1.Cons \wedge \triangle FTrs \vdash s_1.EvReq \wedge s_2.FTrs \vdash s_1.NFReq$。进行非功能演化时(一些非功能特性可能存在冲突,即一种非功能特性的改善,可能导致系统不满足另外一种非功能特性),功能需求和环境需求没有发生演化,所以只需要验证 $s_2.FTrs \vdash s_2.$ $NFReq$ 是否成立;进行环境能演化时,功能需求和非功能需求没有发生演化,所以只需要验证 $s_2.FTrs \vdash \triangle EvReq$ 是否成立(必要时需要考虑 $\triangle EvReq$ 是否满足非功能需求)。

2.5 导致软件演化的运算

需求的变化是导致软件变化的根本原因,需求的增加、删除和替换都会导致软件的演化。

定义 9 设 s_1 和 s_2 是两个软件系统,且 $s_1 \vdash s_1.Req \wedge s_2 \vdash s_2.Req$:

(1)插入运算,$r_2 \notin s_1.FReq \wedge s_2.FReq=s_1.FReq \cup \{r_2\} \wedge s_1 \propto s_2(F)$,为记 $s_1(ins,r_2) \propto s_2(F)$。

(2)删除运算,$r_2 \in s_2.FReq \wedge s_1.FReq=s_2.FReq-\{r_2\} \wedge s_2 \propto s_1(F)$,为记 $s_2(del,r_2) \propto s_1(F)$。

(3)替换运算,$r_1 \in s_1.FReq \wedge r_2 \in s_2.FReq \wedge s_2.FReq=s_1.FReq \cup \{r_2\}-\{r_1\} \wedge s_1 \propto s_2(F)$,为记 $s_1(r_1,rep,r_2) \propto s_2(F)$。

每一种运算可能导致一个或多个构件的改变,包括一些构件的插入、删除或修改。基于需求的运算是对特征迹进行操作,把需求和特征迹联系起来了,有利于对演化的理解、定位和管理。如果 $s_1(ins,r_2) \propto s_2(F)$,则 $(\exists FTr_2 \in s_2.FTrs \cdot FTr_2 \vdash r_2) \wedge (\forall FTr_1 \in s_1.FTrs \cdot (FTr_1 \neq FTr_2))$;如果 $s_2\ (del,r_2) \propto s_1\ (F)$,则 $(\exists FTr_1 \in s_1.FTrs \cdot FTr_1 \vdash r_2) \wedge (\forall FTr_2 \in s_2.FTrs \cdot (FTr_1 \neq FTr_2))$;如果 $s_1(r_1,rep,r_2) \propto s_2(F)$,则 $s_1(del,r_2) \propto s_3(F) \wedge s_3(ins,r2) \propto s_2(F)$。

使用类似定义 9 的方法,可以定义系统的非功能需求运算和环境需求运算。在插入或修改一个非功能需求或一个环境需求,要求所有特征迹都不仅满足未改变的需求,而且满足这些变化了的需求。

3 演化影响分析

3.1 基于可达矩阵的演化影响分析

从宏观上看,即从软件体系结构上看,在图 2(a)中,c_4 和 c_6 的变化对 c_1 和 c_2 都会产生影响;但是从微观上看,即考虑构件内部的一些连接方式,在图 2(b)中,c_4 的变化对 c_2 不产生影响,c_6 的变化对 c_1 不产生影响。所以,如果用基于软件体系结构的可达矩阵来进行演化影响分析,结果可能是不准确的,可能导致对构件影响的过高估计[6]。进一步研究发现,利用可达矩阵和对 SA 的贡献[6]进行分析时,由于只有 c_1 和 c_2 依赖于 c_3,所以 c_3 的贡献为 2,但是由于 c_3 所处位置关键,从使用者和开发者的角度,c_3 的变化将对 $c_1 \rightarrow c_3 \rightarrow c_4$、$c_1 \rightarrow c_3 \rightarrow c_5$、$c_2 \rightarrow c_3 \rightarrow c_5$ 和 $c_2 \rightarrow c_3 \rightarrow c_6$ 四条路径产生影响。显然 c_3 的影响被低估了。

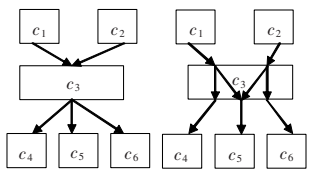

图 2 构件关系示例

3.2 基于特征迹的演化影响分析

在本文中,假定每一条消息与一个方法对应,对消息的改变意味着对方法的改变。用 $FTR(msg)$ 表示依赖于消息 msg 的特征迹集合,用 $FTR(c)$ 表示经过构件 c 的特征迹集合,则:

(1)改变消息 msg 的影响为 $|FTR(msg)|$。

(2)改变构件 c 的影响为 $|FTR(c)|$。

(3)改变消息 msg_1 和 msg_2 的影响为 $|FTR(msg_1) \cup FTR(msg_2)|$。

(4)改变构件 c_1 和 c_2 的影响为 $|FTR(c_1) \cup FTR(c_2)|$。

(5)改变构件 c_1 和消息 msg_1 的影响为 $|FTR(c_1) \cup FTR(msg_1)|$。

定义 10 $\forall msg \in s_1.Msg, FTr \in s.FTrs$,如果:

(1)$|FTR(msg)|=1$,则称 msg 为 FTr 的私有消息。

(2)$1<|FTR(msg)|<|s.FTrs|$,则称 msg 为依赖于它的一组特征迹的局部消息。

(3)$|FTR(msg)|=|s.FTrs|$,则称 msg 为公有消息。

使用类似的消息可以定义一条特征迹的私有构件、一组特征迹的共有构件和系统的公有构件。

定理 1 在消息依赖矩阵中,消息 msg 所在列的每一个非空元素所在的行对应一条特征迹,$|FTR(msg)|$ 等于 msg 所在列对应的非空元素个数。

定理 2 $\forall r \in s.FReq, \exists FTr \in s.FTrs \cdot FTr \vdash r$,如果 r 的变化导致 FTr 上的消息 $msg_1, msg_2, \cdots, msg_m$ 被改变,则受到影响的特征迹条数为 $|FTR(msg_1) \cup FTR(msg_2) \cup \cdots \cup FTR(msg_m)|$。

改变一条特征迹私有的消息或构件,其影响最小;改变一组特征迹共有的消息或构件,其影响适中;改变系统公有的消息或构件,其影响最大。改变影响的大小,虽然不能说明改变本身工作量的大小,但基本能够说明测试工作量的大小,因为每一条受到影响的特征迹对应的需求是否得到满足必须进行测试。另外,使用定理 2 可以计算和比较支持不同需求的特征迹被改变后受到影响的需求数。

定理 3 $\forall rm \in s.Msg | msg \in rm.SMsg \cdot FTR(msg) \supseteq FTR(rm)$。

推论 1 用 $priMsg$、$LocMsg$ 和 $pulMsg$ 分别表示系统全部的私有消息、局部消息和公有消息,有:

(1)$\forall rm \in pulMsg \cdot rm.SMsg \cap (LocMsg \cup priMsg)=\emptyset$,即公有消息不依赖于局部消息和私有消息。

(2) $\forall rm \in LocMsg \cdot rm.SMsg \cap priMsg=\emptyset$,即局部消息不依赖于私有消息。

推论 2 $\forall rm \in s.Msg, msg \in rm.SMsg \cdot |FTR(rm)| \leqslant |FTR(msg)|$。

为了开发可演化的软件,首先把需求分解为基本需求和可变需求,并使用不同的特征迹支持不同的需求;然后把特征迹上的消息分为“私有消息”、“局部消息”和”公有消息”,这些消息应满足定理 3 和推论 1,并应尽可能把预期到的变化纳入到“私有消息”和只在少数几条特征迹上的“局部消息”之中。

另外,变化的消息集中在一个构件中比分布在多个构件中,更容易演化。

3.3 简单实例

假定登录系统需求包括:匿名登录和用户名登录。系统由登录构件、命令行处理构件、认证构件和数据库处理构件组成,其定义简单描述如下:

```
component Log is
RMsg:bool anonymityLog();
     bool userNameLog();
SMsg:string getCommandLine();
     bool anonymityAuth(string line);
     bool userNameAuth(string line);
end component;
component Authentication is
RMsg:bool anonymityAuth(string line);
     bool userNameAuth(string line);
SMsg:int commandLength(string line);\\ 返回命令行参数个数
     string commandLower(string line);\\ 把命令行转换为小写
     int stringCompare(string source,string destination);
     string getPasswordList(string Name);
end component;
component CommandLineprocess is
RMsg:string getCommandLine();
     int commandLength(string line);
     string commandLower(string line);
     int stringComFTrare(string source,string destination);
end component;
component DataProcess is
RMsg:string getPasswordList(string Name);
end component;
```

在上述构件中,消息之间的依赖关系如下:

order(anonymityLog→getCommandLine)={1};
order(anonymityLog→anonymityAuth)={2};
order(userNameLog→getCommandLine)={1};
order(userNameLog→userNameAuth)={2};
order(anonymityAuth→commandLength)={1};
order(anonymityAuth→commandLower)={2};
order(anonymityAuth→stringCompare)={3};
order(userNameAuth→commandLength)={1};
order(userNameAuth→commandLower)={2};
order(userNameAuth→getPasswordList)={3};
order(userNameAuth→stringComFTrare)={4}。

使用消息依赖矩阵描述消息的依赖及其依赖次序更简洁。对于 $DM_R(Log)$,假定矩阵的行依次对应:anonymityLog,userNameLog 矩阵的列依次对应:getCommandLine,anonymityAuth,userNameAuth,于是有:

$$\boldsymbol{DM_R}(Log)=\begin{pmatrix}\{1\} & \{2\} & \emptyset \\ \{1\} & \emptyset & \{2\}\end{pmatrix}$$

对于 $\boldsymbol{DM_R}(Authentication)$,假定矩阵的行为 anonymityAuth,userNameAuth;矩阵的列依次对应:commandLength,commandLower,getPasswordList,stringCompare。于是有:

$$\boldsymbol{DM_R}(Authentication)=\begin{pmatrix}\{1\} & \{2\} & \emptyset & \{3\} \\ \{1\} & \{2\} & \{3\} & \{4\}\end{pmatrix}$$

对于登录系统的消息依赖矩阵 $\boldsymbol{DM_R}(LogSys)$,假定矩阵的行依次对应:anonymityLog,userNameLog 矩阵的列依次对应:getCommandLine,anonymityAuth,userNameAuth,commandLength,commandLower,getPasswordList,stringCompare,于是有:

$$DMR(LogSys)=\begin{pmatrix}\{1\} & \{2\} & \emptyset & \{3\} & \{4\} & \emptyset & \{5\} \\ \{1\} & \emptyset & \{2\} & \{3\} & \{4\} & \{5\} & \{6\}\end{pmatrix}$$

根据系统需求,从依赖矩阵中可以求出系统的两条特征迹:anonymityLog→getCommandLine→anonymityAuth→commandLength→commandLower→stringCompare 和 userNameLog→getCommandLine →userNameAuth →commandLength →commandLower→getPasswordList→stringCompare。

显然,每一条特征迹支持一个用户需求,而且 anonymityLog,userNameLog、anonymityAuth、userNameAuth 和 getPasswordList 是这两条特征迹所特有的消息,它们的影响为 1,getCommandLine、commandLength、stringCompare 和 commandLower 是这两条特征迹所公有的消息,它们的影响为 2。由于每一个需求有一条特征迹支持,并且特征迹上的消息满足定理 3,所以本系统具有较好的演化性。如果要删除用户需求“用户名登录”,则删除消息 userNameLog、userNameAuth 和 getFTRassList 即可,由于这 3 条消息是“用户名登录”所特有,所以删除不会影响系统对其他需求的支持。如果要插入一个需求,则需要借助于 UML 确定用例。

4 总结

为了说明每一条接收消息所依赖的消息,简化软件实际的消息依赖,定义了构件的消息依赖矩阵。为了描述构件的行为,定义了特征迹;为了方便演化管理,定义了基于需求和特征迹的软件演化,并根据需求的不同把演化分为功能演化、非功能演化和环境演化。无论是对软件演化影响进行分析,还是进行演化定位、演化管理和演化验证,可以首先根据功能演化确定特征迹,然后根据特征迹处理非功能演化和环境演化。根据消息与特征迹的关系,把消息分成私有消息、局部消息和公有消息,并讨论了它们之间的依赖关系,得到可演化软件应具有的一些性质。(收稿日期:2007 年 1 月)

参考文献:

[1] Cook S,He J,Harrison R.Dynamic and static views of software evolution[C]//Proc of the Int'1 Conf of Software Maintenance,IEEE,2001:592-601.

[2] Orla Greevy,Stephane Ducasse,Tudor Girba.Analyzing feature traces to incorporate the semantics of change in software evolution analysis[C]//Proceedings of ICSM 2005(21th International Conference on Software Maintenance),2005:347-356.

[3] Harn M,Berzins V,Luqi Shultes B.A formal model for software evolution[C]//Proc of the Int'1 Conf of Computational Intelligence and Multimedia Applications,1999,9:143-147.

(下转 10 页)