# 构件软件变更风险分析技术*)

毛澄映[1,2] 张金隆[1] 卢炎生[3]
(华中科技大学管理学院 武汉 430074)[1] (江西财经大学软件学院 南昌 330013)[2]
(华中科技大学计算机科学与技术学院 武汉 430074)[3]

**摘 要** 构件软件相较于传统软件系统有更快的演化速度,对其变更进行有效的度量将有利于后期的维护活动。本文分别针对代码可见及不可见两种类型的构件,运用改进的构件依赖图建模,表示构件软件系统。分两步分析构件变更所带来的风险:首先在计算变更比例的基础上度量单个构件的变更风险,再通过将构件依赖图转化成构件依赖树来计算变更的构件集给系统所带来的风险。此外,结合实例系统的分析给出了所提出的变更风险度量的若干性质。

**关键词** 构件软件,变更风险,构件依赖图,构件依赖树

## Change Risk Analysis for Component-based Software

MAO Cheng-Ying[1,2] ZHANG Jin-Long[1] LU Yan-Sheng[3]
(School of Management, Huazhong University of Science and Technology, Wuhan 430074)[1]
(School of Software, Jiangxi University of Finance and Economics, Nanchang 330013)[2]
(College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)

**Abstract** Compared with the traditional software systems, component-based software evolves more rapidly in its life-cycle. The effective measurement plays crucial role in the latter maintenance activities for component-based software. For two types of components, i. e., visible-code component or invisible-code component, the paper analyzes its complex and failure results, then modeling the whole system via the improved component dependency graph. Based on the above model representation, two-step algorithm is proposed to calculate the system's change risk. At first, the change risk of single component is measured through calculating the change ratio. Secondly, the whole system's change risk aroused from some changed components is ranked by conversing component dependency graph to component dependency tree. Additionally, a case study is implemented to valid the presented method and some properties of that metric is also addressed.

**Keywords** Component-based software, Change risk, Component dependency graph, Component dependency tree

## 1 引言

构件软件(component-based software, CBS)通常通过组装、集成的方式实现,且构件作为一个"即插即用"的封装实体极易被替换。该方式虽更易于构件工业化生产,有利于实现构件软件系统的升级与维护[1],但将导致构件软件比常规软件系统的演化速度更快,对其演变进行跟踪、控制与度量是基于构件的软件工程(CBSE)的研究热点之一。

构件软件的高可演变性会给后期维护带来困难。为了便于维护活动的开展及优化部署,对变更或变更后的软件版本进行度量显然是十分必要的。早期研究中代表性的工作有:运用 Markov 过程对构件系统的体系结构进行建模,根据软件系统的执行剖面来计算软件的可靠性,并依此进行测试资源的分配[2];毛晓光等人则从构件软件动态执行场景方面计算构件系统的可靠性[3]。然而,在后期测试等维护活动中,维护者更关注软件系统中尚存在哪些风险。基于风险的测试就是按照该思路进行的。

在构件软件风险分析方面,Yacoub 等人运用场景模拟实现动态度量,进而完成构件软件的可靠性风险分析[4];Goseva-Popstojanova 等人在标准 UML 表示的基础上形成离散时间 Markov 链,进而依此评估构件软件的风险[5]。如同提及的可靠性分析一样,以上风险分析技术仅关注于某一特定构件软件版本的风险,没有在时序的指导下考虑软件系统风险演进特征。本文则按照系统演化历程,专注由于构件变更导致后续版本相对于先前版本可能存在的变更风险。

## 2 构件系统及其变更

### 2.1 构件与构件软件

直至目前,软件工程界尚没有一个关于构件的统一规范定义。但综合分析已有的几种典型定义[6,7],不难发现构件具备以下基本特征:构件是近乎独立的、可替换的、满足一定功能的模块;对所处的环境或上下文有一定的依赖关系;构件通常符合一组接口标准,构件之间通过接口进行通讯。面向对象技术为构件的实现提供了基本技术支持,在现阶段一般情况下,构件由 Java、C++/C# 等语言实现。

构件软件是多个构件(可以是异质的)通过连接件(con-

*)国家自然科学基金项目(70571025)、教育部高等学校博士点基金(20060487005)、湖北省自然科学基金(2005ABA266)资助。**毛澄映** 博士后,研究方向为软件测试技术及过程管理、数据挖掘等;**张金隆** 博士,教授,博导,研究方向为现代管理理论与方法、信息资源管理、风险决策等;**卢炎生** 教授,博导,研究方向为软件工程、数据挖掘等。

nector)进行组装而形成的系统，是一种松耦合结构。构件软件潜在的失效风险高度依赖于单个构件和连接件的失效风险。本文主要讨论由于构件的变更所引起的整个软件系统不可靠的风险。研究者在对构件软件进行体系结构分析、测试或可靠性评估过程中，利用多种方式对构件软件系统进行了建模表示。例如，文[8]对 UML 进行形式化的扩展，用标记的转换系统(labeled transition systems，LTS)表示；Wu 等人则用构件交互图 CIG(component interaction graph)描述构件间的交互场景[6]；还有依赖矩阵[9]、构件迁移概率图[3]等表示方式。但最被普遍认同和广泛使用的还是构件依赖图 CDG(component dependency graph)模型，这里我们重新定义构件依赖图以便进行变更风险分析。

**定义 1**(构件依赖图) 构件软件可用一个四元组形式的依赖图 $CDG=(C,T,n_s,n_t)$ 表示，其中 $C$ 为表示构件的顶点集，$T$ 是表示连接件的边集，$n_s$ 和 $n_t$ 分别表示起始、终止节点。构件顶点集 $C$ 和连接件边集 $T$ 可进一步定义如下：

$C=\{\langle c_i,cpx_i,rf_i\rangle\}(1\leqslant i\leqslant|C|)$，其中 $c_i$ 为第 $i$ 个构件的标识符，$cpx_i$ 为构件 $c_i$ 的复杂性，$rf_i$ 为构件 $c_i$ 的失效风险(即若构件 $c_i$ 失效所带来的后果的严重性程度)。

$T=\{\langle t_{ij},p_{ij}\rangle\}(1\leqslant i,j\leqslant|C|)$，其中 $t_{ij}$ 为由构件 $c_i$ 连接到构件 $c_j$ 的连接件的标识符，$p_{ij}$ 为连接件 $t_{ij}$ 被执行的概率，一般有

$$\sum_{j=1}^{k}p_{ij}=1\ (k\text{ 为 }c_i\text{ 所有出向连接件边的总数}) \tag{1}$$

2.1.1 构件复杂性度量

从代码可见性的角度看，构件可分为内部构件(in-house component)和外部构件(external component)。内部构件的代码对构件软件开发者(或维护者)来讲是可见的，外部构件(如第三方 COTS)的代码往往无法获得。这里分两种情况讨论构件的复杂性。

对于代码可见的情形，我们认为程序控制流图的圈复杂性、代码行数以及包含的库文件数目均会对构件的复杂性产生影响。对于构件 $c_i$，其复杂性的 3 个分量(圈复杂性 $cc_i$、行复杂性 $lc_i$ 和库包含复杂性 $lib_i$)按如下公式计算：

$$cc_i=\frac{McCabe(c_i)}{\max\limits_{1\leqslant j\leqslant|C|}\{McCabe(c_j)\}}=\frac{e_i-n_i+p_i}{\max\limits_{1\leqslant j\leqslant|C|}\{e_j-n_j+p_j\}} \tag{2}$$

$$lc_i=\frac{H(c_i)}{\max\limits_{1\leqslant j\leqslant|C|}\{H(c_j)\}}=\frac{opr_i\cdot\log_2 opr_i+opn_i\cdot\log_2 opn_i}{\max\limits_{1\leqslant j\leqslant|C|}\{opr_j\cdot\log_2 opr_j+opn_j\cdot\log_2 opn_j\}} \tag{3}$$

$$lib_i=\frac{incld(c_i)}{\max\limits_{1\leqslant j\leqslant|C|}\{incld(c_j)\}} \tag{4}$$

其中 $McCabe(c_i)$ 是构件 $c_i$ 的代码的环形复杂度，$H(c_i)$ 为其 Halstead 度量，$incld(c_i)$ 则表示构件 $c_i$ 所包含的库文件数目。最后，加权综合三者得到构件的复杂性度量如下：

$$cpx_i=w_1\cdot cc_i+w_2\cdot lc_i+w_3\cdot lib_i(1\leqslant i\leqslant|C|) \tag{5}$$

这里取 $w_1=0.6$，$w_2=0.3$ 且 $w_3=0.1$。

对于代码不可见的外部构件，构件开发者一般运用 XML 等标准、易于交换的文件格式提供构件的梗概信息(元数据)[10]，典型的有代码总行数、方法数目及方法间的调用关系等。构件使用者可以利用这些信息对构件的复杂性进行粗略的估计。构件中的方法及其调用关系可用文[11]定义的方法调用图 MCG 表示。对于构件 $c_i$，初步定义其复杂性如下：

$$cpx_i=\frac{w_1\cdot l(c_i)}{\max\limits_{1\leqslant j\leqslant|C|}\{l(c_j)\}}+\frac{w_2\cdot|V_{MCG}(c_i)|+|E_{MCG}(c_i)|}{\max\limits_{1\leqslant j\leqslant|C|}\{|V_{MCG}(c_j)|+|E_{MCG}(c_j)|\}} \tag{6}$$

其中 $l(c_i)$ 为构件 $c_i$ 的总代码行数；$|V_{MCG}(c_i)|$ 和 $|E_{MCG}(c_i)|$ 分别是 $c_i$ 的调用图中的方法数和调用次数；$w_1$ 和 $w_2$ 为权重系数，这里取 $w_1=w_2=0.5$。

2.1.2 构件失效风险度量

度量一个构件的失效风险就是分析当该构件一旦无法正常执行时给整个软件系统所带来后果的严重性程度。这里，同样分代码可见和代码不可见两种情况分析。当构件的代码可见时，运用变异分析(mutant analysis)的技术来实现，即在构件中植入一些常见错误，再执行整个系统，看产生的后果情况，以确定失效风险系数。我们采用如表 1 所示的失效严重性分类[12]进行构件软件运行结果评估，判断每个构件的失效所导致的风险。

表 1 构件失效后果严重性程度分类

| 序号 | 严重性程度 | 系数 | 字段信息 |
|---|---|---|---|
| 1 | 轻度 | 0.25 | 产生错误的计算结果，不会造成什么损失 |
| 2 | 中度 | 0.5 | 会造成一定的经济损失，但系统可恢复 |
| 3 | 严重 | 0.75 | 给软件使用者带来诸多麻烦，导致较大的损失，但重要数据可恢复 |
| 4 | 灾难性 | 0.95 | 整个系统崩溃，产生巨大经济损失或造成人员伤亡 |

对于代码不可见的构件，可从如下两个方面分析它的失效严重性：(1)组织一批经验丰富的软件设计师对构件的规格说明文档进行评审，再运用模糊综合评判的方法给出一个失效严重性系数；(2)一般来讲，一个构件如果是构件系统中信息交换的枢纽，则它的失效造成的后果往往偏于严重。可以用构件在构件依赖图中扇入(fan-in)、扇出(fan-out)连接数之和作为衡量依据，也可用信息检索领域中评价网页重要性的 PageRank 算法[13]实施度量。无论用何种方式度量构件在系统中的重要性，通常如果度量值越高则它失效造成的后果也就越严重。

2.1.3 连接件执行概率度量

对于连接件 $t_{ij}(1\leqslant i,j\leqslant|C|)$，其执行概率实质上是构件 $c_i$ 执行完毕后转换执行构件 $c_j$ 的转移概率。计算该概率时一般根据构件软件所有可能的执行场景进行统计得出。这里采用文[14]中的计算方法实施计算。

$$p_{ij}=\sum_{k=1}^{|S|}P(s_k)\cdot\left[\frac{|Ineract(c_i,c_j)|}{|Interact(c_i,c_l)|_{1\leqslant l\leqslant|C|\wedge l\neq i}}\right]\quad(c_i,c_j,c_l\text{ in }s_k) \tag{7}$$

其中 $S$ 为软件系统的场景集，$P(s_k)$ 为场景 $s_k$ 的执行概率，$|Interact(c_i,c_j)|$ 表示在某一特定场景(如 $s_k$)中构件 $c_i$ 交互构件 $c_j$ 的次数。需要指出的是，$|Interact(c_i,c_j)|$ 往往不等于 $|Interact(c_j,c_i)|$。

### 2.2 构件软件的变更

在构件软件演化过程中，为了修正错误、提升性能或增强功能等，需要对系统做修改。在一般情况下，软件系统的修改主要体现在如下 3 个方面：(1)构件中的部分代码做了修改；(2)增加构件；(3)删除构件。以上 3 类变更方式中，第 1 类是最为主要的变更途径，我们目前的研究主要考虑此类情形。

为了便于上下文表达，这里给出两个与变更相关的定义。

**定义 2**(变更语句集) 对于构件 $c$，其中被修改(含增加

或删除)的语句的集合称为构件 $c$ 的变更语句集,记作 $S_{sm}(c)=\{s_1,s_2,\cdots,s_m\}$(其中 $m$ 为 $c$ 中变更语句总数)。

**定义 3**(变更方法集) 对于构件 $c$,其中被修改(含增加或删除)的方法的集合称为构件 $c$ 的变更方法集,记作 $S_{mth}(c)=\{m_1,m_2,\cdots,m_n\}$(其中 $n$ 为 $c$ 中变更方法总数)。

## 3 变更风险分析

当构件软件维护人员为了满足新的需求对系统中某些构件做修改后,非常有必要考虑它给系统中其他构件带来哪些影响、给系统的可靠性造成多大的风险等。本文提出的变更风险分析主要是在变更完成之后、回归测试之前进行,用以评估由于构件的修改给系统带来多大额外的不可靠性风险。通过变更风险分析,将有利于回归测试工作的部署与展开。例如,对人力、资金和测试进度的计划,以及指导回归测试用例的选择等。

变更风险分析主要分两步进行:第 1 步,对每一修改的构件,分析变更对自身的可靠性带来的风险;第 2 步,在单个构件变更风险分析的基础上,计算整个系统的变更风险。

### 3.1 单个构件变更风险

在分析单个构件的变更风险之前,需要计算构件中变更部分所占的比例。一般来讲,比例越大,则导致潜在的风险也越大。反之亦然。这里同样分构件代码可见与不可见两种情形考虑。

情形 1:构件的代码可见。根据变更语句集,运用静态切片技术[15]计算出变更语句所能影响到的语句,进而计算出变更比例。首先给出定义变量和使用变量的定义:

**定义 4**(定义/使用变量) 对任一语句 $s$,若 $s$ 中的变量 $v$ 在该语句中被重新指定值,则称 $v$ 是 $s$ 的定义变量,记作 $def(s,v)$;类似地,若该语句引用了 $v$ 的值,则称 $v$ 是 $s$ 的使用变量,记作 $use(s,v)$。

对于变更构件 $c$,可按如下算法计算其变更比例 $ratio(c)$。其中,$forward_slice(s_i,v_j)$是指以$(s_i,v_j)$为切片准则计算的前向静态切片集,$backward_slice(s_i,v_k)$是指按$(s_i,v_k)$计算得到的后向静态切片集。

**算法 1** Calculate Ratio(c)

```
Input:S(c),构件 c 的语句集;
      S_sm(c),构件 c 的变更语句集。
Output:ratio(c),构件 c 的变更比例。
{
  S_tmp=∅; //临时语句集
  for each s_i∈S_sm(c)
  {
    for each def(s_i,v_j) in s_i
      S_tmp=S_tmp∪forward_slice(s_i,v_j);
    for each use(s_i,v_k) in s_i
      S_tmp=S_tmp∪backward_slice(s_i,v_k);
  }
  return |S_tmp|/|S|;
}
```

情形 2:构件代码不可见。在此种情形下,需要根据构件开发者提供的元数据信息构建方法调用图 MCG[11]。再对 $\forall m_i\in S_{sm}(c)(1\leqslant i\leqslant n)$,在 MCG 中以 $m_i$ 为起点进行前、后遍历查找,标记出受 $m_i$ 影响的方法集,记作 $affected(m_i)$。设 MCG 图中方法顶点的集合为 $M$,则可计算构件 $c$ 的变更比例如下:

$$ratio(c)=\left|\bigcup_{i=1}^{n}affected(m_i)\right|/|M| \tag{7}$$

最后,对于 $\forall c_i\in C$,其变更风险 $cr_i$ 可按式(8)计算。显然,没有变更的构件的变更风险值为 0。

$$cr_i=ratio(c_i)\cdot cpx_i\cdot rf_i \tag{8}$$

### 3.2 构件系统的变更风险

计算整个构件软件系统的变更风险实质上就是度量部分修改了的构件对整个系统的可靠性所带来的影响。为了便于分析,需将构件依赖图 CDG 变换成一棵生成树,称之为构件依赖树。

**定义 5**(构件依赖树) 对于构件依赖图 $CDG=(C,T,n_s,n_t)$,可将其转化成如下一棵生成树:

$CDT=(N,E)$,其中 $N=\{\langle c_i,cr_i\rangle\}$为表示构件的节点集,$E=\{\langle e_{ij},p_{ij}\rangle\}$表示构件执行所依赖的有向边集,显然 $1\leqslant i,j\leqslant|C|$。

根据以下步骤实现从构件依赖图 CDG 向构件依赖树 CDT 的转换:

第 1 步,将 CDG 中的起始节点 $n_s$ 作为 CDT 的根结点,同时记根节点为当前节点。

第 2 步,对任一当前节点,广度优先搜索 CDG,将当前节点每一条出向边作为 CDT 中对应节点的一条边,该边目标节点作为当前节点的儿子节点。当满足如下两个条件之一时,CDT 中的节点终止扩展:(1)若当前节点是终止节点 $n_t$;(2)若当前节点(设它处于 CDT 的第 $i(i>1)$层)在 CDT 的 1 至 $i-1$ 层出现过。

第 3 步,迭代进行第 2 步的遍历,直到所有节点都不能扩展为止。

在构件软件系统中,构件间通过对执行信息流的修改进行交互。一般情况下,某一构件的修改将会影响到它所在的执行场景的信息流,进而对在它之后执行的构件产生副作用。对于变更构件来讲,即便变更没有引起某一场景下的自身失效,但仍存在引起其它未修改的构件运行失效的可能。一般而言,变更比例越大或变更构件越复杂,这种潜在的风险也就越大。

在构件依赖树 CDT 的基础上,按算法 2 对变更构件的风险及对其它构件的传播进行赋值,该算法中最为重要的是传播因子 $\theta_p$ 的确定,这里初步选定按公式 $\theta_p=1/(1+\alpha h)$计算。其中,$h$ 为受影响构件与变更构件在 CDT 树中层次的差值,$\alpha$ 为调节系数。

**算法 2** Assign cr

```
Input:CDT, 构件依赖树;
      C, 系统的构件集;
      C_chg, 变更构件子集.
Output: CDT', 扩充了变更风险的构件依赖树.
{
  for each node c_k in CDT
    cr_k=0;
  for each c_i∈C_chg
    在 CDT 对应的节点上填充其 cr_i 值;
  for each c_i∈C_chg in CDT
    for each offspring c_j of c_i in CDT
    {
      计算 c_i 与 c_j 的层次差值,记为 h_ij;
      cr_new=ratio(c_i)·cpx_i·rf_j/(1+αh_ij);
      if(cr_new>cr_j)
        cr_j=cr_new;
    }
  return 指定变更风险值之后的依赖树 CDT';
}
```

值得注意的是,树中与终止节点 $n_t$ 对应的节点并不参与计算,其变更风险值以 0 计。同样地,CDT 树的其实节点的变更风险值也记为 0。得到完整的 CDT 树后,便可按算法 3 计算出整个系统的变更风险 CR。这里,将从根节点 $n_s$ 到任一叶节点 $c_k$(有可能为 $n_t$)的路径记为 $path=\langle n_s,\cdots,c_i,c_j,\cdots,c_k\rangle$。

**算法 3** Calculate CR

```
Input: CDT, 构件依赖树。
Output: CR, 软件系统的变更风险。
{
    R_sum=0;
    for each leaf-node c_k in CDT
    {
        构建一条从树根至 c_k 的路径 path=⟨n_s,…,c_i,c_j,…,c_k⟩;
        R_tmp=1;
        for each node c_i in path and c_j is the next node of it
            R_tmp=R_tmp·(1−cr_i)·p_ij; //当计算到 c_k 时，转移概率取值为 1.
        R_sum=R_sum+R_tmp;
    }
    CR=1−R_sum;
    return CR;
}
```

## 4 实例分析

为了将上述计算过程表述清楚，有必要通过实例进行说明。本文借鉴文[14]所用的示例构件体系结构。一方面由于无法获取构件的相关信息，另一方面为简洁起见，我们依据文[14]提供的部分信息对各构件的复杂性和失效后果进行了人为指定(省略了依据第2节的计算步骤)。最终形成用于实例分析的构件依赖图，如图1所示。

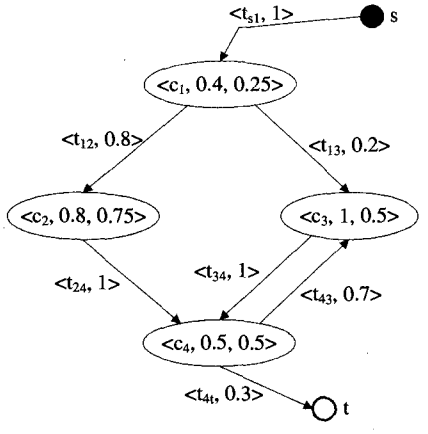


图1 一个实例构件依赖图

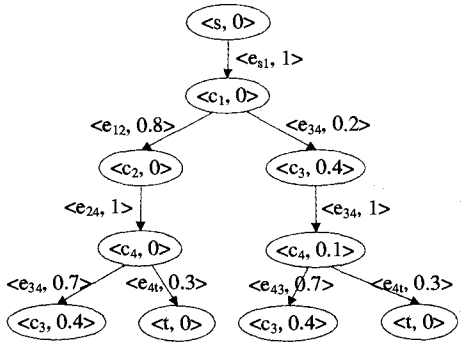


图2 依据CDG生成的构件依赖树

依据转换方法，可将构件依赖图CDG转化为如图2所示的构件依赖树CDT。假定构件 $c_3$ 作了修改且变更比例为 $ratio(c_3)=0,8$，这样 $c_3$ 的变更风险 $cr_3=0.8\times1\times0.5=0.4$；接下来可按算法2为CDT中各节点指定变更风险(结果如图2所示)。图中受 $c_3$ 影响的构件为 $c_4$(这里取 $\alpha=1$)。对于 $c_4$ 而言，$h_{34}=1$，则 $\theta_p=0.5$；进而有 $cr_4=ratio(c_3)\cdot cpx_3\cdot\theta_p\cdot cpx_4\cdot rf_4=0.1$。在CDT中作为叶节点的构件 $c_3$，其变更风险显然也为 $cr_3=0.4$。除构件 $c_3$、$c_4$ 外，其余构件的变更风险均为0。

那么，对于 $path_1=\langle s,c_1,c_2,c_4,c_3\rangle$，其 $R_{tmp}=(1-0)\times1\times(1-0)\times0.8\times(1-0)\times1\times(1-0)\times0.7\times(1-0.4)=0.336$。其他3条路径可按同样的方法计算，进而得到整个系统的变更风险为 $CR=1-(0.336+0.24+0.0454+0.0324)=0.3462$。

进一步地，如果同时存在多个变更构件，仍可按照同样的方法计算对整个系统造成的风险。例如，如果除了构件 $c_3$ 发生了变更外($ratio(c_3)=0,8$)，构件 $c_2$ 也发生了变更且变更比例为 $ratio(c_2)=0.4$，那么根据算法2和3可计算出整个系统的变更风险 $CR=1-(0.2451+0.1751+0.0454+0.0324)=0.502$。

通过实例分析，不难得到系统变更风险的一些性质：(1)$0\leqslant CR\leqslant1$；(2)从系统执行场景上看，经常被执行到的构件如果发生变更，则对系统的可靠性带来更大风险；(3)构件变更比例与系统的变更风险成正比；(4)发生变更的构件的复杂性、失效后果也与系统的变更风险成正比。

**结束语** 软构件技术具备实现构架与逻辑分离、组织大规模协作开发、降低系统开发代价等优点，已被实践证明是一种有效的软件开发模式。然而，构件软件较频繁的演变往往给后期维护带来巨大压力。系统每一次变更后有必要对变更所带来的风险做出准确的评估，以便进行回归测试活动的优化部署。本文在运用重新定义的构件依赖图对构件软件进行表示的基础上，分别给出了变更构件和整个构件系统的变更风险计算方法，为评估构件变更所造成的潜在后果提供了一个初步的定量分析手段。诚然，目前所完成的仅是一个计算构件系统变更风险的初步模型，关于重要参数因子的敏感性分析、大型构件系统的实证等工作是下一步研究的重点。

## 参 考 文 献


1 杨芙清，梅宏，李克勤. 软件复用与软件构件技术 [J]. 电子学报，1999，27(2)：68～75
2 Lo J H, Kuo S Y, Lyu M R, et al. Optimal Resource Allocation and Reliability Analysis for Component-Based Software Applications [C]. In: Proc of COMPSAC'02, 2002. 7～12
3 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
4 Yacoub S, Ammar H H. A Methodology for Architectural-Level Reliability Risk Analysis [J]. IEEE Transaction on Software Engineering, 2002, 28(6): 529～547
5 Goseva-Popstojanova K, Hassan A, Guedem A, et al. Architectural-Level Risk Analysis using UML [J]. IEEE Transaction on Software Engineering, 2003, 29(10): 946～959
6 Wu Y, Pan D, Chen M H. Techniques for Testing Component-based Software [C]. In: Proc of ICECCS'01, 2001. 222～232
7 景涛，白成刚，胡庆培，等. 构件软件的测试问题综述[J]. 计算机工程与应用，2002(24)：1～6
8 Liu W, Dasiewicz P. Component Interaction Testing Using Model-Checking [C]. In: Proc. of Canadian Conference on Electrical and Computer Engineering, 2001. 41～46
9 Li B, Zhou Y, Wang Y, et al. Matrix-based Component Dependency Representation and Its applications in Software Quality Assurance [J]. ACM SIGPLAN Notices, 2005, 40(11): 29～36
10 Orso A, Harrold M J, Rosenblum D. Component Metadata for Software Engineering Tasks [C]. In: Proc. of Int'l Workshop on Engineering Distributed Objects (EDO), 2000. 129～144
11 毛澄映，卢炎生. 构件软件回归测试用例选择策略[J]. 计算机研究与发展，2006，43(10)：1767～1774
12 Procedures for Performing Failure Mode Effects and Criticality Analysis [R]. US MIL_STD_1629A/Notice 2, 1984
13 Page L, Brin S, Motwani R, et al. The PageRank Citation Ranking: Bringing Order to the Web: [Technical Report]. Computer Science Department, Stanford University, 1999
14 Yacoub S, Cukic B, Ammar H H. A Scenario-based Reliability Analysis Approach for Component-based Software [J]. IEEE Transaction on Reliability, 2004, 53(4): 465～480
15 Tip F. A Survey of Program Slicing Techniques [J]. Programming Languages, 1995, 3(3): 121～189