# 面向软件演化的可靠性分析代数方法

张 捷[1,2] 陆 阳[1] 张本宏[1] 刘广亮[1]

**摘 要** 环境和需求的改变导致软件演化发生, 并通常会使软件架构 (Software architecture, SA) 产生变化. 现有的结构化软件可靠性模型对评价软件初始结构设计有不错的效果, 但在软件演化时的实时分析方面有局限性. 从软件结构建模出发, 通过使用代数方法将软件演化描述为原子操作序列, 并跟踪分析序列中每一步操作对可靠性的影响, 从而建立基于过程的可靠性分析方法. 方法可分析演化关键环节及整体趋势, 用以进一步反馈和约束演化方案设计, 最终达到提高软件产品质量的目的. 通过对 2 个实际算例的深入分析与讨论, 方法的有效性与易用性得到验证.

**关键词** 软件可靠性, 软件演化, 软件架构, 代数方法

**引用格式** 张捷, 陆阳, 张本宏, 刘广亮. 面向软件演化的可靠性分析代数方法. 自动化学报, 2021, **47**(1): 148−160

**DOI** 10.16383/j.aas.c180143

## Reliability Analysis Algebraic Approach to Software Evolution

ZHANG Jie[1,2] LU Yang[1] ZHANG Ben-Hong[1] LIU Guang-Liang[1]

**Abstract** Because of changes in the environment and needs, software evolution often occurs and leads to changes in software architecture (SA). The existing structural software reliability models have a beneficial effect on the evaluation of the initial software architecture, but it has limitations in real-time analysis of software evolution. From the software architecture modeling, the software evolution is described as an atomic operation sequence by using the algebraic method and the reliability influence of each step in the sequence is tracked. Accordingly, a procedural reliability analysis method is established. The approach can be used to analyze the key links and the overall trend of evolution, and further feedback and constrain the evolution scheme design, ultimately to improve the quality of software products. Two practical examples are analyzed and discussed in detail, and the validity and usability of the proposed approach are verified.

**Key words** Software reliability, software evolution, software architecture (SA), algebraic method

**Citation** Zhang Jie, Lu Yang, Zhang Ben-Hong, Liu Guang-Liang. Reliability analysis algebraic approach to software evolution. *Acta Automatica Sinica*, 2021, **47**(1): 148−160

软件演化一直是软件工程领域的挑战性问题. 由于客户需求、环境变化、技术进步等原因, 软件有着更新演化的现实需要, 由此带来的开发及管理问题可能非常复杂. 针对软件演化的定量分析已经被公认是横亘软件生命周期的最复杂问题之一, 而软件架构 (Software architecture, SA) 的提出为问题的表述与解决提供了方向. 近年来, 通过使用 SA 相关方法和工具已较好解决了软件演化所带来的障碍、成本等问题, 并且涌现出一些新的观点如演化风格、演化路径等[1−3]. 但在对软件演化过程的精确描述和完整建模上, 并没有出现公认的一般性方法. 需要说明的是, 目前针对演化的研究都是过程性研究, 它基于架构工程师对软件更迭过程的完整监控, 若此工作仅依赖编码人员, 会不可避免地出现架构侵蚀或架构偏移问题[4]. 一个可行思路是对现有的 SA 工具进行推广和扩充, 使之能够适用于面向软件演化的过程性分析需要. SA 发展至今, 其工具和方法的易用性一直是尚待解决的难题, 如何准确、高效地描述演化需求和过程, 进而使得架构设计者和开发者都可以快速掌握和应用, 有很实际的意义.

另一方面, 脱离架构指导的代码演变极易导致软件设计与实现的错位, 而要修正此类问题往往代价巨大. 以目前用户最多的学习管理软件 Moodle 为例, 它在演化过程中曾经历过重大变化以及大量问题的修复[5]. 虽然拥有庞大的开发者社群和完整的开发过程记录, 但是此开源项目尚没有清晰的架构设计和演化方案描述. 每当新版本发布, 仅用文字记录下哪些新开发的组件被加入, 哪些组件被更改, 工程方法的缺失使得版本更迭脱离了 SA 设计

收稿日期 2018-03-14 录用日期 2018-12-12
Manuscript received March 14, 2018; accepted December 12, 2018
国家重点研发计划专项 (2016YFC0801804), 国家自然科学基金 (61572167) 资助
Supported by National Key Research and Development Program (2016YFC0801804), National Natural Science Foundation of China (61572167)
本文责任编委 蔡开元
Recommended by Associate Editor CAI Kai-Yuan
1. 合肥工业大学计算机与信息学院 合肥 230601 2. 安徽师范大学计算机与信息学院 芜湖 241003
1. School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230601 2. School of Computer and Information, Anhui Normal University, Wuhu 241003

指导, 很可能最终导致代码实现架构与设计架构的差异. 研究已经证实这些差异或称错位情形, 对软件系统的质量指标如可用性、可维护性、可靠性等将产生非常负面的影响[6].

最近的软件演化研究多集中于实证分析, 通过软件度量和失效数据等在演化过程中的变化揭示一般规律. 如文献 [7] 提出通过分析驱动演化的错误报告及变更请求等以评估演化过程质量, 其方法完全基于对过程度量数据的标准化衡量, 利于工具实现. 文献 [8] 利用复杂网络对 Linux 操作系统演化过程进行实证研究, 通过对前后近 1 300 个发布版本中所有 C 函数及其相互调用关系构建有向网络, 展现了网络在规模、入/出度、聚类系数等不同拓扑性质下的演化过程. 利用复杂网络拓扑属性分析, 作者揭示了函数模块的各类演化形式, 并指出主要组件函数模块演化的统计学规律. 而文献 [9] 在对 2002 年 ∼ 2016 年 Linux 各版本的 Bug 报告进行分类整合基础上, 重点关注了故障触发在版本演化过程中的规律性特征, 指明 Linux 组件模块在测试时的重要性排列以及聚类系数对衡量错误类型比例的作用. 上述研究使用的模型或方法具有新颖性与可操作性, 但其关注的重点在对既有演化过程的数据分析, 对演化过程及演化行为本身的描述并没有论及. 此外, 文献 [10] 通过构造 Markov 过程用以表示运行在多重环境下的系统演化, 用于系统可靠性及环境可靠性的数值分析. 文献 [11] 关注了使用演化博弈论解决复杂网络环境下个体间间接互惠及合作演化的问题, 对演化行为研究提供了新的思路. 在文献 [3] 中, 作者提出了一种基于 QVT 语言的方法, 将源码层面的演化行为转换到 SA 层面以缓解架构与代码的失配问题. 上述方法并不面向演化过程建模, 且方法的易用性对开发者而言也并非友好. 本文旨在使用轻量且抽象程度更高的代数方法建立软件演化过程的序列化描述, 侧重于分析序列中演化行为对软件系统整体的影响, 并结合代数描述的可计算性实时得出量化结果.

软件工程活动的主要目的在于开发和维护高质量的软件系统, 对软件演化的定量分析应以提高软件产品质量为出发点. 评价软件质量的指标与方法众多, 本文选取可靠性指标进行研究, 这是因为: 1) 结构化软件可靠性模型 (区别于传统增长类模型) 与 SA 有直接相关性, 它可伴随结构的演变工作, 适用于架构工程师预先评估整个演化过程的质量发展趋势; 2) 可靠性的计算基于对软件结构的精确分析, 这与其他软件质量指标相同或相近 (如可维护性), 使得研究不失一般性特点. 特别地, 当对象为一类安全关键软件系统时, 因其对版本更迭前后的质量抖动更加敏感, 相应的演化需求及演化进程需要更严格的评估及监控, 而可靠性作为最关键的非功能性指标具有重要价值. 基于此, 本文站在可靠性的角度分析软件演化过程及其对软件质量的影响, 主要解决以下三方面的问题:

1) 建立模型以准确描述软件结构的演化;
2) 演化过程中对软件可靠性的实时计算;
3) 对演化关键步骤及趋势的分析.

结构化软件可靠性建模研究始于 Littlewood[12] 的 SMP (Semi-Markov process) 模型, 他首先提出单个组件成功执行概率 (或称组件可靠性) 和在工作流上的组件间控制转移概率是决定系统整体可靠性的两个关键因素. Cheung[13] 在此基础上给出 DTMC (Discrete time Markov chain) 模型, 它明确了二者与系统可靠性之间的函数关系. DTMC 模型相较 SMP 具有强 Markov 性质, 但它仅将组件执行时间看作单位时间并忽略其在建模中的作用. 随后 Laprie[14] 使用 CTMC (Continuous time Markov chain) 将组件执行时间均值作为建模参数引入, 用于刻画系统执行稳态. 需要指出的是, 在 CTMC 模型中执行时间必须服从指数分布以满足 Markov 性质, 而近来有研究认为, 这一限定在更复杂的应用场景中已不合时宜. 如 Zheng 等[15] 在分析 Web 服务的性能与可靠性时回归了 SMP 方法, 其强调在单个服务上的逗留时间满足一般分布, 并通过在转移时间点 (或称更新点) 上建立 DTMC, 以计算多种结构类型的组合服务可靠性和单个/组合服务的响应时间均值、方差, 结果可用于可靠性及性能瓶颈分析. 进一步地, 通过将一维 SMP 泛化至二维 MRGP (Markov regenerative process), 可描述服务端、用户端在不同场景、策略、行为下的组合状态迁移, 其在用户感知的服务性能评估上优于传统 CTMC 模型[16].

上述模型和方法的差异在对组件执行时间的处理上, 而本文主要讨论软件演化可靠性分析一般性方法, 倾向使用相对简便的 DTMC 模型, 以突出需要解决的核心问题是对软件演化的描述与可靠性实时分析. 而解决问题的关键在于引入代数方法. 通过代数方法将软件演化过程序列化, 并跟踪分析序列中每一步操作, 使得整个演化过程受到完整监控; 同时代数方法本身的精确、轻量、易用等特征也确保了架构设计者的意图能够被开发人员准确理解, 且不会造成他们太多额外的负担. 本文余下的内容组织如下: 第 1 节将介绍相关知识背景, 包括软件结构化可靠性分析的主流方法以及简单增量式演化的计算; 第 2 节给出了结构演化的实例, 说明对其描述的困难程度; 第 3 节讨论如何使用代数方法构建面向演化过程的序列化模型; 两个算例在第 4 节中被深入讨论, 以验证代数方法的有效性和易用性.

## 1 DTMC 模型与增量式演化

软件可靠性分析的关键在于模型的选择. 传统的软件可靠性增长类模型基于测试期失效数据, 并不适用于软件结构演化时的可靠性分析. 这里只介绍主流的结构化模型分析方法, 其基本模型为 Cheung 的 DTMC 模型[13]. 下文以欧洲航天局 ESA 的小型控制系统软件[17] 为例进行说明.

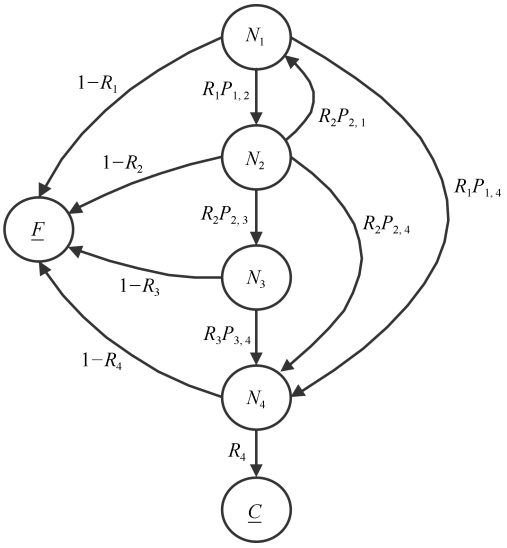



图 1 ESA 软件组件迁移图

Fig. 1 The component transition diagram of ESA software

由图 1 示, 该软件系统含有 4 个主要组件 $N_1 \sim N_4$, 图中组件节点反映了系统执行的 4 个稳态, 节点间的控制转移以有向弧表示, 弧上标注转移概率. 注意到图 1 中包含两个吸收态$\underline{F}$与$\underline{C}$, 分别表示组件失效后到达以及成功执行到达的状态节点. 去除节点$\underline{F}$与$\underline{C}$后, 可建立 DTMC 一步随机转移矩阵 $Q$ 如下:

$$Q = \begin{array}{c} \\ N_1 \\ N_2 \\ N_3 \\ N_4 \end{array} \begin{array}{c} \begin{array}{cccc} N_1 & N_2 & N_3 & N_4 \end{array} \\ \begin{pmatrix} 0 & R_1P_{1,2} & 0 & R_1P_{1,4} \\ R_2P_{2,1} & 0 & R_2P_{2,3} & R_2P_{2,4} \\ 0 & 0 & 0 & R_3P_{3,4} \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

其元素 $Q_{i,j}$ 表示了在 Markov 过程中从状态节点 $i$ 转移至节点 $j$ 的转移概率. 如 $Q_{1,2} = R_1P_{1,2}$, 它反映由节点 $N_1$ 经一步转移至节点 $N_2$ 的概率等于成功执行 $N_1$ 组件的概率 $R_1$ 与转移分支概率 $P_{1,2}$ 的乘积. 矩阵 $Q$ 的 $n$ 次幂 $\mathrm{Q}^n$ 为 $n$ 步随机转移矩阵, 其元素 $Q^n_{i,j}$ 反映了由节点 $i$ 经 $n$ 步转移至节点 $j$ 的概率. 而 $Q$ 的 Neumann 级数 $S$ 表达了由 $N_1$ 经所有可能步数成功到达 $N_4$ 的概率和, 即

$$S = I + Q + Q^2 + \cdots = \sum_{k=0}^{\infty} Q^k = (I - Q)^{-1} \quad (1)$$

其中, $I$ 为单位矩阵, 并且易知级数收敛. 对 ESA 软件系统, 其整体可靠性计算为

$$R_{sys} = S_{1,4} \cdot R_4 \quad (2)$$

即由节点 $N_1$ 出发成功到达 $N_4$, 并正确执行 $N_4$ 的概率.

由式 (2), 可认为 $R_{sys}$ 为单个组件可靠度 $R_i$ 的函数. 称 $B_i$ 为系统整体可靠性对组件 $N_i$ 的敏感度, 有

$$B_i = \frac{\partial R_{sys}}{\partial R_i} = \frac{\partial}{\partial R_i}(S_{1,4}R_4), \quad i = 1, 2, 3, 4 \quad (3)$$

一个明显的结论是 $B_4 = S_{1,4}$, 但其余并不容易得出.

DTMC 模型可以直接计算一类简单的增量式演化, 增量指的是单个组件因为改动而导致的可靠度增加或降低[18]. 令 $A = I - Q$, $B = M_{4,1}$, 其中, $M_{4,1}$ 为元素 $A_{4,1}$ 的余子式. 考查组件 $N_3$ 获得增量 $\delta$ 后的情形, 演化后 $S_{1,4}$ 可计算为

$$S_{1,4} = (-1)^{4+1}\frac{|B|(1 + \delta \cdot B^{-1}_{3,3})}{|A|(1 + \delta \cdot A^{-1}_{4,3})} \quad (4)$$

式 (4) 即为 $N_3$ 增量演化后的整体可靠性计算方法. 对单个组件的简单增量式演化行为, 可靠性分析的重点在于由增量幅度与整体可靠性变化幅度的关系, 这显然与式 (3) 的 $B_i$ 有关.

考虑余子式一般情形, 令 $U = M_{n,1}$, 可展开整理为

$$|U| = -\sum_{j=1}^{n-1}(R_iP_{i,j+1}C_{i,j} + C_{i,i-1}) := f_1(R_i) \quad (5)$$

其中, $C_{i,j}$ 为元素 $U_{i,j}$ 的代数余子式, 规模为 $(n-2) \times (n-2)$.

对矩阵 $A = I - Q$, 可整理为

$$|A| = -\sum_{j=1}^{n}(R_iP_{i,j}D_{i,j} + D_{i,i}), := f_2(R_i) \quad (6)$$

其中, $D_{i,j}$ 为元素 $A_{i,j}$ 的代数余子式, 规模为 $(n-1) \times (n-1)$. 综合式 (5) 和式 (6), 对式 (3) 加以改进, 得到

$$B_i = \frac{\partial R_{sys}}{\partial R_i} = \frac{\partial}{\partial R_i}\left((-1)^{n+1}\frac{f_1(R_i)}{f_2(R_i)}R_n\right) \quad (7)$$

此为单组件增量式演化后的敏感度, 推导过程这里不再展开. 可知计算过程非常复杂, 并且时间复杂度

在 $O(n^3)$ 量级, 这意味着随着转移矩阵 $Q$ 规模的增长, 计算负荷问题将凸显.

## 2　软件结构的演化

区别于简单增量式演化, 本节讨论更一般的情形. 软件系统在发布后会不断调整其体系结构以适应需求或者运行环境的变化. 这些调整即演化行为可能来自软件的自适应机制 (例如服务组件的动态匹配), 也可能来自软件版本的更新 (出于功能修补、性能优化等目的). 站在结构度量的角度, 我们希望可以完整跟踪软件的演化行为, 以确定软件某些关键性能指标 (可靠性、可用性、可维护性等) 的变化趋势, 并定位那些导致整体性能抖动剧烈的单个组件或局部结构模块.

以上一节 ESA 系统为例. 图 2 标注了各组件可靠度及组件间转移概率, 数据来自文献 [17]. 假设该软件在运行期间发生了演化, 其主要模块 Parsing、Computing 及 Formatting 得到了更新, 同时系统结构也因为组件接口更迭和局部性能优化发生改变. 图 3 表示其在演化发生后的情形. 阴影表示组件 $N_1 \sim N_3$ 已被更新, 可靠度也相应发生变化: 解析模块因算法改进可靠度得以提升 (增加 0.04); 格式模块因接口增加导致可靠度下降 (减少 0.02); 组件 $N_5$ 作为 $N_2$ 的复制被加入到结构中, 用来分担系统实时计算的压力, 并且为了进一步优化性能, $N_2$ 与 $N_5$ 耦合为并行结构, 同时 $N_2$ (及 $N_5$) 因为并行功能扩展导致其可靠度降低 (均减少 0.01).

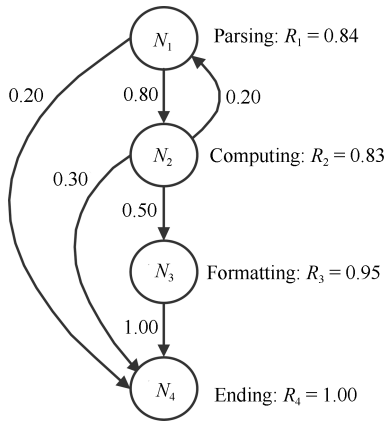



图 2　ESA 软件组件迁移图 (标注可靠性信息)

Fig. 2　The component transition diagram of ESA software (labeled with reliability information)

图 3 较图 2 有多处更改, 这些更改并非同时发生, 而是遵循一个演化过程来进行. 过程中的每一步都由一个或多个操作组成, 而这些操作 (下文称演化原子操作) 都会成为影响系统整体可靠性的小的变量. 如果忽略过程直接使用 DTMC 模型, 对软件开发并没有实际的指导意义. 我们认为针对软件演化的可靠性分析应兼具宏观与微观的视角: 微观上, 定位演化过程中这些小粒度原子操作以及它们所对应的组件模块和局部结构, 可暴露软件系统演化时的潜在风险, 将有利于分析影响整体可靠性的关键因素; 宏观上, 将过程看作操作序列的累积, 可用于分析完整演化进程的可靠性趋势.

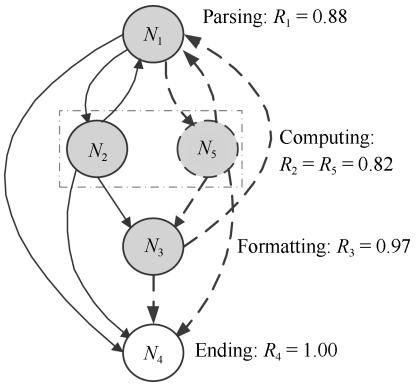



图 3　ESA 软件组件迁移图 (演化后)

Fig. 3　The component transition diagram of ESA software (after evolution)

满足上述需求的前提是对演化过程的理解与表达. 事实上, 追踪和描述软件系统中模块及模块间关系的变化过程一直是软件可视化技术的研究热点, 可视化技术可以帮助开发者兼具静态、动态及演化的视角以分析结构和代码[19]. 但当前可视化技术仅聚焦于多视角分析, 因计算和描述能力的限制尚无法揭示演化过程中的潜在效用变化, 故仍然不适用于完整分析软件演化过程[20]. 图 4 给出了一个针对图 3 演化版本的系列图, 使用了可视化技术中常见的小格图[21] (Small multiples) 表达.

小格图显示了软件结构沿时间 $t$ 连续变化的过程: 在 $t = 2$ 时刻, 模块 $N_1$、$N_3$ 已经完成了更新; 在 $t = 3$ 时刻加入了 $N_2$ 的复制 $N_5$, 同时也相应增加了模块间关系 (边). 有时为了清晰反映单步变化过程, 也可以在两个时刻间插入中间态 (可使用动画), 例如这里的 $t = 2 \to 3$. 需要说明的是, 即使插入中间态的动画过渡, 也无法准确描述出潜在关键信息. 如这里 $t = 4$, 相较于前一时刻新增了模块 $N_2$ 与 $N_5$ 之间的并行关系, 但是这种特殊的耦合结构在图中没有明确表达, 而这又是影响系统整体可靠性的关键信息. 诸如小格图 (包括其改进)、Difference maps[22] 以及 Glyphs[23] 等可视化方法, 在描述演化进程时都存有类似问题, 并且它们都无法回避占用计算空间大、仍需手动对图识别比较等缺点.

下节将引入一种代数方法, 它更轻量化易于描述和计算, 能准确表达演化步骤和过程.

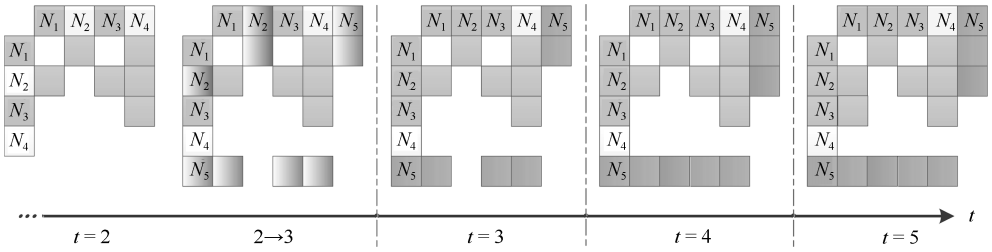



图 4 ESA 软件演化过程的小格图表示

Fig. 4 Evolution process of ESA software represented by the small multiples

## 3 代数方法

将图 2 所示的软件结构定义为三元组 $\langle C, O, \Omega\rangle$, 其中, $C$ 为组件模块集, $O$ 为使用连接子集, $\Omega$ 为模块关系集. 它的完整含义如下:

$\langle C, O, \Omega\rangle$

$C = \{N_1, N_2, N_3, N_4\}$

$O = \{\oplus\}$

$\Omega = \{Role_1 = N_1 \oplus N_2,\ Role_2 = N_1 \oplus N_4,$
$Role_3 = N_2 \oplus N_1,\ Role_4 = N_2 \oplus N_3,\ Role_5 = N_2 \oplus N_4,\ Role_6 = N_3 \oplus N_4\}$

称上述三元组 $\langle C, O, \Omega\rangle$ 为 ESA 软件结构代数模型, 其中关系集 $\Omega$ 含 6 个代数表达式. 这里沿用文献 [24] 的定义, 将连接子定义为代数算子形式, 关系集即为由组件模块和代数算子连接而成的表达式集合. 此例中仅含激发算子 $\oplus$, 它被用来表述组件模块间最基本的交互方式. 如 $\Omega$ 中的第 1 个表达式 $Role_1 = N_1 \oplus N_2$, 其涵义为 $N_1$ 对 $N_2$ 进行了一次激发, 激发动作完成后, 系统执行稳态将由 $N_1$ 迁移至 $N_2$. 更多的算子可参考笔者前期所做的工作[25].

作为对软件体系结构的高度抽象, 代数模型优势在于轻量化和可计算性方面. 当相关参数信息完整时, 使用现有形式语言分析技术对关系集 (即表达式集) 进行一趟扫描, 即可完整计算系统整体可靠性数值. 笔者于文献 [25] 中验证了一个语法分析器, 其算法基于使用广泛的 LR(1) 分析, 并针对代数模型进行优化以保证可扫描并识别出 $\Omega$ 中所有表达式. 在此基础上, 图 5 给出了本文的可靠性自动分析流程. 在建立代数模型之后, 流程可对模型进行预处理和扫描解析. 扫描过程中每当匹配成功一个基本结构范式, 在状态空间对应更新系统状态节点; 当扫描结束, 一个状态空间上的 DTMC 全部节点信息获取完成, 参照式 (2) 即可完成一次整体可靠性计算. 需要说明的是, 一次可靠性计算并不意味着流程终止, 每当新的演化需求产生, 其代数形式表达将用以更新现有代数模型, 流程将自动重走一趟以完成针对此次演化的可靠性实时计算.

限于篇幅, 对代数模型的预处理及扫描解析算法不再重述. 下文详细解释框架中对演化处理的部分. 对应图 3, 首先将 ESA 软件系统演化后代数模型表示如下:

$\langle C', O', \Omega'\rangle$

$C' = \{N_1, N_2, N_3, N_4, N_5\}$

$O' = \{\oplus\}$

$\Omega' = \{\ Role_1 = N_1 \oplus (N_2 \parallel N_5),\ Role_2 = N_1 \oplus N_4,$
$Role_3 = (N_2 \parallel N_5) \oplus N_1,\ Role_4 = (N_2 \parallel N_5) \oplus N_3,$
$Role_5 = (N_2 \parallel N_5) \oplus N_4,\ Role_6 = N_3 \oplus N_4,$
$Role_7 = N_3 \oplus N_1\ \}$

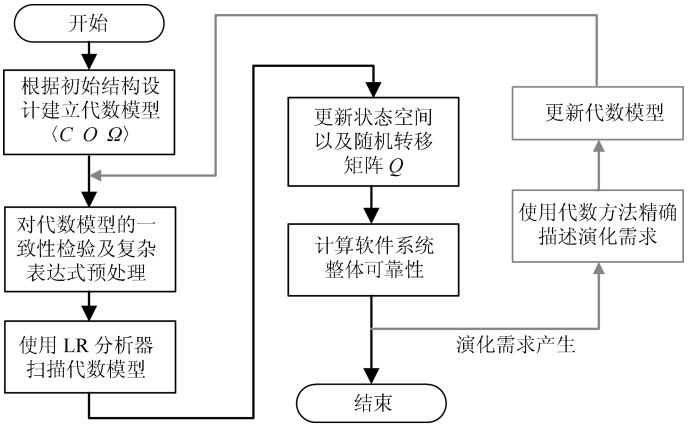



图 5 面向软件演化的可靠性分析流程

Fig. 5 The reliability analysis process for software evolution

这里算子 ‖ 表示并行耦合关系.

如前所述, 认为从图 2 至图 3 必然经历一个演化过程. 站在结构分析的角度, 演化过程可看成由若干原子操作组成的行为序列. 为了方便讨论, 首先给出如下记号:

1) $A_i$, 结构中间版本, 对应第 $i$ 步演化步骤相对前一版本 $A_{i-1}$ 的更改, 其中, $A_0$ 为初始版本, $A_n$ 为演化完成版本;

2) $Q_i$, 对应版本 $A_i$ 的整体软件质量度量 (这里只讨论可靠性, 以 $R_i$ 代替);

3) $D_i = |Q_i - Q_{i-1}|$, 用来表示相邻版本质量之差, 这里总是取正值以描述演化过程中的可靠性抖动程度. 图 6 反映了上述记号之间的关系.

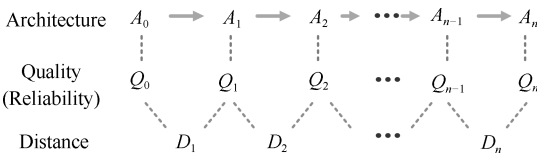



图 6　软件演化过程的版本表示

Fig. 6　The version-expression in software evolution process

在具体演化操作方面, 文献 [26] 在软件质量相关度量基础上给出了演化原子操作分类, 但其并不适用于基于结构分析的可靠性建模. 文献 [7] 从故障数、变动率以及人工成本等角度对演化操作进行数据分析与度量, 但其仍非面向过程的方法, 无法精确描述演化行为. 因此本文给出如表 1 所示演化原子操作分类及定义, 用以面向建模的精确性及可计算性.

表 1 强调了演化操作定义的原子性, 即操作不可再分. 每种操作都将对应更新代数模型的三个集合 $C$、$O$ 及 $\Omega$. 如增加组件 AM, 其对应了组件集 $C$ 中元素的增加, 而移除组件间依赖 RMD 会使得关系集 $\Omega$ 中的表达式被移除.

表 1　演化原子操作分类

Table 1　Classification of evolutionary atomic operations

| 名称 | 定义描述 |
|---|---|
| AM | 增加组件模块 |
| RM | 移除组件模块 |
| AMD | 增加模块间依赖关系 |
| RMD | 移除模块间依赖关系 |
| UM | 更新组件模块 (算法、功能) |
| UMD | 更新模块间依赖关系 (接口) |
| SM | (单个) 模块分割 |
| UM | (多个) 模块耦合 |

注意到原子操作间具有较强关联性, 一种操作往往会关联另一种操作, 如结构中增加了新的组件模块, 因其必然参与系统执行稳态的控制转移, 故会导致模块间依赖关系的更新. 单个演化步骤中不应只对应单个原子操作, 有时也须考虑若干原子操作相关联的情况.

结合上文演化前代数模型 $\langle C, O, \Omega\rangle$ 和演化后模型 $\langle C', O', \Omega'\rangle$, 可对演化过程作如下描述:

$A_0$–$\langle C, O, \Omega\rangle$

$A_1$–UM $C\{N_1\}$,　$A_2$–UM $C\{N_3\}$

$A_3$–AM $C\{N_5\}$ and AMD $\Omega\{N_1 \oplus N_5, N_5 \oplus N_1, N_5 \oplus N_3, N_5 \oplus N_4\}$ and UMD $\Omega\{N_1 \oplus N_2\}$

$A_4$–UM $C\{N_2\, N_5\}$, $A_5$–CM $O\{\|\}\Omega\{N_2 \parallel N_5\}$

$A_6$–AMD $\Omega\{N_3 \oplus N_1\}$ and UMD $\Omega\{N_3 \oplus N_4\}$

$A_6$–$\langle C', O', \Omega'\rangle$#

其中, $A_6$ 为演化完成版本. 注意这里 $A_4$ 步完成后, 应将 $\Omega$ 集中所有 $N_2$ 及 $N_5$ 处替换为 $N_2 \parallel N_5$ 并删除冗余项.

中间版本 $A_1$、$A_2$ 及 $A_4$ 关联原子操作 UM, 对应简单的增量式演化. 注意除了 $A_1$ 外, 其余不能直接套用式 (4) 计算可靠性, 因为版本 $A_2$、$A_4$ 已经 "身处" 演化进程之中, 它们均是对前一版本更改而非对初始版本更改. 相对地, 包括处理 $A_5$ 中更复杂的结构演化情形, 本文提出流程框架展现了良好的适用性: 通过对表达式集合 $\Omega$ 的一趟扫描, 各中间版本的计算可实时自动完成. 就演化过程建模而言, 代数方法相对小格图等其它方法强调了过程描述的精确性与完整性.

根据上述演化模型并结合由图 2 演化至图 3 的具体数据变动, 流程自动计算从 $A_0$ 至 $A_6$ 各版本系统整体可靠性. 图 7 反映了可靠性数值在演化进程中的趋势, 可以看到: 在 $A_4$ 版本之前, 系统整体可靠性维持在一个较稳定水平; 当到达 $A_5$ 版本后, 可靠性数值显著下降, 这是由于并行结构的引入使得组件间耦合度增加, 从而对可靠性造成负面影响. 图

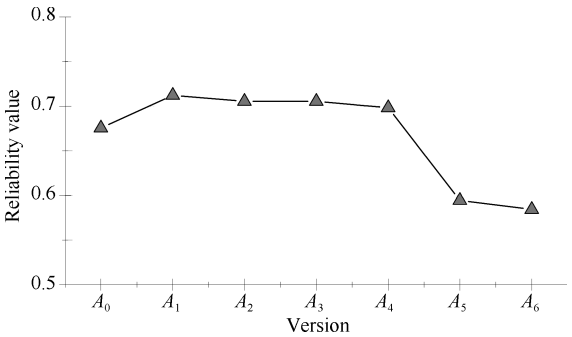



图 7　ESA 软件演化过程可靠性变化趋势

Fig. 7　The reliability trends in evolution process of ESA software

8 显示演化时的版本抖动程度, 其数值意义为相邻版本的可靠性差值: $D_5$ 明显区别于其余值, 反映了由版本 $A_4$ 演变至版本 $A_5$ 的原子操作是影响整个演化过程的关键; $D_1$、$D_2$ 与 $D_4$ 对应了简单增量式演化 (对应 UM 操作), 其值一定程度上可揭示组件的重要度, 与式 (7) 所计算的敏感度值相似, 但计算难度大幅降低.

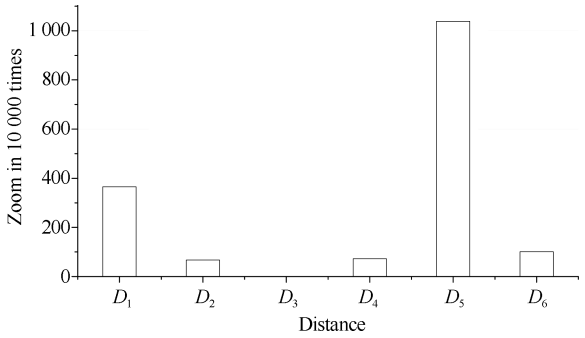



图 8 演化版本间抖动程度

Fig. 8 The reliability distance between evolution versions

## 4 算例研究

本节使用两个算例以验证本文提出代数方法的有效性与实用性.

### 4.1 算例 1

#### 4.1.1 数据介绍及参数设置

算例 1 数据来自文献 [27], 该算例因具有典型性故被广泛应用于可靠性模型验证. 最近对该算例的研究仍在持续, 如在文献 [28] 中被用以比较一类 DTMC 模型的性能. 图 9 中标明了算例 1 的可靠性相关参数设置, 包括单组件 $N_i$ 的可靠性数值 $R_i$, 以及组件间控制转移概率 $P_{i,j}$. 图 9 含该系统的初始结构设计: 系统含 15 个组件模块, 初始设计时并不含特殊结构, 即组件间仅以最基本的激发方式进行交互. 为进行算例的演化验证, 这里设定系统最终发布时, 含有并行、容错及调用返回三种特殊结构类型, 它们分别是: 模块 $N_3$、$N_4$ 构成并行结构、模块 $N_{10}$ 对 $N_9$ 构成容错结构、模块 $N_{11}$、$N_8$ 以及 $N_8$、$N_5$ 构成调用返回结构.

#### 4.1.2 方法运行及阶段性结果

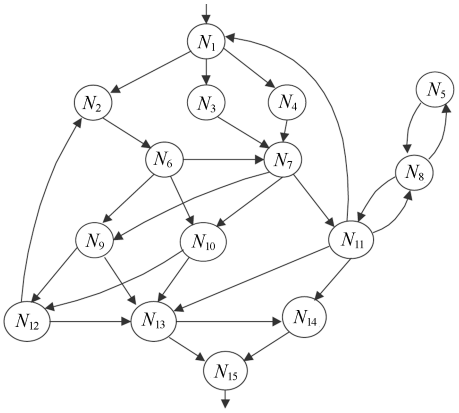
根据系统初始结构设计, 建立代数模型如下:

$\langle C, O, \Omega\rangle$

$C = \{N_1, N_2, N_3, N_4, N_5, N_6, N_7, N_8, N_9, N_{10}, N_{11}, N_{12}, N_{13}, N_{14}, N_{15}\}$

$O = \{\oplus\}$

$\Omega = \{Role_1 = N_1 \oplus N_2,\ Role_2 = N_1 \oplus N_3,\ Role_3 = N_1 \oplus N_4,\ Role_4 = N_2 \oplus N_6,\ Role_5 = N_3 \oplus N_7, Role_6 = N_4 \oplus N_7,\ Role_7 = N_5 \oplus N_8,\ Role_8 = N_6 \oplus N_7, Role_9 = N_6 \oplus N_6,\ Role_{10} = N_6 \oplus N_{10},\ Role_{11} = N_7 \oplus N_9,\ Role_{12} = N_7 \oplus N_{10},\ Role_{13} = N_7 \oplus N_{11},\ Role_{14} = N_8 \oplus N_5,\ Role_{15} = N_8 \oplus N_{11},\ Role_{16} = N_9 \oplus N_{12},\ Role_{17} = N_9 \oplus N_{13},\ Role_{18} = N_{10} \oplus N_{12},\ Role_{19} = N_{10} \oplus N_{13},\ Role_{20} = N_{11} \oplus N_1,\ Role_{21} = N_{11} \oplus N_{13},\ Role_{22} = N_{11} \oplus N_{14},\ Role_{23} = N_{12} \oplus N_2,\ Role_{24} = N_{12} \oplus N_{13},\ Role_{25} = N_{13} \oplus N_{14},\ Role_{26} = N_{13} \oplus N_{15},\ Role_{27} = N_{14} \oplus N_{15}\}$




$R_1 = 0.998,\ R_2 = 0.990,\ R_3 = 0.980,\ R_4 = 0.995$
$R_5 = 0.999,\ R_6 = 0.985,\ R_7 = 0.996,\ R_8 = 0.975$
$R_9 = 0.990,\ R_{10} = 0.998,\ R_{11} = 0.950,\ R_{12} = 0.965$
$R_{13} = 0.970,\ R_{14} = 0.980,\ R_{15} = 0.992$

$P_{1,2} = 0.40,\ P_{1,3} = P_{1,4} = 0.30$
$P_{2,6} = P_{3,7} = P_{4,7} = P_{5,8} = P_{14,15} = 1.00$
$P_{6,7} = 0.10,\ P_{6,9} = P_{6,10} = 0.45,\ P_{7,11} = 0.25$
$P_{7,9} = P_{7,10} = 0.375,\ P_{8,5} = 0.20,\ P_{8,11} = 0.80$
$P_{9,12} = P_{10,12} = 0.70,\ P_{9,13} = P_{10,13} = 0.30$
$P_{11,1} = P_{11,14} = 0.15,\ P_{11,8} = 0.20,\ P_{11,13} = 0.50$
$P_{12,2} = P_{12,13} = 0.50,\ P_{13,14} = 0.40,\ P_{13,15} = 0.60$

图 9 算例 1 初始结构与相关参数

Fig. 9 The reliability trends in evolution process of ESA software

经流程框架自动扫描并计算, 初始结构整体可靠性数值为 0.8762. 令初始结构版本 $A_0$ 为演化起点, 以最终发布版本 $A_{17}$ 为演化终点, 将演化过程作如下描述:

$A_0$–$\langle C, O, \Omega\rangle$

$A_1$–UM $C\{N_1\}$, $A_2$–UM $C\{N_2\}$, $A_3$–UM $C\{N_3\}$

$A_4$–UM $C\{N_4\}$, $A_5$–UM $C\{N_5\}$, $A_6$–UM $C\{N_6\}$

$A_7$–UM $C\{N_7\}$, $A_8$–UM $C\{N_8\}$, $A_9$–UM $C\{N_9\}$

$A_{10}$–UM $C\{N_{10}\}$, $A_{11}$–UM $C\{N_{11}\}$

$A_{12}$–UM $C\{N_{12}\}$, $A_{13}$–UM $C\{N_{13}\}$

$A_{14}$–UM $C\{N_{14}\}$, $A_{15}$–UM $C\{N_{15}\}$

$A_{16}$–CM $O\{\|\}\Omega\{N_3 \parallel N_4\}$

$A_{17}$–CM $O\{\Psi\}\Omega\{N_9 \Psi N_{10}\}$

$A_{17}$–$\langle C', O', \Omega' \rangle$#

这里 ‖ 为并行算子, Ψ 为容错算子.

中间版本 $A_1 \sim A_{15}$ 对应单个组件的简单增量式演化. 设定因平台迁移使用了新的事件系统, 所有组件实现发生变化, 导致可靠度平均下降了 0.005. 版本 $A_{16}$、$A_{17}$ 中使用了 CM 操作, 分别对应 $N_3$、$N_4$ 以及 $N_9$、$N_{10}$ 进行结构耦合后的情形. 表 2 中列出演化过程的中间计算结果. 其中各演化版本可靠性 $R_i$ 的计算由流程框架自动完成, 图 10 表达演化过程的可靠性趋势. 因为组件平均可靠度的降低, 系统整体可靠性呈下行趋势, 当到达最低点 $A_{16}$ 版本后, 因为 $A_{17}$ 中引入容错机制的原因使得可靠性最终有所回升.

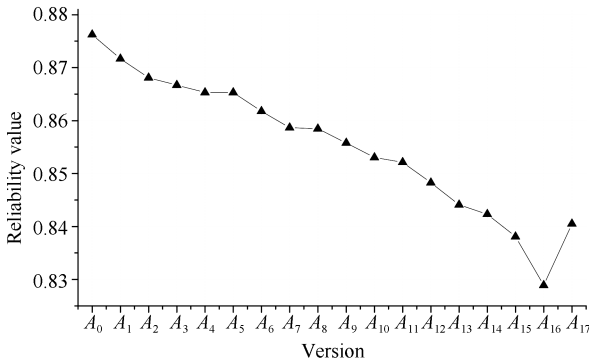



图 10　算例 1 演化可靠性变化趋势

Fig. 10　Reliability trends in evolution of Case 1

### 4.1.3　最终结果及分析

就面向过程的可靠性评价而言, 本文倾向于使用类似图 10 的可靠性趋势序列表达结果, 以代替对最终版本的单次可靠性计算. 除趋势外, 将演化过程序列化的另一优势在于敏感度分析, 即指出在演化过程中造成整体可靠性抖动明显的关键步骤及其背后关联的原子操作. 如代数方法描述, 中间版本 $A_1 \sim A_{15}$ 对应 UM 原子操作, 即单个组件更新的情形. 因为组件可靠度平均下降幅度相同, 故相邻版本质量差值 $D_1 \sim D_{15}$ 很大程度上可反映被更新组件 $N_1 \sim N_{15}$ 的重要程度. 为说明其有效性, 表 2 中也给出了可靠性关键程度 (Criticality) 的计算结果, 它由下式定义:

$$C_i = \frac{\Delta R_{sys}}{\Delta R_i} \tag{8}$$

其中, $\Delta R_i$ 为单个组件 $N_i$ 的可靠性变化增量, $\Delta R_{sys}$ 对应因此引起的系统整体可靠性增量. 当组件的平均可靠度增量幅度非常小时 ($\leq 0.005$), $C_i$ 能够近似代替敏感度 $B_i$, 而相较于 $C_i$, 计算 $B_i$ 的代价通常要大的多.

图 11 中为了与前 15 个 $D_1 \sim D_{15}$ 作对比, 一组 Criticality 值 $C_1 \sim C_{15}$ 以曲线形式呈现 (经过适当放大). 观察 Criticality 曲线变化与下方的 Distance 图形基本一致. 除去起始节点组件 $N_1$ 与终止节点组件 $N_{15}$ 外, 组件 $N_{13}$ 的 Criticality 值 ($C_{13} = 0.87296$) 最大而组件 $N_{12}$ 其次 ($C_{12} = 0.79360$), 这与 $D_{13}$ (0.00417) 及 $D_{12}$ (0.00387) 是吻合的. 这表明演化过程含多个简单增量式演化步骤时, 通过计算 Distance 值 (增量幅度不同时考虑 $D_i/\Delta R_i$ 比值) 评估不同组件的可靠性关键程度是有效的. 而计算 Criticality 值几乎没有代价, 它仅是为分析演化过程保留的中间结果.

图 11 体现了演化过程的可靠性抖动情况. 可看到处于最后的 $D_{16}$、$D_{17}$ 明显高出其余 Distance 值一个量级, 这说明涉及结构的演化原子操作 (SM、CM) 对系统整体可靠性的影响往往最为关键. 其中又以 $D_{17}$ 的值最为突出, 这是因为容错结构本身即具有高可靠特征, 如果对结构中关键节点 (可靠性敏感的) 组件进行容错结构的演化设计, 可对系统整体可靠性提升起显著作用. 从架构设计者立场, 需要与编码人员一起严格监控以使演化过程中的质量抖动幅度被限制在合理的区间内. 序列化的代数

表 2　算例 1 演化过程计算结果

Table 2　Evolution process calculation results of Case 1

| 版本可靠性 | | 相邻版本之差 | | 组件关键程度 | | 版本可靠性 | | 相邻版本之差 | | 组件关键程度 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $R_0$ | 0.87623 | | | | | $R_9$ | 0.85580 | $D_9$ | 0.00268 | $C_9$ | 0.55552 |
| $R_1$ | 0.87167 | $D_1$ | 0.00456 | $C_1$ | 0.91264 | $R_{10}$ | 0.85302 | $D_{10}$ | 0.00278 | $C_{10}$ | 0.55552 |
| $R_2$ | 0.86810 | $D_2$ | 0.00357 | $C_2$ | 0.73408 | $R_{11}$ | 0.85213 | $D_{11}$ | 0.00089 | $C_{11}$ | 0.19840 |
| $R_3$ | 0.86671 | $D_3$ | 0.00139 | $C_3$ | 0.27776 | $R_{12}$ | 0.84826 | $D_{12}$ | 0.00387 | $C_{12}$ | 0.79360 |
| $R_4$ | 0.86532 | $D_4$ | 0.00139 | $C_4$ | 0.27776 | $R_{13}$ | 0.84409 | $D_{13}$ | 0.00417 | $C_{13}$ | 0.87296 |
| $R_5$ | 0.86532 | $D_5$ | 0.00000 | $C_5$ | 0.00000 | $R_{14}$ | 0.84231 | $D_{14}$ | 0.00179 | $C_{14}$ | 0.37960 |
| $R_6$ | 0.86175 | $D_6$ | 0.00357 | $C_6$ | 0.73408 | $R_{15}$ | 0.83806 | $D_{15}$ | 0.00425 | $C_{15}$ | 0.88330 |
| $R_7$ | 0.85868 | $D_7$ | 0.00308 | $C_7$ | 0.61504 | $R_{16}$ | 0.82888 | $D_{16}$ | 0.00918 | | |
| $R_8$ | 0.85848 | $D_8$ | 0.00020 | $C_8$ | 0.03968 | $R_{17}$ | 0.84053 | $D_{17}$ | 0.01165 | | |

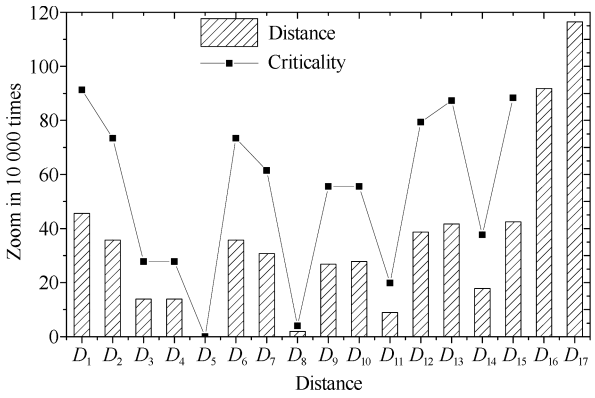



图 11 算例 1 版本抖动与组件关键程度

Fig. 11 Version distance and component criticality of Case 1

建模方法针对了上述需求, 并且方法本身是简洁、易用的.

## 4.2 算例 2

### 4.2.1 数据介绍及参数设置

算例 2 来自一个大型交换机系统的软件结构设计, 它最早被文献 [13] 所引用, 并同样因其结构具有代表性多被用来比较和验证结构化软件可靠性模型. 在文献 [29] 中, 作者对该算例从相关性、敏感度等角度进行了深入分析与讨论, 并与一类基于路径可靠性模型作出比较. 图 12 中标明了算例 2 的可靠性相关参数设置, 同时也给出了系统的初始结构设计. 该系统含有 10 个组件模块, 初始设计不含有特殊结构. 在算例 2 的演化验证中, 将重点关注组件间的控制转移变化对可靠性影响. 这里设定系统结构关键组件 $N_1$、$N_2$ 及 $N_5$ 在最终发布前控制转移分支概率发生变化, 在演化过程中将对这一行为建模并分析分支的可靠性敏感度.

### 4.2.2 方法运行及阶段性结果

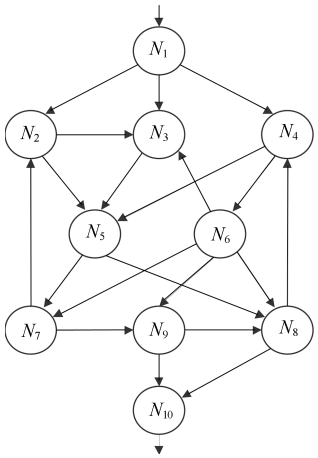
根据系统初始结构设计, 可建立代数模型如下:

$\langle C, O, \Omega \rangle$

$C = \{N_1, N_2, N_3, N_4, N_5, N_6, N_7, N_8, N_9, N_{10}\}$

$O = \{\oplus\}$

$\Omega = \{\ Role_1 = N_1 \oplus N_2,\ Role_2 = N_1 \oplus N_3,\ Role_3 = N_1 \oplus N_4,\ Role_4 = N_2 \oplus N_3,\ Role_5 = N_2 \oplus N_5, Role_6 = N_3 \oplus N_5,\ Role_7 = N_4 \oplus N_5,\ Role_8 = N_4 \oplus N_6, Role_9 = N_5 \oplus N_7,\ Role_{10} = N_5 \oplus N_8,\ Role_{11} = N_6 \oplus N_3, Role_{12} = N_6 \oplus N_7,\ Role_{13} = N_6 \oplus N_8,\ Role_{14} = N_6 \oplus N_9,\ Role_{15} = N_7 \oplus N_2,\ Role_{16} = N_7 \oplus N_9,\ Role_{17} = N_8 \oplus N_4,\ Role_{18} = N_8 \oplus N_{10},\ Role_{19} = N_9 \oplus N_8,\ Role_{20} = N_9 \oplus N_{10}\ \}$




$R_1 = 0.999,\ R_2 = 0.980,\ R_3 = 0.990,\ R_4 = 0.970$
$R_5 = 0.950,\ R_6 = 0.995,\ R_7 = 0.985,\ R_8 = 0.950$
$R_9 = 0.975,\ R_{10} = 0.985,$

$P_{1,2} = 0.60,\ P_{1,3} = P_{1,4} = 0.20$
$P_{2,3} = 0.70,\ P_{2,5} = 0.30,\ P_{3,5} = 1.00$
$P_{4,5} = 0.40,\ P_{4,6} = 0.60$
$P_{5,7} = 0.40,\ P_{5,8} = 0.60$
$P_{6,3} = P_{6,7} = P_{6,9} = 0.30,\ P_{6,8} = 0.10$
$P_{7,2} = P_{7,9} = 0.50,\ P_{8,4} = 0.25$
$P_{8,10} = 0.75,\ P_{9,8} = 0.10,\ P_{9,10} = 0.90$

图 12 算例 2 初始结构与相关参数

Fig. 12 The reliability trends in evolution process of ESA software

对此算例设定二阶段演化过程: 首先在版本 $A_1 \sim A_{10}$ 中使组件 $N_1 \sim N_{10}$ 可靠度依次下降 0.01 (对应 UM 原子操作), 通过实时计算版本可靠性及相邻版本 Distance 值, 识别出结构中的关键组件节点; 其次对关键组件, 调整与之相关的控制转移概率 (对应 UMD 原子操作), 以分析节点转移分支的偏重对整体可靠性影响的程度. 演化过程如下:

$A_0$–$\langle C, O, \Omega \rangle$

Stage 1:

$A_1$–UM $C\{N_1\}$, $A_2$–UM $C\{N_2\}$, $A_3$–UM $C\{N_3\}$
$A_4$–UM $C\{N_4\}$, $A_5$–UM $C\{N_5\}$, $A_6$–UM $C\{N_6\}$
$A_7$–UM $C\{N_7\}$, $A_8$–UM $C\{N_8\}$, $A_9$–UM $C\{N_9\}$
$A_{10}$–UM $C\{N_{10}\}$

Stage 2:

$A_{11}$–UMD $\Omega\{N_1 \oplus N_2,\ N_1 \oplus N_4\}$
$A_{12}$–UMD $\Omega\{N_1 \oplus N_2,\ N_1 \oplus N_3\}$
$A_{13}$–UMD $\Omega\{N_2 \oplus N_3,\ N_2 \oplus N_5\}$
$A_{14}$–UMD $\Omega\{N_5 \oplus N_7,\ N_5 \oplus N_8\}$
$A_{14}$–$\langle C', O', \Omega' \rangle \#$

根据设定, 前 10 个中间版本 $A_1 \sim A_{10}$ 被看作第一阶段, 后 4 个版本 $A_{11} \sim A_{14}$ 看作第二阶段. 参照表 3 中各版本可靠性数值, 图 13 给出了演化过程的整体趋势. 可看到系统可靠性在第一部分呈逐步快速下降趋势, 符合预期. 图 14 中给出了相

应 Distance 值 $D_1 \sim D_{10}$, 经比较易知软件结构中 $N_1$、$N_2$、$N_3$、$N_5$ 及 $N_{10}$ 属于相对可靠性敏感的重要节点组件. 其中, $D_5$ (0.01074) 甚至超过了起始节点 $D_1$ (0.00837) 及终止节点 $D_{10}$ (0.00789), 说明其在结构中的关键程度. 同样地, 为了说明 $D_1 \sim D_{10}$ 值的有效性, 在图 14 中也附以 Criticality 值 $C_1 \sim C_{10}$ (经放大处理), 观察易知其曲线与 Distance 图形基本保持一致, 说明使用 Distance 值分析组件节点的可靠性敏感度是有效的.

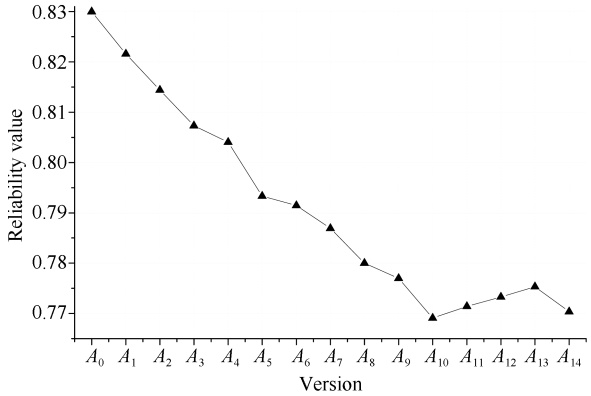



图 13　算例 2 演化可靠性变化趋势

Fig. 13　Reliability trends in evolution of Case 2

### 4.2.3　最终结果及分析

在第一阶段演化过程计算结果基础上, 从中筛选出可靠性敏感程度最高的组件 $N_1$、$N_2$ 及 $N_5$. 因在这些组件上的变动更易于引发明显的可靠性抖动, 利于于后续演化分析.

第二阶段中有 4 个中间版本 $A_{11} \sim A_{14}$, 分别对应以 $N_1$、$N_2$ 及 $N_5$ 为分支节点的 UMD 原子操作. 3 个节点都具有代表性: $N_1$ 处于特殊的起始位置, $N_5$ 具有最大 Distance 值, 而 $N_2$ 是结构内部仅次于 N5 的可靠性关键节点.

对 UMD 操作分析的困难在于多参数情形. 从版本 $A_{11}$ 开始, 设定 $N_1$ 的实现发生更改, 将直接影响其与后续组件 $N_1 \sim N_4$ 之间的控制转移关系, 表现为相关分支概率的变化. 在 $A_{11}$ 步, 两组激发表达式被关联 UMD 操作, 使得分支转移概率比值 $P_{1,2}$:$P_{1,3}$:$P_{1,4}$ 由 0.6:0.2:0.2 更新为 0.5:0.2:0.3, 注意只有 $P_{1,2}$ 与 $P_{1,4}$ 作为参数, 而 $P_{1,3}$ 保持不变. 图 15 (a) 中曲线 II 表达了版本可靠性 $R_{11}$ 在 $P_{1,2}$ 与 $P_{1,4}$ 作为参数情况下所有可能的取值. 当 $P_{1,4}$ 所占比越大时, $R_{11}$ 值越高, 实际 $A_{11}$ 版本较 $A_{10}$ 版本提升了可靠性. 不考虑演化, 曲线 I 给出 $R_{sys}$ 值与参数 $P_{1,2}$、$P_{1,4}$ 之间的关系, 它与曲线 II 一致, 说明了在演化过程中按曲线 II 分析转移概率影响是有效的.

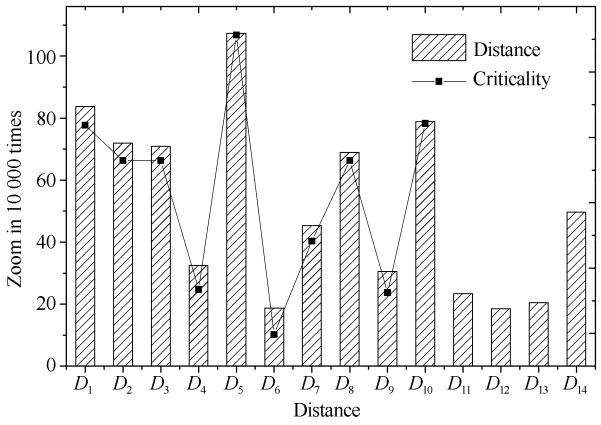



图 14　算例 2 版本抖动与组件关键程度

Fig. 14　Version distance and component criticality of Case 2

在 $A_{12}$ 步, 比值 $P_{1,2}$:$P_{1,3}$:$P_{1,4}$ 经演化更新为 0.4:0.3:0.3. 这一步中 $P_{1,2}$ 与 $P_{1,3}$ 作为参数, $P_{1,4}$ 保持不变. 图 15 (b) 中曲线 IV 对应 $R_{12}$ 所有可能取值, 易知 $R_{12}$ 随着 $P_{1,3}$ 占比增大而提升. 同样地, 曲线 III 表达了 $R_{sys}$ 与参数 $P_{1,2}$、$P_{1,3}$ 之间的关系, 作为参照它与曲线 IV 一致. 版本 $A_{11}$、$A_{12}$ 的两步演化在现实情况下可能只对应一步 (由 0.6:0.2:0.2 至 0.4:0.3:0.3), 增加 1 个可变参数无疑加大了分析难度, 亦不能给出两两比较的直观结果. 故这里设定演化操作可进一步细分 (如此处 $A_{11}$、$A_{12}$), 将更改限制在仅 2 个参数可变的情况. 组件节点拥有超过 3 个及以上的控制转移分支都可按此分析.

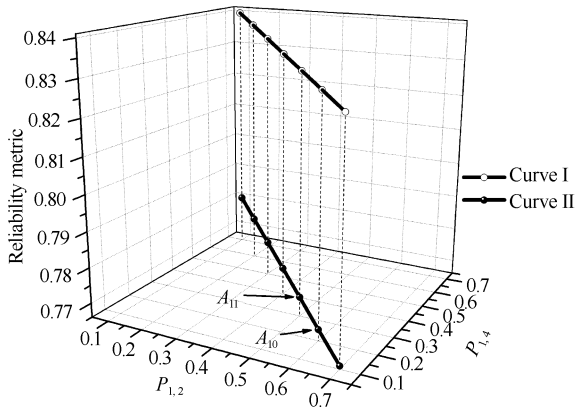
根据表 3, $R_{13}$ 值较前一版本增加, 说明对节点 $N_2$ 而言, 后续控制转移偏重 $N_5$ 分支则有利于可靠性提升. 而 $R_{14}$ 值较前减少, 说明对节点 $N_5$, 控制转移偏重 $N_7$ 将导致可靠性下降. 通过观察图 14 中 $D_{11} \sim D_{14}$ 部分, 第二阶段整体可靠性抖动程度相对平缓, 但其中, $D_{14}$ (0.00497) 明显大于 $D_{11}$ (0.00234)、$D_{12}$ (0.00185) 及 $D_{13}$ (0.00205). 版本 $A_{14}$ 对转移概率的更改幅度与之前版本相近, 其影响程度却明显放大, 从另一角度验证了节点 $N_5$ 在整体结构中是最关键的, 这与文献 [29] 中敏感度分析结论一致.

目前可靠性模型对控制转移分支敏感度的分析仍然缺乏有效的方法, 涉及对多参数的分析往往较为困难. 一类基于路径的可靠性模型[24, 26, 29] 中使用了多个执行路径来对应节点存有多个控制转移的情形, 但是这种方法计算流程繁琐, 不能简明分析关键节点组件的分支敏感度. 本文在序列化的演化过程中使用原子操作细化描述控制转移概率的变化, 每一步操作均通过实时扫描代数模型计算出可靠性变化, 可监测出影响整体可靠性的关键步骤, 具有可操作性与现实意义.

表 3 算例 2 演化过程计算结果

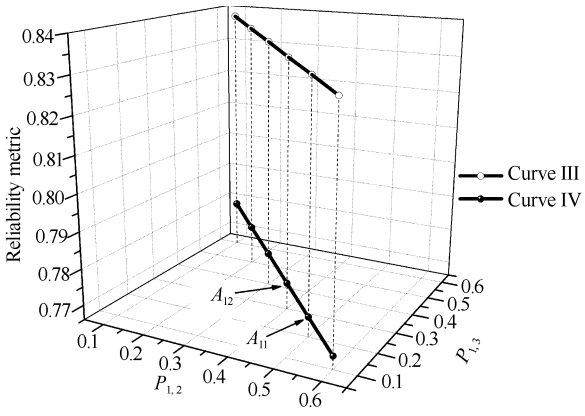
Table 3 Evolution process calculation results of Case 2

| 版本可靠性 | | 相邻版本之差 | | 组件关键程度 | | 版本可靠性 | | 相邻版本之差 | | 组件关键程度 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $R_0$ | 0.82996 | | | | | $R_8$ | 0.78002 | $D_8$ | 0.00690 | $C_8$ | 0.72890 |
| $R_1$ | 0.82159 | $D_1$ | 0.00837 | $C_1$ | 0.83725 | $R_9$ | 0.77697 | $D_9$ | 0.00305 | $C_9$ | 0.32505 |
| $R_2$ | 0.81440 | $D_2$ | 0.00719 | $C_2$ | 0.72890 | $R_{10}$ | 0.76908 | $D_{10}$ | 0.00789 | $C_{10}$ | 0.82996 |
| $R_3$ | 0.80731 | $D_3$ | 0.00709 | $C_3$ | 0.72890 | $R_{11}$ | 0.77142 | $D_{11}$ | 0.00234 | | |
| $R_4$ | 0.80406 | $D_4$ | 0.00325 | $C_4$ | 0.33490 | $R_{12}$ | 0.77327 | $D_{12}$ | 0.00185 | | |
| $R_5$ | 0.79332 | $D_5$ | 0.01074 | $C_5$ | 1.11305 | $R_{13}$ | 0.77532 | $D_{13}$ | 0.00205 | | |
| $R_6$ | 0.79145 | $D_6$ | 0.00187 | $C_6$ | 0.19700 | $R_{14}$ | 0.77035 | $D_{14}$ | 0.00497 | | |
| $R_7$ | 0.78692 | $D_7$ | 0.00453 | $C_7$ | 0.48265 | | | | | | |




(a) $P_{1,2}$, $P_{1,4}$ 作为参数

(a) Function of $P_{1,2}$ and $P_{1,4}$




(b) $P_{1,2}$, $P_{1,3}$ 作为参数

(b) Function of $P_{1,2}$ and $P_{1,3}$

图 15 可靠性受分支概率参数变化的影响

Fig. 15 Reliability as a function of branch probability

## 5 结论

面向演化过程建模与分析一直是软件工程领域的难点问题. 使用代数方法描述软件结构是精确无二义的, 且相较于图形工具在可计算性上具有优势, 适用于可靠性实时分析、计算. 将演化过程序列化是本文的创新点. 演化各中间版本 (步骤) 可独立建模, 中间版本前后衔接为完整的演化过程, 从而建立起过程化分析模型. 本文方法的有效性与易用性得到了算例验证, 下一步, 将在开源软件项目上开展实证研究, 通过对软件更迭版本依序构建演化代数模型, 并基于设计文档、源代码及代码度量数据获取可靠性建模参数, 用以计算和分析可靠性变化趋势, 以及揭示软件版本演化中的规律性特征和一些重要、易被忽视的中间环节.

## References


1 Barnes J M, Garlan D, Schmerl B. Evolution styles: Foundations and models for software architecture evolution. *Software & Systems Modeling*, 2014, **13**(2): 649−678

2 Behnamghader P, Le D M, Garcia J, Shahbazian A, Medvidovic N. A large-scale study of architectural evolution in open-source software systems. *Empirical Software Engineering*, 2017, **22**(3): 1146−1193

3 Haitzer T, Navarro E, Zdun U. Reconciling software architecture and source code in support of software evolution. *Journal of Systems and Software*, 2017, **123**: 119−144

4 Iulian N, Xie G, Chen J. Towards a better understanding of software evolution: An empirical study on open-source software. *Journal of Software: Evolution and Process*, 2013, **25**(3): 193−218

5 Macho H J, Robles G. Preliminary lessons from a software evolution analysis of Moodle. In: Proceedings of the First International Conference on Technological Ecosystem for Enhancing Multiculturality. New York, USA: IEEE, 2013. 157−161

6 Macho H J, Robles G, Nakagawa E Y, Sousa E P M D, Murata K D B, Andery G. Software architecture relevance in open source software evolution: A case study. In: Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference. New York, USA: IEEE, 2008. 1234−1239

7 Sneed H M, Prentner W. Analyzing data on software evolution processes. In: Proceedings of Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement. New York, USA: IEEE, 2016. 1−10

8 Xiao G, Zheng Z, Wang H. Evolution of Linux operating system network. *Physica A: Statistical Mechanics and its Applications*, 2017, **466**: 249−258

9 Xiao G, Zheng Z, Yin B, Trivedi K S, Du X, Cai K. Experience report: fault triggers in linux operating system: from evolution perspective. In: Proceedings of the 28th International Symposium on Software Reliability Engineering. New York, USA: IEEE, 2017. 101−111

10 Liu B, Cui L, Si S, Wen Y. Performance measures for systems under multiple environments. *IEEE/CAA Journal of Automatica Sinica*, 2016, **3**(1): 90−95

11 Zhang Yan-Ling, Liu Ai-Zhi, Sun Chang-Yin. Development of several studies on indirect reciprocity and the evolution of cooperation. *Acta Automatica Sinica*, 2018, **44**(1): 1−12 (张艳玲, 刘爱志, 孙长银. 间接互惠与合作演化的若干问题研究进展. 自动化学报, 2018, **44**(1): 1−12)

12 Littlewood B. Software reliability model for modular program structure. *IEEE Transactions on Reliability*, 1979, **R-28**(3): 241−246

13 Cheung R C. A user-oriented software reliability model. *IEEE Transactions on Software Engineering*, 1980, **SE-6**(2): 118−125

14 Laprie J C. Dependability evaluation of software systems in operation. *IEEE Transactions on Software Engineering*, 1984, **SE-10**(6): 701−714

15 Zheng Z, Trivedi K S, Qiu K, Xia R. Semi-markov models of composite web services for their performance, reliability and bottlenecks. *IEEE Transactions on Services Computing*, 2017, **10**(3): 448−460

16 Zheng Z, Trivedi K S, Wang N, Qiu K. Markov regenerative models of webservers for their user-perceived availability and bottlenecks. *IEEE Transactions on Dependable and Secure Computing*, 2018, 1−1

17 Goyevapopstojanova K, Aditya M P, Kishor T S. Comparison of architecture-based software reliability models. In: Proceedings of the 12th International Symposium on Software Reliability Engineering. New York, USA: IEEE, 2001. 22−31

18 Meedeniya I, Grunske L. An efficient method for architecture-based reliability evaluation for evolving systems with changing parameters. In: Proceedings of the 21st International Symposium on Software Reliability Engineering. New York, USA: IEEE, 2010. 229−238

19 Diehl S. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Berlin: Springer-Verlag, 2007.

20 Merino L, Ghafari M, Anslow C, Nierstrasz O. A systematic literature review of software visualization evaluation. *Journal of Systems and Software*, 2018, **144**: 165−180

21 Mahmood S, Mueller K. An exploded view paradigm to disambiguate scatterplots. *Computers & Graphics*, 2018, **73**: 37−46

22 Archambault D, Purchase H C, Pinaud B. Difference map readability for dynamic graphs. *Lecture Notes in Computer Science*, 2010, **6502**: 50−61

23 Legg P A, Maguire E, Walton S, Chen M. Glyph visualization: a fail-safe design scheme based on Quasi-Hamming distances. *IEEE Computer Graphics & Applications*, 2017, **37**(2): 31−41

24 Zhao Hui-Qun, Sun Jing. An algebraic model of Internetware software architecture. *Scientia Sinica (Informationis)*, 2013, **43**(1): 161−177
(赵会群, 孙晶. 网构软件体系结构代数模型. 中国科学: 信息科学, 2013, **43**(1): 161−177)

25 Zhang Jie, Lu Yang, Liu Guang-Liang. Algebraic approach of software reliability estimation based on architecture analysis. *Systems Engineering and Electronics*, 2015, **37**(11): 2654−2662
(张捷, 陆阳, 刘广亮. 基于结构分析的软件可靠性评估代数方法. 系统工程与电子技术, 2015, **37**(11): 2654−2662)

26 Li B, Liao L, Si J. A technique to evaluate software evolution based on architecture metric. In: Proceedings of the 14th International Conference on Software Engineering Research, Management and Applications. New York, USA: IEEE, 2016. 1−8

27 Wang W L, Wu Y, Chen M H. An architecture-based software reliability model. In: Proceedings of the Pacific Rim International Symposium on Dependable Computing. New York, USA: IEEE, 1999. 143−150

28 Chen H. Analysis and comparison of reliability models based on software architecture. IEEE International Conference of Online Analysis and Computing Science. Chongqing, China, 2016.

29 Hsu C J, Huang C Y. An adaptive reliability analysis using path testing for complex component-based software systems. *IEEE Transactions on Reliability*, 2011, **60**(1): 158−170

**张 捷** 合肥工业大学计算机与信息学院博士研究生. 2009 年获得同济大学电子与信息工程学院硕士学位. 主要研究方向为系统可靠性, 软件可靠性, 可靠性工程. E-mail: zjzj2526@163.com

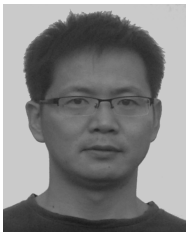
(**ZHANG Jie** Ph. D. candidate at the School of Computer Science and Information Engineering, Hefei University of Technology. He received his master degree from Tongji University in 2009. His research interest covers system reliability, software reliability, and reliability engineering.)

**陆 阳** 合肥工业大学计算机与信息学院教授. 2002 年获得合肥工业大学计算机应用技术专业博士学位. 主要研究方向为分布式控制, 可靠性工程, 物联网工程. 本文通信作者.
E-mail: luyang.hf@126.com

(**LU Yang** Professor at the School of Computer Science and Information Engineering, Hefei University of Technology. He received his Ph. D. degree in computer application technology from Hefei University of Technology in 2002. His research interest covers distributed control, reliability engineering, and Internet of Things engineering. Corresponding author of this paper.)

**张本宏** 合肥工业大学计算机与信息学院副教授. 2010 年获得合肥工业大学计算机应用技术专业博士学位. 主要研究方向为分布式控制, 嵌入式系统, 可靠性工程. E-mail: zhangbh@hfut.edu.cn

(**ZHANG Ben-Hong** Associate professor at the School of Computer Science and Information Engineering, Hefei University of Technology. He received his Ph. D. degree in computer application technology from Hefei University of Technology in 2010. His research interest covers distributed control, embedded system, and reliability engineering.)



**刘广亮** 合肥工业大学计算机与信息学院博士研究生. 主要研究方向为软件可靠性工程与数据挖掘.
E-mail: homecs@126.com

(**LIU Guang-Liang** Ph. D. candidate at the School of Computer Science and Information Engineering, Hefei University of Technology. His research interest covers software reliability engineering and data mining.)