基于构件的数据流软件可靠性模型

徐钦桂[1,2] 刘桂雄[2]

(东莞理工学院计算机学院 东莞 523808)[1] (华南理工大学机械与汽车工程学院 广州 510640)[2]

摘 要 基于构件的数据流软件由输入数据激活的构件确定程序执行路径,其可靠性受输入数据分布特性的影响,难以采用基于状态或基于路径等传统模型进行评测。提出一个结合构件执行频度和操作剖面的可靠性模型,其从分析数据流程序结构入手,通过定义组合节点,将程序表示成多级层次结构的形式。根据构件间数据流和控制流关系,确定实际激活的构件,计算其执行频度,并将操作剖面沿着数据流向本层和下层构件传递。利用基于深度优先的递归算法思想,按照相反顺序,逐层估算各级组合节点的可靠性,最后获得整个软件的实际可靠性。应用实例表明,模型能有效地估算基于构件数据流软件的实际可靠性,反映输入接口有效数据就绪状态及分布特性。

关键词 软件可靠性,体系结构可靠性模型,数据流软件,操作剖面

中图法分类号 TP311.5 **文献标识码** A

Software Reliability Model for Component-based Dataflow Software

XU Qin-gui[1,2] LIU Gui-xiong[2]

(School of Computer, Dongguan University of Technology, Dongguan 523808, China)[1]

(School of Mechanical and Automotive Engineering, South China University of Technology, Guangzhou 510640, China)[2]

Abstract Due to the mechanism that component-based dataflow software has its execution path determined by components activated by data fed into input interfaces, its software reliability can be influenced by distribution of input data, and is thus difficult to evaluate using traditional models such as state-based or path-wise ones. A reliability model combining frequency statistics and operational profile was put forward in this paper. Starting from analysis of dataflow software structure, programs were denoted as multi-layer structure by introduction of composite node. On the basis of data and control flow relations, activated components and their execution probability were estimated. The operational profile was transferred along the direction of data flow to components on the same or a lower layer. By means of depth-first rule, reliability of composite nodes on all levels was evaluated in opposite order until the actual reliability of the whole software was obtained. A case study shows that the model can effectively evaluate actual reliability of component-based data flow software, reflecting ready state of effective data in input interfaces and data distribution attributes.

Keywords Software reliability, Architecture-based reliability model, Dataflow software, Operational profile

1 引言

软件可靠性是软件系统在规定的时间及环境条件下完成规定功能的能力,它作为衡量软件质量的关键指标之一,成为软件工程学科的重要研究领域。

早期采用可靠性增长模型建模(Software Reliability Growth Model, SRGM),将软件看成一个单一的整体,通过"黑盒"测试方法获得缺陷数量、等级和故障间隔等数据,来估计模型参数或者校准模型[1,2]。随着软件规模的扩大及复杂性的增加,"黑盒"测试方法呈现出测试用例集膨胀而覆盖性变差的问题,因而大型构件化软件倾向于采用面向体系结构的可靠性模型。按照对软件结构和运行模式的认识角度不同,有基于状态、基于路径和基于操作剖面3种可靠性模型。基于状态的模型使用软件的控制流图描述软件体系结构,假设构件间的控制转移具有马尔可夫属性,以转移概率为权值,综合各构件的可靠性获得整个软件的可靠性[3-5]。基于路径的模型通过分析找出所有执行路径,将路径上各构件可靠性相乘,计算每条路径可靠性,以路径执行频率为权值,综合各路径可靠性,求得软件整体的可靠性[6,7]。基于操作剖面的模型认为软件可靠性与软件使用环境有关,构件开发者度量构件在不同输入子域下的可靠性,软件实际运行时以操作剖面为权值综合各输入子域的可靠性得到构件可靠性,沿执行路径传递操作剖面,计算沿途各构件可靠性,与基于状态或基于路径的方法相结合综合所有构件可靠性[8,9]。结构化模型能反映软件结构、执行路径和输入数据分布对软件可靠性的影响,但其假定软件按照程序自然顺序或控制结构定义的流程逐条执行,适合基于控制流和基于事件驱动软件的可靠性分析。在测量控制和数字信号处理等领域,经常采用数据流

到稿日期:2010-08-06 返修日期:2010-11-06 本文受国家自然科学基金(50775077),广东省科技计划项目(2007B010200046)资助。

徐钦桂(1967-),男,副教授,主要研究方向为计算机系统结构、计算机性能评测,E-mail:xqg@dgut.edu.cn;刘桂雄(1968-),博士,教授,主要研究方向为智能化检测、仪器仪表技术。

驱动的软件结构。如 Labview 虚拟仪器[10]的源程序由数据流控制构件执行的时机及先后次序，一旦某构件所需输入数据全部就绪，即可启动构件执行。这种程序模型可以利用多线程机制挖掘构件间的并发性，提高程序的执行效率[11,12]，但却打破了基于状态模型中构件的执行受单一前驱构件控制和基于路径模型中软件具有单一执行线索的假定[3-9]，因而需要探索基于构件多数据流并发软件的可靠性评估方法。

针对上述问题，本文提出一种基于构件数据流软件可靠性评估模型，将程序看成由数据流驱动而关联起来的构件集合，并引入选择结构和循环结构，增强程序的控制能力。该模型通过基于子域的可靠性、操作剖面及剖面传递计算各构件的实际可靠性，根据数据流程序结构推导基本结构的可靠性计算公式，进而提出估算整个数据流程序的递归算法，并通过实例验证了模型的有效性。

2 基于构件数据流软件体系结构

基于构件数据流软件通常采用数据流图的源程序形式，其节点分为构件节点和连接件节点两种。构件节点分为应用功能节点和流程控制节点。前者包括过程、函数、控件、组件和代码段等，实现应用逻辑功能；后者是为增强程序流程控制功能和提高编程灵活性而从传统程序设计语言中引入的顺序、选择和循环等控制结构，又可称为容器节点。在 Labview 等数据流图源程序中，受其控制的其它节点被置于这些节点内部。连接节点是将构件节点连接起来的胶合逻辑。为方便程序分析，还可将流程控制节点及其内部节点合起来称为组合节点。组合节点也可包含其它组合节点，从而形成多层的嵌套结构。一个基于构件数据流程序的示例如图 1 所示，其嵌套关系为：数据流程序 $P=\{C_1,C_2,C_3,C_4,C_5,B_1,L_1,L_2,L_3,L_4,L_6\}$，循环控制构件 C_9 与其内部节点构成组合节点 $B_1=\{C_9,C_6,B_2,L_5\}$，选择控制构件 C_{10} 与其内部节点构成组合节点 $B_2=\{C_{10},B_3,B_4\}$，C_{10} 的两个选择分支也可为组合节点 $B_3=\{C_7\}$，$B_4=\{C_8\}$。

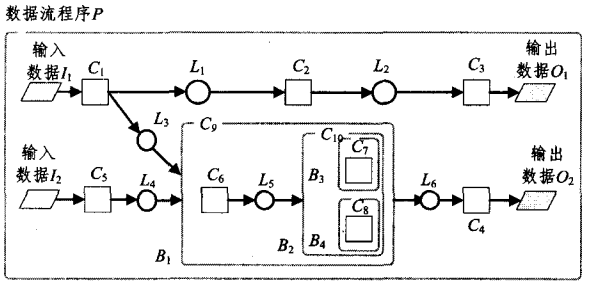

图 1 数据流程序的流程控制嵌套结构

对以上基于构件数据流程序的结构、行为和属性进行高级抽象，可得到其软件体系结构的概念，它由构成元素的描述、元素间相互作用、元素集成模式以及模式约束组成，并表示成：

$$SA=\{components,connectors,constraints\}$$

式中，*components* 为构件的集合，是软件功能的承载部件，包括控件、组件和程序模块等；*connectors* 是连接件的集合，用于实现构件间合作与交互行为，包括信息交换、数据转换、数据缓冲和过程调用等；*constraints* 是限制条件集合，用于限制软件体系结构元素的选择、组合方式和系统拓扑结构。

3 可靠性评估模型

软件可靠性与其采用的体系结构密切相关，基于构件数据流程序的可靠性 R 是构件可靠性 RC、连接件可靠性 RL、构件间集成模式 $CONN$ 以及软件操作剖面 SOP 的函数，可表示成：

$$R=f(RC,RL,CONN,SOP) \tag{1}$$

为计算 R，需要确定式(1)中的 4 个参数及函数关系 f。$CONN$ 可由软件结构直接得到，SOP 根据输入数据分布求得，RL 可通过理论计算或从已有测试数据查得，因此问题的关键是获得各构件可靠性 RC 及可靠性计算函数 f。

3.1 基于操作剖面的构件可靠性

构件可靠性的准确度量应采用白盒测试策略根据代码结构进行覆盖性测试，但构件使用者通常只能获得二进制形式的构件，故构件可靠性应由构件开发者给出。考虑到构件在不同输入域环境下运行可表现出不同的可靠性，如果将开发环境下测得的构件可靠性直接用于计算实际运行环境中的软件可靠性，则可能导致结果准确度和可信度降低。为此，本文利用操作剖面估算构件的实际可靠性。

不妨设构件 C 的 k 个输入为 $x_1,x_2,\cdots,x_k$，其数据处理功能用变换函数 $y=f(x_1,x_2,\cdots,x_k)$ 表示，y 为构件的输出结果。基于操作剖面可靠性估算要求构件开发者按子域给出构件 C 的可靠性，将开发者划分的子域集合记为 $Domain(C)=\{V_1,V_2,\cdots,V_l\}$，对各子域 $V_i\in Domain(C)$ 的可靠性记为 $R(C,V_i)$，则按子域的可靠性可表示成：

$$R_D(C)=\{(V_i,r_i)\mid 1\leqslant i\leqslant l\wedge r_i\in[0,1]\} \tag{2}$$

式中，$R(C,V_i)=r_i$，且满足：

$$Cond_1:[\forall V,V'\in Domain(C)[V\neq V'\rightarrow V\cap V=\varnothing]\wedge \text{dom } f\subseteq \bigcup_{(V,r)\in R_D(C)} V] \tag{3}$$

即 $R_D(C)$ 的子域应覆盖函数 f 定义域中的所有输入元组，且不同子域间互不重叠。

记构件 C 实际运行时中的输入剖面为：

$$Q_{in}(C)=\{(U_i,p_i)\mid 1\leqslant i\leqslant s\wedge p_i\in[0,1]\} \tag{4}$$

则各输入子域 U_i 及其实际概率 p_i 应满足：

$$\forall i\neq j[U_i\cap U_j=\varnothing]\wedge\sum_{i=1}^{s}p_i=1 \tag{5}$$

为计算构件 C 在操作剖面 $Q_{in}(C)$ 下的实际可靠性，可将 $Q_{in}(C)$ 中的输入剖面映射成 $R_D(C)$ 中各子域的发生概率，或将 $R_D(C)$ 中各子域可靠性转换成 $Q_{in}(C)$ 中各子域可靠性。为简便起见，采用前一种方法：假设各实际输入子域中的值落在 $Q_{in}(C)$ 各子域的概率呈均匀分布，计算 $R_D(C)$ 各子域 V 的概率：

$$\forall V\in Domain(C):P(C,V)=\sum_{(U,p)\in Q_{in}(C)}\left(p\cdot\frac{|V\cap U|}{|U|}\right) \tag{6}$$

这样，可将构件 C 的操作剖面和按子域可靠性统一表示成：

$$QR_D(C)=\{(V,p',r)\mid (V,r)\in R_D(C)\wedge p'=\sum_{(U,p)\in Q_{in}(C)}\left(p\cdot\frac{|V\cap U|}{|U|}\right)\} \tag{7}$$

由式(3)可知应满足条件：

$$Cond_2:[\forall (V,p,r),(V',p',r')\in QR_D(C)[V\neq V'\rightarrow V\cap V'=\varnothing]] \tag{8}$$

由 $QR_D(C)$ 可导出构件 C 实际可靠性计算公式：

$$R(C)=\sum_{(V,p',r)\in QR_D(C)}p'\cdot r=\sum_{V\in Domain(C)}P(C,V)\cdot R(C,V) \tag{9}$$

3.2 构件操作剖面的传递

不妨设构件 $C_1,C_2,\cdots,C_k$ 的输出作为构件 C_0 的输入，

将构件 $C_i(1\leqslant i\leqslant k)$ 的数据处理功能用函数 f_i 表示，f_i 对子域 V 中的输入进行变换，获得输出构成的数据域，用 $f_i(V)$ 表示，则构件 C_i 的输出剖面 $Q_{out}(C_i)$ 和构件 C_0 第 i 个输入的输入剖面 $Q_{in}(C_i,i)$ 可通过将 $QR_D(C_i)$ 中的子域 V 用 $f(V)$ 代替并保持各元组前两个分量不变得到，但应将通过 f 映射到同一子域的元组合并，故有：

$$Q_{in}(C_0,i)=Q_{out}(C_i)=\{(L,p')\mid \exists V\in Domain(C_i)[f_i(V)=L]\wedge p'=\sum_{V\in Domain(C_i)\wedge f_i(V)=L}P(C_i,V)\} \tag{10}$$

式中，$\sum_{(L,p)\in Q_{in}(C_0,i)}p=1$，但并不保证 $Cond_1$，即 $Q_{in}(C_i,i)$ 中的子域间可能存在重叠。对 $(L,p)\in Q_{in}(C_i,i)$，记 $Q_{in}(C_0,i,L)=p$，则可将构件 $C_1,C_2,\cdots,C_k$ 的输出子域进行组合，得到构件 C_0 的一个可能存在重叠的输入剖面：

$$Q_{in}(C_0)=\{((L_1,L_2,\cdots,L_k),p)\mid \forall i(1\leqslant i\leqslant k)\ \exists!\ p_i[(L_i,p_i)=Q_{in}(C_0,i)]\wedge p=\prod_{i=1}^{k}Q_{in}(C_0,i,L_i)\} \tag{11}$$

即使如此，仍可将 C_0 代入式(7)，得到构件 C_0 的操作剖面和按子域可靠性的统一表示形式 $QR_D(C_0)$，从而将构件 $C_1,C_2,\cdots,C_k$ 的操作剖面传递到构件 C_0 的操作剖面，再根据式(9)计算出该构件的实际可靠性 $R(C_0)$。

3.3 基本结构可靠性

按构件间关系及构件触发方式，可将数据流程序看成由数据流结构、选择结构和循环结构 3 种基本结构构成的组合节点嵌套构成。利用构件可靠性计算公式，根据基本结构特征及其中构件的执行频度，可推导其可靠性计算公式。

(1)数据流结构：通过数据流关系连接起来且不包含回路的节点及连接件构成的组合节点，其实例如图 2 所示。数据流结构组合节点可包含应用功能节点、连接件节点和流程控制节点，每个流程控制节点与其包含的其他节点一起在数据流结构中作为一个组合节点对待。一个数据流结构 B 的可靠性是由有效数据输入触发的节点的可靠性之积，而被触发的节点是仅依赖有效输入数据的节点。

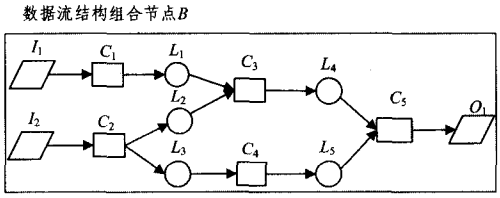

图 2 数据流结构可靠性模型

令 $I(B)=\{I_1,I_2,\cdots\}$ 为 B 中具有有效输入数据的输入接口集合，$\bar{I}(B)=\{\bar{I}_1,\bar{I}_2,\cdots\}$ 是 B 中无有效输入数据的输入接口集合，$Cs(B)=\{C_1,C_2,\cdots\}$ 为 B 的构件集合，$Ls(B)=\{L_1,L_2,\cdots\}$ 为 B 的连接件集合，$G(B)$ 为 B 的数据流图，则 $I(B)$ 可激活的构件集合为：

$C'(B)=\{C\mid C\in Cs(B)$ 且 $\forall x\in\bar{I}(B)$：$G(B)$ 不存在 x 到 C 的路径$\}$

同样，激活的连接件集合为：

$L'(B)=\{L\mid L\in Ls(B)$ 且 $\forall x\in\bar{I}(B)$：$G(B)$ 不存在 x 到 L 的路径$\}$

因此 B 的可靠性为：

$$R(B)=\prod_{C\in C'(B)}R(C)\cdot\prod_{L\in L'(B)}R(L) \tag{12}$$

(2) 选择结构：由一个流程选择构件和若干个数据流结构分支构成，将流程选择构件 C_{sl} 看成由 $C_{sl\text{-}in}$，$C_{sl\text{-}tmp}$ 和 $C_{sl\text{-}out}$ 3 个内部构件构成，将各数据流结构分支看成 n 个数据流结构组合节点 $B_1,B_2,\cdots,B_n$ 及相应的连接件 $L_1,L_2,\cdots,L_n$，则选择结构可靠性模型可用图 3 表示。当选择结构所需的输入数据全部就绪时，$C_{sl\text{-}in}$ 计算选择条件 $t_1,t_2,\cdots,t_n$，每次选择条件成立的一个分支执行，这些选择条件满足：

$$Cond_3:(t_1\vee t_2\vee\cdots\vee t_n)\wedge\forall i,j[1\leqslant i\leqslant n\wedge 1\leqslant j\leqslant n\wedge i\neq j\rightarrow\neg(t_i\wedge t_j)] \tag{13}$$

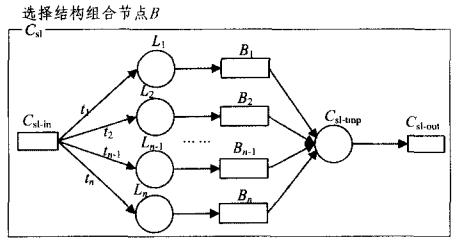

图 3 选择结构可靠性模型

各分支的执行结果通过 C_{sl} 的内部连接件 $L_{sl\text{-}tmp}$ 汇合到内部构件 $C_{SL\text{-}out}$。整个选择结构 B 的可靠性为：

$$R(B)=R(C_{sl})\cdot\prod_{i=1}^{n}[R(B_i)\cdot R(L_i)]^{\Pr(t_i=\text{true})} \tag{14}$$

式中，$\Pr(t_i=\text{true})$ 是选择执行第 i 个分支的概率，可采用固定的预估值或根据流程选择构件的输入剖面计算。

(3)循环结构：由循环控制构件 C_{loop} 和 1 个数据流组合节点 B_1 组成，循环控制构件 C_{loop} 可分解成 $C_{loop\text{-}in}$ 和 $C_{loop\text{-}end}$ 两部分，如图 4 所示。该组合节点可在循环控制构件 C_{loop} 控制下重复执行多次，每次循环迭代的输出数据可通过反馈节点 $C_{loop\text{-}end}$ 作为下一循环迭代的输入，因此该循环结构组合节点的可靠性为：

$$R(B)=R(C_{loop})\cdot[R(B_1)\cdot R(L_1)]^n \tag{15}$$

式中，n 为循环迭代次数，可根据循环构件的输入剖面估算。

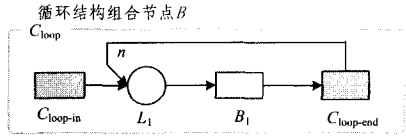

图 4 循环结构可靠性模型

3.4 数据流程序可靠性的递归计算

数据流程序中各构件和节点构成一个树形嵌套层次结构，如图 5 所示。整个数据流程序看成一个数据流结构组合节点，其中可包含功能构件节点和其它组合节点。而组合节点可以是由流程控制构件及所属节点构成的组合节点，也可以是数据流组合节点。下层组合节点的构成方式与此相同。

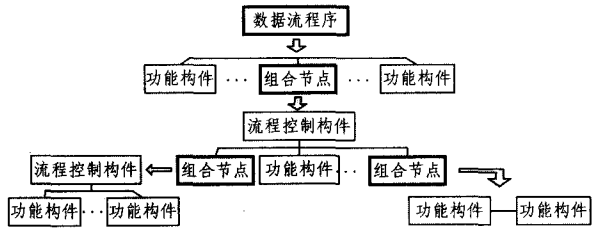

图 5 数据流程序节点树形层次结构

计算数据流程序的可靠性就是计算顶层组合节点的可靠性。鉴于这种嵌套特性，可采用以下递归算法估算数据流程

序及各级组合节点的可靠性。

算法 1 组合节点可靠性评估递归算法

ComputeDataflowReliability(B) //数据流组合节点可靠性

begin

$\forall x \in (Cs(B)-C'(B)) \cup (Ls(B)-L'(B))$:从数据流图 $G(B)$ 中删除节点 x 及关联边

$\forall x \in I(B)$:从数据流图 $G(B)$ 中删除节点 x 及其关联边

$S_1=C'(B) \cup L'(B)$

$S_2=\varnothing$

while($\exists x \in S_1 \wedge x$ 在 $G(B)$ 中无前驱)

begin

$S_1=S_1-x$

$S_2=S_2+x$

按式(11)将 $C'(B) \cup L'(B)$ 中 x 的前驱节点的操作剖面传递到 x

Switch(节点 x 的类型)

begin

Case 功能节点,连接件:

使用式(9)直接计算 x 的可靠性 $R(x)$

Case 选择组合节点:

调用 *Reliability_of_select*(x)计算 $R(x)$

Case 循环组合节点:

调用 *Reliability_of_loop*(x)计算 $R(x)$

Case 数据流组合节点:

调用 *Reliability_of_dataflow*(x)计算 $R(x)$

end

从 $G(B)$ 删去节点 x 及关联边

end

按式(12)计算节点 B 的可靠性:$R(B)=\prod_{x \in S_1} R(x)$

end

ComputeSelectReliability(B) //选择组合节点可靠性

begin

将选择结构组合节点 B 分解成图 3 中的选择结构构件 C_{sl}、数据流组合节点 $B_1, B_2, \cdots, B_n$、连接件节点 $L_1, L_2, \cdots, L_n$ 及选择条件 $t_1, t_2, \cdots, t_n$

按式(11)将 B 的操作剖面传递到所属的各节点

直接计算 C_{sl} 的可靠性 $R(C_{sl})$

$\forall x \in \{L_1, L_2, \cdots, L_n\}$:直接计算 x 的可靠性 $R(x)$

$\forall x \in \{B_1, B_2, \cdots, B_n\}$:调用 *Reliability_of_dataflow*(x)计算 $R(x)$按式(14)计算 B 的可靠性 $R(B)$

end

ComputerLoopReliability(B) //循环结构组合节点可靠性

begin

将循环结构组合节点 B 分解成图 4 中的循环结构构件 C_{loop}、数据流组合节点 B_1 及连接件节点 L_1

按式(11)将 B 的操作剖面传递到所属的各节点

直接计算 C_{loop} 的可靠性 $R(C_{loop})$

直接计算连接件 L_1 的可靠性 $R(L_1)$

调用 *Reliability_of_dataflow*(B_1)计算 $R(B_1)$

按式(15)计算 B 的可靠性 $R(B)$

end

begin

调用 *ComputeDataflowReliability*(P),计算数据流程序 P 的可靠性

end

4 实例研究

以图 1 所示的数据流程序为例验证上述可靠性模型的有效性。因连接件逻辑简单,流程控制构件通常由编程语言提供,为方便研究,可假设这两类构件的可靠性为 1,即在下面的可靠性计算中可将其忽略。

4.1 实验方案与实验结果

软件可靠性分析中需要各构件的运行可靠性,这应通过综合各构件的分子域可靠性及其操作剖面得到。前者可由构件开发者给出,后者依赖于构件的具体逻辑,超出了本文研究范围。为此,在表 1 中直接给出了图 1 中各构件按子域表示的可靠性和操作剖面。

表 1 构件可靠性数据

构件	输入子域	域大小比率	子域概率	子域可靠性
C_1	I_1	0.35	0.10	0.75
	I_2	0.65	0.90	0.95
C_2	I_1	0.30	0.10	0.85
	I_2	0.50	0.75	0.97
	I_3	0.20	0.15	0.90
C_3	I_1	0.35	0.15	0.88
	I_2	0.65	0.85	0.98
C_4	I_1	0.28	0.12	0.90
	I_2	0.72	0.88	0.98
C_5	I_1	0.24	0.19	0.90
	I_2	0.76	0.81	0.98
C_6	I_1	0.33	0.2	0.95
	I_2	0.55	0.75	0.99
	I_3	0.12	0.05	0.97
C_7	I_1	0.45	0.23	0.96
	I_2	0.55	0.77	0.99
C_8	I_1	0.22	0.1	0.95
	I_2	0.78	0.9	0.99

(1)构件可靠性计算

分别以子域大小比率和子域概率为权值计算各构件的测试可靠性和实际可靠性,结果如表 2 所列。

表 2 构件可靠性计算结果

构件	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
测试可靠性	0.88	0.92	0.95	0.96	0.96	0.97	0.97	0.98
运行可靠性	0.93	0.95	0.97	0.97	0.96	0.98	0.98	0.99

由表 2 可以看出,在构件测试期间以子域大小为权值计算的可靠性与运行期间以操作剖面为权值获得的可靠性通常会有差别,如果输入数据更多地分布在可靠性较高的子域,则运行可靠性可高于测试可靠性。

(2)软件可靠性估算

假定循环控制构件 C_9 的循环次数为 $n=10$,选择控制构件 C_{10} 的两个选择分支 B_3 和 B_4 被触发执行的概率分别为 $\Pr(t_1=\text{true})=0.7$ 和 $\Pr(t_2=\text{true})=0.3$,现按照哪些输入接口数据有效分两种情况计算软件可靠性。

(Ⅰ)$I(P)=\{I_1, I_2\}$,$\bar{I}(P)=\varnothing$

略去连接件和流程控制构件。在两个输入接口都加载有效数据的情况下,程序中所有构件都被激活执行,产生输出结果集$\{O_1, O_2\}$。根据算法 1 可推算出软件可靠性计算公式为:

$$R(P)=R(C_1)\cdot R(C_2)\cdot R(C_3)\cdot R(C_4)\cdot R(C_5)\left[R(C_6)\cdot R(C_7)^{\Pr(t_1-\text{true})}\cdot R(C_8)^{\Pr(t_1-\text{true})}\right]^n$$

$$=0.560$$

(Ⅱ) $I(P)=\{I_1\}$, $\bar{I}(P)=\{I_2\}$

仅构件集$\{C_9, C_2, C_3\}$被激活执行，产生输出结果O_1，因此软件可靠性为：

$R(P)=R(C_1)\cdot R(C_2)\cdot R(C_3)=0.857$

从计算结果可知，在不同的数据输入接口集上加载有效数据，软件执行的功能和数据流激活的构件集不同，可使软件运行时的可靠性出现较大差异。

4.2 讨论、分析

软件运行时的实际可靠性与各构件输入域划分、操作剖面传递及有效输入接口集合存在密切关系。

(1)输入子域设置的影响

输入子域数设置太少，难以反映软件的实际可靠性。当缩减到单个输入子域时，实际可靠性退化为测试可靠性。输入子域划分越多、越细，操作剖面就越能反映实际运行的输入数据分布，实际可靠性计算结果就越准确。但划分太多子域，会增加开发者构件测试工作量及操作剖面传递计算的难度，因此需要对输入子域的划分粒度进行折中。

(2)操作剖面传递精确性的影响

要有效计算软件运行时的实际可靠性，应对构件操作剖面进行准确传递。精确地计算构件的输出剖面，不但需要获得构件功能函数的单调性、连续性、极大极小值等特性，还需要实际运行构件，计算以子域边界点作为输入的输出。划分过细的输入子域，可能给输出剖面的计算带来困难。应适当控制输入剖面子域数量，并采用近似简便方法估算构件输出剖面，以利于控制问题的复杂度。

(3)有效输入接口集合的影响

输入接口上的有效数据驱动构件的执行，加载有效数据的输入接口的不同可对软件可靠性造成影响，因此准确分析构件间的数据流关系及构件执行频率，是影响模型精确性的关键因素之一。

结束语 本文将基于频率统计和操作剖面的方法相结合，提出评估基于构件数据流软件实际可靠性的模型。将基于构件的数据流结构划分为由基本结构组合节点构成的多层嵌套结构。在每个层次上，根据构件间的数据流及控制流关系，估算实际运行中被激活的构件集合，并统计其执行频率。利用基于深度遍历的递归算法思想，逐层计算各级组合节点的可靠性，能反映软件实际运行中输入接口的有效数据就绪状态及由此引起的数据流关系的动态变化。在递归算法中融入操作剖面传递过程，计算各层次构件的输入剖面，并与按子域表示的构件可靠性相结合估算构件、基本结构、组合节点和整个程序的实际可靠性，使之反映软件实际运行时输入数据的分布特性。

软件可靠性计算结果对操作剖面的反映程度取决于输入子域划分的精细度与操作剖面传递的精确性，而这些都与具体应用逻辑密切相关。细分的输入子域和精确的剖面传递要求可使操作剖面计算及传递变得复杂，因而应根据具体构件逻辑及可靠性要求进行折中。

参 考 文 献

[1] Lyu M R. Software Reliability Engineering: A Roadmap [C] // Proc of Future of Software Engineering. Washington: IEEE Computer Society, 2007: 153-170

[2] 谢景燕，安金霞，朱纪洪. 考虑不完美排错情况的 NHPP 类软件可靠性增长模型[J]. 软件学报，2010，21(5)：942-949

[3] 雷航，马成功. Markov 模型的软件可靠性测试充分性问题的研究[J]. 电子科技大学学报，2010，39(1)：101-105

[4] Wang Wen-li, Pan Dai, Chen Meihwa. Architecture-based software reliability modeling [J]. Journal of Systems and Software, 2006, 79(1): 132-146

[5] 詹涛，周兴社，符宁，等. 软件能力可信研究综述[J]. 小型微型计算机系统，2008，29(5)：786-792

[6] Katerian G P, Trivedi K S. Architecture-based approaches to software reliability prediction[J]. Computers and Mathematics with Applications, 2003, 46: 1023-1036

[7] 毛晓光，邓勇进. 基于构件软件的可靠性通用模型[J]. 软件学报，2004，15(1)：27-32

[8] 万晓民，张德平，聂长海，等. 统计测试中操作剖面的一种优化设计方法[J]. 东南大学学报：自然科学版，2008，138(12)：233-238

[9] Johnston W M, Hanna J R P, Millar R J. Advances in Dataflow Programming Languages[J]. ACM Computing Surveys, 2004, 36(1): 1-34

[10] LABVIEW. 2000. LabView User Manual[OL]. ustin, TX: National Instruments, A2000

[11] Wesley M, Hanna P J R, Richard M J. Advances in dataflow programming languages[J]. Advances in Dataflow Programming Languages, 2004, 36(1): 1-34

[12] Jurij S, Borut R, Theo U. Asynchrony in parallel computing: From dataflow to multithreading[J]. Journal of Parallel and Distributed Computing Practices, 1998, 1: 1-33

(上接第 129 页)

[2] Chen F, Li S, Yang H. Enforcing role-based access controls in software systems with an agent based service oriented approach [C] // Proceedings of 2007 IEEE International Conference on Networking, Sensing and Control. London, UK: IEEE Computer Society, 2007: 15-17

[3] Bertino E. RBAC models-concepts and trends[J]. Computers & Security, 2003, 22(6): 511-514

[4] 王治刚，王小刚，卢正鼎，等. OntoRBAC：基于本体的 RBAC 策略描述与集成[J]. 计算机科学，2007，34(2)：82-85

[5] Memon Q A. Implementing role based access in healthcare Adhoc networds[J]. Journal of Networks, 2009, 4(3): 192-199

[6] 万宏辉，朱更明. 基于 Web Service 的多 Web 应用系统访问控制集成[J]. 计算机应用与软件，2009，26(7)：28-30

[7] Li N, Mao Z. Administration in role-based access control[C] // Proceedings of 2nd ACM Symposium on Information, Computer and Communications Security. Singapore: ACM New York, 2007: 127-138

[8] Dekker M A C, Crampton J, Etalle S. RBAC administration in distributed systems[C] // Proceedings of 13th ACM Symposium on Access Control Models and Technologies. Estes Park, United States: ACM New York, 2008: 93-101

[9] 李晓峰，冯登国，徐震. 一种通用访问控制管理模型[J]. 计算机研究与发展，2007，44(6)：947-957