基于多维属性的构件化软件演化相似性度量方法研究

钟林辉 李俊杰 夏 鲸 薛良波
(江西师范大学计算机信息工程学院 南昌 330022)

摘 要 对不同软件进行演化相似性度量和比较能帮助软件维护人员理解软件演化及预测其演化趋势。然而,传统的研究大多度量单一软件演化属性的变化,虽然有些也涉及到多维演化属性,但并没有与软件的演化相似性相联系;同时亦缺乏在较高抽象层次度量软件演化相似性的有效途径。为此,以构件为基本单位,提出了一种基于多维演化属性的构件化软件演化相似性度量模型。即通过在原子构件层和系统(复合构件)层定义演化属性,进而度量原子构件之间以及系统(复合构件)之间的演化相似性。初步的实验表明,在原型工具的支持下该方法能辅助软件维护人员进行软件演化相似性的判断。

关键词 构件化软件,软件演化,多维演化属性,演化相似性度量
中图法分类号 TP311 **文献标识码** A

Research on Evolution Similarity Measurement of Component-based Software Based on Multi-dimensional Evolution Properties

ZHONG Lin-hui LI Jun-jie XIA Jin XUE Liang-bo
(College of Computer and Information Engineering, Jiangxi Normal University, Nanchang 330022, China)

Abstract By measuring and comparing the evolution similarity for the different component-based software, the software developer can understand the software evolution and predict its evolution tendency. However, most traditional researches focus on the change of a single software evolution during the software evolution process. Although some of them are involved with multi-dimensional evolution properties, they are not related to the software evolution similarity and lack the ability to measure the evolution similarity at a higher level. This paper proposed an evolution similarity measure model for component-based software based on multi-dimensional evolution properties, which can measure evolution similarity for different atomic component or system (compose component) by selected evolution attributes. The experiments show the method can aid the software maintainer to judge the evolution similarity by the prototype support.

Keywords Component-based software, Software evolution, Multi-dimensional evolution properties, Evolution similarity measurement

1 前言

演化性是软件的固有属性,软件演化是软件不断更新变化的过程,是软件的本质特征之一[19]。早在 20 世纪 70 年代,Lehman 等人通过研究软件的变化历史信息,发现了软件具有"持续变化"等演化规律[16]。

一般地,传统的软件演化信息度量技术主要针对的是单一软件演化属性在软件演化过程中的变化,例如不同版本之间文件个数的变化、源代码行数的变化、分析或设计模型差异性等。虽然也有研究考虑在软件多维属性上进行软件演化度量(例如文献[13]中将类之间的耦合性度量、模块复杂性度量纳入到一个统一的度量公式中),但是这些软件演化度量研究的目的主要是研究单个软件系统规模的演化趋势、软件架构等演化稳定性及同软件维护开销的关系等,并没有研究如何度量不同软件系统(特别是构件化软件系统)之间的演化相似性。

为了解决上述问题,本文在支持构件的演化模型[21]以及基于单维演化属性的软件演化相似性度量模型[22]的基础上,进一步提出了一种基于多维演化属性的演化相似性度量模型,通过刻画构件化软件的多维演化属性在早期版本和近期版本等多种情况下的变化,并利用相似性计算公式度量构件化软件之间在多维演化属性上的演化相似性。本文第 2 节介绍基于度量的软件演化的相关研究;第 3 节提出了构件化软件演化分析框架;第 4 节介绍基于矩阵的演化数据模型及相似性度量公式;第 5 节提出了构件化软件的演化相似性度量模型,并给出了具体的计算步骤;第 6 节则通过对开源软件的度量,分析其在多维属性上的演化相似性;最后是系统原型及总结全文。

2 相关研究

与传统软件度量着眼于软件单个版本的特征(例如复杂性、文件大小等)[2]不同,软件演化度量需要研究软件系统在多个版本上的变化[1]。早期软件演化度量主要在源程序代码

本文受国家自然科学基金项目(61262015,61462040),江西省自然科学基金(20142BAB207027)资助。
钟林辉(1974—),男,博士,副教授,主要研究方向为软件体系结构、构件化软件、软件维护与软件演化,E-mail:shiningto@jxnu.edu.cn;李俊杰(1992—),男,硕士生,主要研究方向为软件演化度量;夏 鲸(1991—),男,硕士生,主要研究方向为软件演化度量;薛良波(1991—),男,硕士生,主要研究方向为软件重构。

行、文件大小等较低层次度量不同版本之间的变化，例如 Lehman 等通过度量软件模块的变化数目，发现了软件演化具有线性变化的特点[3]。Zurieta 通过度量软件源代码行、文件目录以及软件大小，得出了开源软件的变化不一定比传统软件的变化要快的结论[5]。

近年来，研究者试图从软件模型、软件结构或者软件体系结构等较高层次研究软件演化的度量。1)在模型层次：Hismo 模型从元模型的角度对软件演化过程中的概念进行了建模，并借助可视化工具展示软件多个属性的变化程度[6]；Christoph 等利用本体描述语言作为软件资产的数据交换格式，扩展了本体查询语言 SPARQL，以实现对软件演化的可视化、软件度量和代码坏味的检查[10]。2)在软件结构层次：通过度量可执行语句和非可执行语句的数目、控制流图中节点个数和执行路径条数的变化，研究了软件结构变化同软件缺陷之间的关系[7]；Pamela 等人通过分析各个程序版本的调用图和协作图，以及基于错误修改和变化请求的开发者间协作图，来预测软件维护的代价[8]；Gregorio Robles 则提出使用函数的语义信息，而不是度量文件大小或者代码行数来度量软件演化[11]。3)软件体系结构层次：文献[14]提出了基于图核(Graph Kernel)理论的、基于结构特征的软件体系结构距离度量方法。即将软件体系结构表示为标签图(Label Graph)，将采用随机行走算法获得的一组路径作为软件体系结构的特征向量，进而使用图核理论度量不同软件体系结构之间的差异性。文献[15]提出了软件体系结构变化公式 a2a，用于度量软件体系结构的差异性。

同时，有的研究者通过考察多个软件演化属性来度量软件的演化。例如 Gonzalez-Barahona 等人通过度量 Debian 系统在 1998－2007 年之间所有版本的源代码演化情况，包括 SLOC、包的个数、包与包之间的依赖、变化的包个数和没有变化的包个数等多个演化属性，来预测 Debian 稳定版本规模的演化趋势[4]；Sangwan 等以 Hibernate 为实验对象，提出一种综合的、基于度量的多维方法来检查发生在软件架构稳定性和体系结构复杂性上的演化变化[9]；D'Ambros 等提出了一个交互式的分析方法，供用户查看软件模块间依赖关系、软件代码大小的演化，以及基于模型标注技术的协同分析能力[12]；Emanue 等人为了度量开源系统的开发质量，提出"模块化系数(Modularity Index)"的概念，将面向对象程序中对类的质量、包的质量以及软件体系结构的度量统一到一个度量公式中[13]。

上述软件演化度量的研究主要度量软件在演化过程中单一个或者多个属性上的变化。尚未有研究报道对多个不同软件系统之间在软件演化相似性方面的度量。

3 构件化软件演化分析框架

为了有效地支持构件化软件的开发和演化，在我们的早期研究中提出了扩充构件描述语言(xCDL)支持基于构件的系统组装与演化的策略[21]，不仅可以有效地支持基于构件的系统构造定义，而且可以支持系统的演化以及系统的部署。其中的关键技术包括扩充了构件描述语言以支持对构件及软件体系结构本身的演化的表示和跟踪，通过基于构件的软件配置管理模型 CBSCM 实现对构件化软件演化数据的存储[20]。基于上述思想和关键技术，本文进一步提出了以软件体系结构为中心的构件化软件演化分析框架，如图 1 所示。

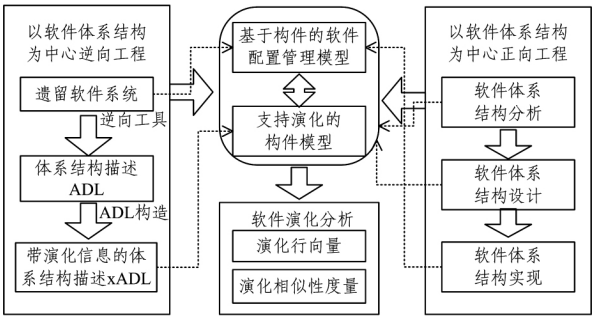

图 1 构件化软件演化分析框架

整个框架分为 3 个部分，包括以软件体系结构为中心的正向工程、以软件体系结构为中心的逆向工程以及软件演化分析过程。其中，以软件体系结构为中心的正向工程所产生的软件制品(例如每个版本的软件体系结构描述、软件体系结构的实现等)均存储在基于构件的软件配置管理系统中；而以软件体系结构为中心的逆向工程通过现有的逆向工程工具，对遗留系统的演化历史进行软件体系结构层次的重建，构造出带版本信息的软件体系结构描述，并将其存储在基于构件的软件配置管理系统中。软件演化分析则通过提取基于构件的软件配置管理系统中存储的带版本信息的软件体系结构描述，度量相邻版本之间的变化性，并实现对不同软件系统的演化相似性度量。

4 基于矩阵的软件演化数据模型

4.1 相关定义

定义 1 演化属性矩阵 EPM(Evolution Property Matrix-EPM)：一个 $m\times n$ 的矩阵，其中 m 表示版本的个数($m>1$)，n 表示演化属性的个数($n>0$)，矩阵中元素 EPM_{ij} ($1\leqslant i\leqslant m$，$1\leqslant j\leqslant n$)表示演化属性在某个版本下的值。例如图 2(a)所示的矩阵中，$\{v_1,v_2,v_3,v_4,v_5,v_6\}$ 是版本集合，$\{EP_A,EP_B,EP_C\}$ 是属性集，矩阵的值表示当前版本对应的属性值。

定义 2 演化属性变化矩阵 EPCM(Evolution Property Change Matrix-EPCM)：一个 $(m-1)\times n$ 的矩阵，表示软件版本与演化属性变化值的对应关系。$(m-1)$ 是版本变化的个数($m>1$)，n 表示演化属性的个数($n>0$)。$EPCM_{ij}=EPM_{ij}\ \delta EPM_{(i+1)j}$，运算符 δ 计算从 EPM_{ij} 到 $EPM_{(i+1)j}$ 的变化值，具体实现跟演化属性有关。例如演化属性是文件大小时，$EPCM_{ij}=|EPM_{ij}-EPM_{(i+1)j}|$；另外，有的演化属性变化值不一定需要显式地通过计算演化属性值得到，例如软件体系结构的演化变化属性值可以通过计算两个版本的图编辑距离得到。图 2(b)所示的是通过图 2(a)计算得到的演化属性变化矩阵。

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array}\begin{array}{c} \begin{array}{ccc} EP_A & EP_B & EP_C \end{array} \\ \begin{bmatrix} 5 & 4 & 8 \\ 2 & 6 & 7 \\ 4 & 9 & 10 \\ 20 & 5 & 8 \\ 4 & 2 & 9 \\ 3 & 8 & 5 \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ v_2\to v_1 \\ v_3\to v_2 \\ v_4\to v_3 \\ v_5\to v_4 \\ v_6\to v_5 \end{array}\begin{array}{c} \begin{array}{ccc} EP_A & EP_B & EP_C \end{array} \\ \begin{bmatrix} 3 & 2 & 1 \\ 2 & 3 & 3 \\ 16 & 4 & 2 \\ 16 & 3 & 1 \\ 1 & 6 & 4 \end{bmatrix} \end{array}$$

(a) (b)

图 2 演化属性及演化属性变化矩阵的例子

定义 3 演化行向量 ESRV (Evolution Similar Row Vector-ESRV)：表示软件系统的演化属性集合 $\{a_1,a_2,\cdots,a_n\}$ 在一定时间段内的变化度量。$ESRV=f(EACM)$，f 是对

$EACM$ 列的一个映射。本文将 f 定义为多维演化属性在较新版本下的映射和在早期版本的映射 2 种情况。同时，根据处理对象（原子构件、系统或者复合构件）的不同，区分为计算原子构件近期演化行向量的 $CLEP_{\xrightarrow{a_1 a_2 \cdots a_n}}$、系统（复合构件）近期演化行向量的 $BLEP_{\xrightarrow{a_1 a_2 \cdots a_n}}$、原子构件早期演化行向量的 $CEEP_{\xrightarrow{a_1 a_2 \cdots a_n}}$ 以及系统（复合构件）早期演化行向量的 4 种情况，具体见 5.2 节和 5.3 节。

4.2 相似性度量方法

4.2.1 度量公式

演化相似性度量指的是针对给定的演化属性（例如系统版本之间的文件数目变化、软件体系结构变化等），度量其在给定时间段内的变化相似性。演化相似性度量公式为：

$$SIM_{\xrightarrow{p_1,p_2,\cdots,p_n}}(ESRV_{S_1},ESRV_{S_2}) = \frac{ESRV_{S_1} \cdot ESRV_{S_2}}{\| ESRV_{S_1} \| \cdot \| ESRV_{S_2} \|} = \frac{\sum_{i=1}^{n}(w_i)^2 * a_i * b_i}{\sqrt{\sum_{i=1}^{n}(w_i * a_i)^2} * \sqrt{\sum_{i=1}^{n}(w_i * b_i)^2}}$$

其中，$ESRV_{S_1}$，$ESRV_{S_2}$ 分别表示多维演化属性在软件 S_1 和软件 S_2 的多个版本下的变化行向量；即系统 S_1 对应的多维演化属性变化行向量为：$\langle w_1 * a_1, w_2 * a_2, \cdots, w_n * a_n \rangle$，系统 S_2 对应的多维演化属性变化行向量为：$\langle w_1 * b_1, w_2 * b_2, \cdots, w_n * b_n \rangle$；其中，$a_k$ 和 b_k 表示属性的变化值，w_k（$1 \leqslant k \leqslant n$）表示权重，$\sum_{k=1}^{n} w_k = 1$，本文如不特殊说明时 $w_1 = w_2 = \cdots = w_n$。显然，演化相似性度量的值是位于 0～1 之间的，数值越大表示系统之间在多维属性上的演化相似度越大；反之，则相似度越小。

4.2.2 虚拟版本插值算法

在进行相似性比较时，所选取的演化属性矩阵的版本个数可能不相同，需要将其进行升维或者降维处理，使之保持版本的个数一致。本文采用升维处理方式，在尽可能保持版本时间变化趋势的前提下插入新的版本（即虚拟版本）。算法 1 是用类 Pascal 语言描述的算法。

算法 1 虚拟版本插值算法

Input：给定类型为日期型 Date 的两个数组 A 和 B，数组的长度分别为 m，n，即 m＝A. length，n＝B. length。假设 m＜n，满足 ∀（i：1≤i≤m－1：A[i]＜A[i＋1]）and ∀（j：1≤j≤n－1：B[j]＜B[j＋1]）。

Output：输出数组 A 和 B，满足：n＝A. length，A. length＝B. length，∀（i：1≤i≤n－1：A[i]＜A[i＋1]∧B[i]＜A[i＋1]），F（A[n]，B[n]）＝MIN（k：1≤k≤P_m^{n+1}：F（Map_k（A[m]，A[n]），B[n]），其中 F 表示在平面上 A、B 构成的折线段对应夹角差的总和。Map_k 表示在第 k 种情况下将 n－m 个点插入到 B 中。

```
Procedure
Begin
  minSum= calucateSumOfAngleSub(A,B,1,m)/(m-2);
  For(k=1 to n-m)Do// will insert n-m nodes
  Begin
    For k1=2 to m Do // the possible insert place
    Begin // insert in the k1 place,the length of A increase
  moveDimElement(A,k1);A.length++;
  date1=calucateMinSumOfAngleSub(A,B,k1-2,k1+2);
      Fort1=1 to m-1Do
  sum+=abs(calucateSumOfAngleSub(A,B,t1,t1+2);
      sum=sum/(m-1);
      if (sum<minSum) insertedPlace=k1;
  else
  Begin
  insertPlace=m+1;length++;// insert in the tail
  date1=calucateMinSumOfAngleSub(A,B,m-1,m+1);
  End
  End
    insert(A,insertPlace,date1);
  End
End
```

其中，$moveDimElement(A,k1)$ 将位置从 $k1$ 开始的元素往后移；$calucateMinSumOfAngleSub(A,B,k1-2,k1+2)$ 计算位置 $A[k1]$ 的值，使之满足在平面上从坐标（$k1-2$，$A[k1-2]$）到（$k1+2$，$A[k1+2]$）构成的折线段 $l1$ 与从（$k1-2$，$B[k1-2]$）到（$k1+2$，$B[k1+2]$）构成的折线度 $l2$，$l1$ 和 $l2$ 每个对应的夹角差的总和最小；$calucateSumOfAngleSub(A,B,t1,t1+2)$ 则计算坐标（$t1+1$，$A[t1+1]$）和坐标（$t1+1$，$B[t1+1]$）对应夹角之差。例如，表 1 和表 2 列出了利用虚拟插值算法在 2015-5-24 得到的虚拟版本 V_2^*。

表 1 C_1 和 C_2 各版本对应的日期

	V_1	V_2	V_3	V_4
C_1	2015-05-10	2015-05-17	2015-05-31	2015-06-10
C_2	2015-05-09	2015-05-15	2015-05-31	

表 2 C_2 插值后新增的版本及日期

	V_1	V_2	V_3	V_4
C_1	2015-05-10	2015-05-17	2015-05-31	2015-06-10
	V_1	V_2	V_3	V_4
C_2	05-09	05-15	2015-05-24	05-31

4.2.3 演化相似性度量步骤

给定两个构件或系统的演化属性矩阵 A 和 B。按照下列步骤执行：

（1）如果矩阵 A 和 B 存在行列不相同的情况，按如下方式处理：

1）如果行数不相同，例如 A 的行数大于 B 的行数，调用虚拟版本插值算法，将 B 变换为矩阵 B'；

2）如果矩阵列数不同，将 A 的属性 $A.attributes$ 和 B' 的属性 $B'.attributes$ 作并操作，即

$attributes = A.attributes \cup B'.attributes$

3）在 $attributes'$ 上重新构造各自新的演化属性矩阵，例如 A' 和 B''；A' 的属性集合（$attributes$-$A.attributes$）对应列值设为 0；同理，B'' 的属性集合（$attributes$-$B'.attributes$）亦设为 0。

（2）在 A' 和 B'' 上，计算各自的演化行向量及演化相似性度量。

5 构件化软件演化相似性度量模型

5.1 构件化演化模型

文献[21]提出了支持构件演化的模型，该模型涉及的核心概念如图 3 所示。其中：

构件：构件是一个数据单元或一个计算单元。从逻辑上看，构件是可以被多个软件系统所复用的具有独立功能的系统构成成分。

构件版本和构件版本树：构件在开发或修改过程中可能会出现多个版本，这些版本记录了构件开发或修改的历史。构件可能同时存在多个开发流，构件的一个版本分支代表一

个开发流，每个分支包含多个版本。因此，构件的所有版本不再表现为版本的序列，而是构成一个树型结构，称为构件版本树。构件的版本控制通过构件的检出和检入操作完成。构件版本树记录了构件的演化情况。

配置：配置是指一组配置项的集合，其中每个配置项可以是一个构件，也可以是一个配置（作为配置项的配置也称子配置），配置具有自包含性。配置可以表示基于构件的软件开发中的复合构件，也可以表示组装出来的系统。

配置的基线：为配置及其所有子配置中的构件都选定一个特定版本，就得到了配置的一个基线，该操作称为基线操作。配置的基线表示复合构件或系统的一个版本。

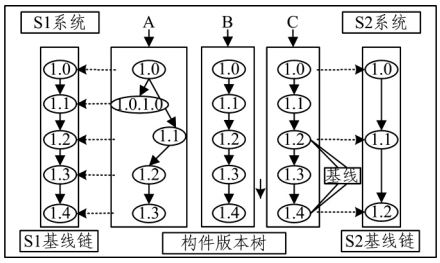

图 3　构件化软件演化模型

上述模型中，A、B 和 C 对应的是原子构件，由一组源程序或者二进制文件构成。本文将原子构件的实现假设为面向对象的程序（例如 Java 源程序）。S1 和 S2 是复合构件或者系统（系统可以看作一类特殊的复合构件），由原子构件或者其他复合构件组合而成；1.2，1.3 等表示原子构件或者系统（复合构件）对应版本或者基线的标号；标号在给定的分支或者基线链中存在一个位置关系，例如原子构件 B 的分支{1.2，1，3，1.4}中标号 1.2 的位置为 1。

在度量构件化软件演化相似性时区分 2 种情况：取任意构件的不同分支进行比较；取系统层次不同基线链进行比较。在构件化软件中，系统通常是由其他构件组装而成，因此系统可以看成一类特殊的复合构件。

5.2　（原子）构件的演化相似性度量

对原子构件，本文区分的是原子构件的近期演化行向量 $CLEP_{\overrightarrow{a_1a_2\cdots a_n}}(C,branch)$ 和早期演化行向量 $CEEP_{\overrightarrow{a_1a_2\cdots a_n}}(C,branch)$。公式如下：

$$\cdot\ CLEP_{\overrightarrow{a_1a_2\cdots a_n}}(C,branch)=\langle w_1*CLEP_{a_1}(C,branch),w_2*CLEP_{a_2}(C,branch),\cdots,w_n*CLEP_{a_n}(C,branch)\rangle$$

其中，$CLEP_a(C,branch)=\sum_{branch.start+1}^{branch.end}CEP_a(C,k)*2^{k-maxRank}$（$1\leqslant branch.start<k\leqslant branch.end=maxRank$）。

其中，CEP_a 为单维演化属性在相邻版本间（按照版本号在分支中位置递增的顺序，版本号本身不一定连续）变化的差值；$CLEP_a(C,branch)$ 作为某个构件演化的整体性指标，不仅关注构件属性的总变化，而且关注最新构件版本的变化，对每个 CEP_a 增加权重函数 $2^{k-maxRank}$，减弱过去版本属性变化的重要性，统计构件生成最新版本前所有的版本属性变化的加权总和，即在属性 a 上，构件 C 从位置为 $branch.start$ 的版本到位置为 $branch.end$ 的版本的 CEP_a 加权求和。

若原子构件的演化属性设置为文件变化（$file$）、类变化（$Class$）和大小变化（组成构件的文件个数变化——$Size$）。则相应的 $CEP_a(C,i)$ 定义如下：

$$CEP_{file}(C,i)=\begin{cases}numberFiles(C,1), & i=1\\ diff(C,i-1,i), & i>1\end{cases}$$

$$CEP_{class}(C,i)=\begin{cases}numberFiles(C,1), & i=1\\ class\_diff(C,i-1,i), & i>1\end{cases}$$

$$CEP_{size}(C,i)=\begin{cases}fileSize(C,1), & i=1\\ size\_diff(C,i-1,i), & i>1\end{cases}$$

$$CLEP_{\overrightarrow{file,class,size}}(C,branch)=\langle w_1*CLEP_{file}(C,branch),w_2*CLEP_{class}(C,branch),w_nCLEP_{size}(C,branch)\rangle$$

$$CEEP_{\overrightarrow{a_1a_2\cdots a_n}}(C,branch)=\langle w_1*CEEP_{a_1}(C,branch),w_2*CEEP_{a_2}(C,branch),\cdots,w_n*CEEP_{a_n}(C,branch)\rangle$$

其中，$CEEP_a(C,branch)=\sum_{branch.start+1}^{branch.end}CEP_a(C,k)*2^{minRank+1-k}$（$maxRank=branch.start<k\leqslant branch.end$）。

与 $CLEP_a$ 不同，$CLEP_a(C,branch)$ 反映的是构件属性早期版本变化的总体情况，即构件 C 从位置为 $branch.start$ 的版本到位置为 $branch.end$ 的版本的 CEP_a 加权求和。其中，对每个构件版本的 CEP_a 增加权重函数 $2^{minRank+1-k}$，减弱较新版本属性变化的重要性。

$$CEEP_{\overrightarrow{file,class,size}}(C,branch)=\langle w_1*CEEP_{file}(C,branch),w_2*CEEP_{class}(C,branch),w_n*CEEP_{size}(C,branch)\rangle$$

例如表 3 中构件 C_1 和 C_2 在属性 $File$，$Class$ 和 $Size$ 上的演化属性变化矩阵可知 $CLEP_{file}=2^{1-3}*1+2^{2-3}*2+2^{3-3}*5=6.25$；同理，$CLEP_{class}=6.0$，$CLEP_{size}=6.5$。因此，构件 C_1 的近期演化行向量为 $CEEP_{\overrightarrow{file,class,size}}(C_1,\{V_1,V_2,V_3,V_4\})=\langle 6.25,6.0,6.5\rangle$；类似地，$CEEP_{file}=2^{3-1}*1+2^{3-2}*2+2^{3-3}*5=13$，$CEEP_{class}=18$，$CLEP_{size}=20$，构件 C_1 的早期演化行向量 $CLEP_{\overrightarrow{file,class,size}}(C_1,\{V_1,V_2,V_3,V_4\})=\langle 13,18,20\rangle$。同理，构件 C_2 的近期演化行向量及早期演化行向量分别如下：

$$CLEP_{\overrightarrow{file,class,size}}(C_1,\{V_1,V_1^*,V_2,V_3\})=\langle 3.5,5.5,9.0\rangle;$$

$$CEEP_{\overrightarrow{file,class,size}}(C_2,\{V_1,V_1^*,V_2,V_3\})=\langle 5.0,10.0,15.0\rangle$$

构件 C_1 和构件 C_2 在近期的演化相似性为：

$$SIM_{\overrightarrow{file,class,size}}=(CLEP_{\overrightarrow{file,class,size}}(C_1,\{V_1,V_2,V_3,V_4\}),CLEP_{\overrightarrow{file,class,size}}(C_2,\{V_1,V_1^*,V_2,V_3\}))=(\langle 6.25,6.0,6.5\rangle,\langle 3.5,5.5,9.0\rangle)=0.94191$$

构件 C_1 和构件 C_2 在早期的演化相似性为：

$$SIM_{\overrightarrow{file,class,size}}=(CEEP_{\overrightarrow{file,class,size}}(C_1,\{V_1,V_2,V_3,V_4\}),CEEP_{\overrightarrow{file,class,size}}(C_2,\{V_1,V_1^*,V_2,V_3\}))=(\langle 13,18,20\rangle,\langle 5.0,10.0,15.0\rangle)=0.9748$$

表 3　构件 C_1，C_2 的变化

C_1				C_2			
⟨File，Class，Size⟩							
$V_1->V_2$	1	2	2	$V_1->V_1^*$	0	0	0
$V_2->V_3$	2	3	4	$V_1^*->V_2$	1	3	4
$V_3->V_4$	5	4	4	$V_2->V_3$	3	4	7

5.3　系统（或复合构件）的演化相似性度量

对复合构件，本文亦区分的为复合构件的近期演化行向量 $BLEP_{\overrightarrow{a_1a_2\cdots a_n}}(B,baseline)$ 和早期演化行向量 $BEEP_{\overrightarrow{a_1a_2\cdots a_n}}(B,baseline)$。

$$\cdot\ BLEP_{\overrightarrow{a_1a_2\cdots a_n}}(B,baseline)=\langle w_1*BLEP_{a_1}(B,baseline),w_2*BLEP_{a_2}(B,baseline),\cdots,w_n*BLEP_{a_n}(B,baseline)\rangle$$

其中，$BLEP_a(B,baseline)=\sum_{baseline.start+1}^{baseline.end} BEP_a(B,k)*2^{k-maxRank}$ $(1\leqslant i<k\leqslant j=maxrank)$。

$BLEP_a(B,baseline)$作为系统演化的整体性指标，反映的是系统在指定基线链上的变化。因此，该度量从系统全局的角度来分析，重点关注最新基线的变化，对每个基线的 BEP 增加权重函数 $2^{k-maxRank}$，统计基线生成最新版本前所有的基线属性变化的加权总和，即系统 B 从位置为 $baseline.start$ 的基线到位置为 $baseline.end$ 的基线的 BEP 加权求和。

在系统（复合构件）层次，可设置演化属性为构件、软件体系结构，即：

$$BEPcomponent(B,i)=\begin{cases}numberOfComponents(B,1), & i=1\\ component\_diff(B,i-1,i), & i>1\end{cases}$$

$$BEP_{SA}(B,i)=\begin{cases}SA\_diff(B,null), & i=1\\ SA\_diff(B,i-1,i), & i>1\end{cases}$$

其中，$component_diff(B,i-1,i)$用于计算系统（复合构件）第 i 个基线和第 $i-1$ 个基线之间变化（包括增加、剔除和修改）的构件数目；$SA_diff(B,i-1,i)$表示系统（复合构件）第 $i-1$ 个基线和第 i 个基线的差异性，本文利用 MojoFM[18] 来计算两个基线在软件体系结构 SA 上的差异性。

$BLEP_{\overrightarrow{component,SA}}(B,baseline)=\langle w_1*BLEP_{component}(B,baseline),w_2*CLEP_{SA}(B,baseline)\rangle$

· $BEEP_{\overrightarrow{a_1a_2\cdots a_n}}(B,baseline)=\langle w_1*BEEP_{a_1}(B,baseline),w_2*BEEP_{a_2}(B,baseline),\cdots,w_n*BEEP_{a_n}(B,baseline)\rangle$

其中，$BEEP_a(B,baseline)=\sum_{baseline.start+1}^{baseline.end} BEP_a(B,k)*2^{minRank+1-k}$ $(minRank=baseline.start<k\leqslant baseline.end)$。

类似地，$BEEP_a(B,baseline)$从系统的角度分析系统从开始基线（$baseline.start$）到最新基线（$baseline.end$）的早期属性变化的加权总和，即：

$BEEP_{\overrightarrow{component,SA}}(B,baseline)=\langle w_1*BEEP_{component}(B,baseline),w_2*BEEP_{SA}(B,baseline)\rangle$

例如表 4 为系统（复合构件）S_1 和 S_2 在属性 $Component$ 和 SA 上的演化属性变化矩阵，经计算 S_1 和 S_2 在近期的演化相似性为：

$SIM_{\overrightarrow{component,SA}}(BLEP_{\overrightarrow{component,SA}}(S_1,\{B_1,B_2,B_3,B_4\}),BLEP_{\overrightarrow{component,SA}}(S_2,\{B_1,B_2,B_3,B_4\})=(\langle 4,3.25\rangle,\langle 2.5,2.75\rangle)=0.9886$

S_1 和 S_2 在早期的演化相似性为：

$SIM_{\overrightarrow{component,SA}}(BEEP_{\overrightarrow{component,SA}}(S_1,\{B_1,B_2,B_3,B_4\}),BCEEP_{\overrightarrow{component,SA}}(S_2,\{B_1,B_2,B_3,B_4\})=(\langle 13,10\rangle,\langle 16,17\rangle)=0.9872$

表 4　系统（复合构件）S_1，S_2 的变化

	S_2		S_1	
	⟨Component，SA⟩			
$B_1->B_2$	2	1	2	3
$B_2->B_3$	1	2	4	2
$B_3->B_4$	3	2	0	1

6　实例分析

针对提出的构件化软件演化分析框架及演化相似性度量方法，本文选择 3 个开源软件系统（Hbase，cassandra 和 openjpa）进行了实验。

6.1　数据预处理

目前的软件版本控制系统（例如 CVS 等）以文件为存储单位，记录了每个文件的变化历史。为了实现构件化软件演化相似性度量，本文基于文献[17]的基础，在逆向得到的软件体系结构视图的基础上为其每个构件构造其版本编号，并进一步生成其构件的版本树和系统的基线。表 5 列出了这 3 个开源软件系统经过数据处理阶段得到的构件个数、部分构件的分支长度以及系统版本个数（即基线长度）的统计结果。其中，经逆向得到 cassandra 系统每个版本的构件组成如图 4 所示，例如系统 cassandra 的 0.5.1 版本由构件 cassandra $ apache_cassandra_db 的 1.2 版本、cassandra $ apache_cassandra_gms 的 1.2 版本、cassandra $ apache_cassandra_locator 的 1.0 版本以及 cassandra $ apache_cassandra_cli 的 1.2 版本等组成。

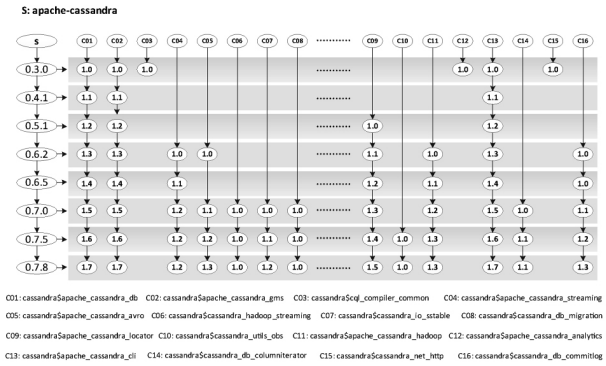

图 4　逆向得到的 cassandra 系统及其构件变化历史

表 5　3 个开源系统的预处理汇总

系统名称	构件个数	前 16 个构件的分支长度								基线长度
Hbase	64	1	1	3	5	6	1	7	5	12
		5	1	1	3	1	1	5	6	
cassandra	37	8	8	1	3	4	1	3	1	8
		6	1	4	1	8	2	1	4	
openjpa	43	12	3	12	4	12	9	12	5	12
		12	5	12	8	9	12	8	12	

6.2　度量及分析

6.2.1　针对构件分支

对构造出来的构件分支两两之间进行软件演化相似性度量，在度量过程中若分支长度不同，则首先进行虚拟版本插值处理。表 6 是计算得到的 cassandra 系统中部分构件（前 7 个）间早期演化相似性度量矩阵。

表 6　构件（前 7 个）间早期演化相似性矩阵

构件	值向量间的余弦						
	1	2	3	4	5	6	7
1	1.000	0.937	0.864	0.928	0.932	0.946	0.865
2	0.937	1.000	0.981	0.987	0.963	0.987	0.965
3	0.864	0.981	1.000	0.985	0.961	0.975	0.991
4	0.928	0.987	0.985	1.000	0.993	0.999	0.989
5	0.932	0.963	0.961	0.993	1.000	0.994	0.981
6	0.946	0.987	0.975	0.999	0.994	1.000	0.981
7	0.865	0.965	0.991	0.989	0.981	0.981	1.000

1-cassandra $ apache_cassandra_db
2-cassandra $ apache_cassandra_gms
3-cassandra $ cql_compiler_common
4-cassandra $ apache_cassandra_streaming
5-cassandra $ apache_cassandra_avro
6-cassandra $ cassandra_hadoop_streaming
7-cassandra $ cassandra_io_sstable

基于构件早期演化相似性矩阵采用层次聚类算法对 cassandra 的 37 个构件进行聚类，可以判断哪些构件的演化相似性是一类的。例如在图 5 所示的冰状图中，当指定聚类个数为 36 时，构件 cassandra $ apache_cassandra_db 和构件 cassandra $ apache_cassandra_utils 位于同一个簇内，其演化相似性类似。

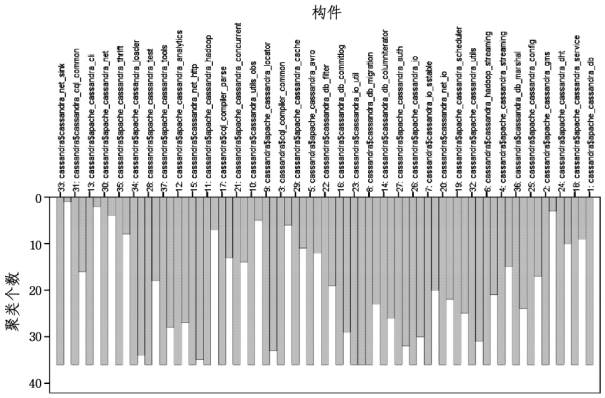

图 5　使用层次聚类算法效果图

6.2.2　针对系统基线

表 5 中的 3 个开源系统的基线长度不一致，在实验过程中以 Hbase 的基线长度为标准将 Cassandra 升维到 12 个版本，得到的近期演化行向量分别为：

$BEEP\xrightarrow[component,SA]{}=(Cassandra, baseline)=\langle 2.46875, 11.025390625\rangle$

$BEEP\xrightarrow[component,SA]{}=(Hbase, baseline)=\langle 70.578125, 167.544921875\rangle$

$BEEP\xrightarrow[component,SA]{}=(openJPA, baseline)=\langle 65.4140625, 91.35742875\rangle$

经计算，Cassandra 与 Hbase 的近期演化相似性为：0.98412，Cassandra 与 openJPA 的近期演化相似性为 0.92062，Hbase 与 openJPA 的近期演化相似性为 0.97530，因此 Cassandra 与 Hbase 在近期的演化更相似。类似地，Cassandra 与 Hbase 的早期演化相似性为 0.94283，Cassandra 与 openJPA 的早期演化相似性为 0.96735，Hbase 与 openJPA 的早期演化相似性为 0.99651，因此，Cassandra 与 openJPA 的早期演化更相似。事实上，当以 OpenJPA 为标准进行升维处理时，得到的演化相似性结果也是相同的。

7　系统原型

整个软件演化相似性度量系统采用 B/S 的网络架构模式开发，如图 6 所示。该系统总共分为 3 层：用户界面层、演化信息处理层、数据存储层。用户界面层：用于与终端用户进行交互，提供构件化软件演化信息管理、构件化软件演化相似性度量及查询的输入输出界面。演化信息处理层：1）构件化软件演化信息的度量模块，例如构建演化属性矩阵、演化行向量、构件分支或者软件体系结构基线在早期版本和最新版本下的演化相似性度量；2）构件化软件演化信息获取模块，例如逆向工程工具生成系统的软件体系视图，并基于版本演化恢复其构件的版本树及系统的基线链。数据存储层：主要实现对构件相关信息的存储，如：构件名称、基于 SCM 获得的构件版本序列、构件的多维演化属性值等。

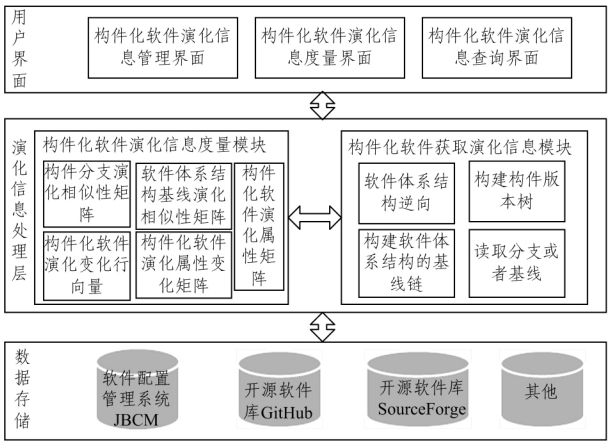

图 6　系统体系结构

结束语　在构件化软件开发中，利用演化信息能够预测软件在未来一段时间内的演化趋势，从而更好地辅助软件的开发以及后期对软件的维护。传统的对软件演化信息的度量研究并没有将多维演化属性与软件的演化相似性关联起来。为此，本文提出了一种构件化软件演化相似性度量模型，用于度量软件在多维演化属性上的演化相似性。同时，通过开发相应的支持平台，对 3 个开源软件系统进行了软件演化相似性的聚类分析。在今后的工作中，将进一步改进版本树的生成算法、增加更多属性和实验等不同权重对度量结果的影响，实现对构件化软件演化趋势的预测。

参 考 文 献

[1] Louridas P, Spinellis D, Vlachos V. Power Laws in Software[J]. Acm Transactions on Software Engineering & Methodology, 2008, 18(1): 617-632

[2] Jenkins S, Kirk S R. Software architecture graphs as complex networks: A novel partitioning scheme to measure stability and evolution[J]. Information Sciences, 2007, 177(12): 2587-2601

[3] Lehman M M, Ramil J F, Wernick P D, et al. Metrics and laws of software evolution-the nineties view[C]// IEEE International Software Metrics Symposium. 1997

[4] Gonzalez-Barahona J M, Robles G, Michlmayr M, et al. Macro-level software evolution: a case study of a large software compilation[J]. Empirical Software Engineering, 2009, 14(3): 262-285

[5] Izurieta C, Bieman J. The evolution of FreeBSD and Linux [C]// International Symposium on Empirical Software Engineering. 2006: 204-211

[6] Girba T, Ducasse S. Modeling history to analyze software evolution[J]. Journal of Software Maintenance & Evolution Research & Practice, 2006, 18(3): 207-236

[7] Nikora A P, Munson J C. An Approach to the Measurement of Software Evolution[J]. Journal of Software Maintenance & Evolution Research & Practice, 2005, 17(1): 65-91

[8] Bhattacharya P, Iliofotou M, Neamtiu I, et al. Graph-based analysis and prediction for software evolution[C]// International Conference on Software Engineering. 2012: 419-429

[9] Sangwan R S, Vercellone-Smith P, Neill C J. Use of a multidimensional approach to study the evolution of software complexi-

ty[J]. Innovations in Systems & Software Engineering, 2010, 6(4): 299-310

[10] Alexandrescu R, Bottle A, Min H J, et al. Mining Software Repositories with iSPARQL and a Software Evolution Ontology [C]// International Workshop on Mining Software Repositories, 2007. ICSE Workshops MSR. 2007: 10-10

[11] Robles G, Herraiz I, German D M, et al. Modification and developer metrics at the function level: Metrics for the study of the evolution of a software project[C]// International Workshop on Emerging Trends in Software Metrics. IEEE, 2012: 49-55

[12] D'Ambros M, Lanza M. A Flexible Framework to Support Collaborative Software Evolution Analysis[C]// Csmr. IEEE Computer Society. 2008: 3-12

[13] Emanuel A W R, Wardoyo R, Istiyanto J E, et al. Modularity Index Metrics for Java-Based Open Source Software Projects[J]. International Journal of Advanced Computer Sciences & Applications, 2013, 2(11): 52-58

[14] Nakamura T, Basili V R. Metrics of Software Architecture Changes Based on Structural Distance[C]// IEEE International Symposium on Software Metrics. IEEE, 2005

[15] Le D M, Behnamghader P, Garcia J, et al. An empirical study of architectural change in open-source software systems[C]// MSR. 2015: 235-245

[16] Lehman M M. Laws of software evolution revisited[C]// European Workshop on Software Process Technology. Springer-Verlag, 1996: 108-124

[17] Kouroshfar E, Mirakhorli M, Bagheri H, et al. A Study on the Role of Software Architecture in the Evolution and Quality of Software[C]// Mining Software Repositories. IEEE, 2015: 246-257

[18] Tzerpos V, Holt R C. MoJo: A Distance Metric for Software Clusterings[C]// Working Conference on Reverse Engineering. IEEE Computer Society, 1999: 187-193

[19] 杨芙清. 软件工程技术发展思索[J]. 软件学报, 2005, 16(1): 1-7

[20] 张路, 谢冰, 梅宏, 等. 基于构件的软件配置管理技术研究[J]. 电子学报, 2001, 29(2): 266-268

[21] 钟林辉, 谢冰, 邵维忠. 扩充 CDL 支持基于构件的系统组装与演化[J]. 计算机研究与发展, 2002, 39(10): 1361-1365

[22] 钟林辉, 侯长源, 宗洪雁, 等. 构件化软件演化信息及演化相似性度量技术研究[J]. 计算机应用研究, 2015, 32(5): 1399-1402, 1416

(上接第 473 页)

Jim Gray 认为当今以及未来科学的发展趋势是随着数据量的高速增长，计算机将不仅仅能做模拟仿真，还能进行分析总结得到理论[6,10]。也就是说，过去由牛顿、爱因斯坦等科学家从事的工作，未来可以由计算机来做。这种科学研究的方式，即为今后会高速发展的数据密集型科学。通俗来说，大数据的核心就是预测。实际上，谷歌的广告优化配置、战胜国际围棋大师李世石的谷歌机器人 AlphaGo 也是极其成功的案例，这就是数据密集型科学的魅力所在。

显然，处在这个时代，随着大数据技术的快速发展，人类对科学研究思维方式的转变将会逐步放弃对以往各种传统因果关系的探求并开始重视相关关系的研究。换言之，这将颠覆以往人类的思维惯例，人们将不需要像以往那样知道为什么而只需要知道是什么就行，这对世界交流的方式是一种挑战，对各国人民的认知也是一种挑战。Chris Anderson(克里斯·安德森)2008 年曾发出惊人的断言：The data deluge makes the scientific method obsolete(数据洪流使传统科学方法变得过时)[11]。人类在获得海量数据(即 big data)后，运用各种大数据分析工具，如 Hadoop 等来处理这些数据，将为后人理解世界提供的一条完整的新途径。当前各国投入巨大的人力物力研究开发大数据处理技术，将会为人类创造出不容置疑的变革，这类可量化的维度是前所未有的。大数据及其处理技术即将成为全球人类创造新发明和新服务的思维方式的源泉，越来越多的改革正蓄势待发，世界将由大数据带来巨变。

结束语　大数据的研究成果让人类能够完成没有建立完整的模型和假设，也可以对收集到的数据进行分析。如果将收集的数据录入计算机集群，只要有相互关系的数据，通过专业的大数据分析工具就可以发现过去的科学方法发现不了的新模式、知识和规律。数据量巨大的大数据让我们体会到：Correlation is enough。常规的模型的探索可以停顿，因为因果关系已经被相互关系取代。不管对“数据密集型科学”的理解有多深，必须得承认：数据密集型科学不仅是科研方式的转变，也是人们思维方式的大变化。

参 考 文 献

[1] 范平. 大数据时代，你不得不懂中关村在线[EB/OL][2013-02-18]. http://chuansong.me/, http://blog.sina.com, http://oatos.diandia

[2] 大数据. 维基百科[EB/OL]. [2012-10-5]. http://zh.wikipedia.org/wiki

[3] 王雄. 大数据究竟是什么？一篇文章让你认识并读懂大数据[EB/OL]. [2015-06-18]. http://blog.sina.com.cn/u/2262489275

[4] 维克托迈尔·舍恩伯格, 肯尼思·库克耶著. 大数据时代[M]. 杭州: 浙江人民出版社, 2013

[5] 黄宜华. 深入理解大数据-大数据处理与编程实践[M]. 北京: 机械工业出版社, 2014

[6] 孙晓立. 大数据：让“云”落地成“雨”[J]. 中国科技投资, 2012, (Z2): 43-45

[7] 周傲英. 数据密集型计算-数据管理技术面临的挑战[J]. 中国计算机学会通讯, 2009, 5(7): 50-53

[8] Big data -Wikipedia. the free encyclopedia[EB/OL]. [2013-09-23]. http://en.wikipedia.org/wiki/Big_data

[9] http://v.qq.com/boke/page/y/0/e/y0115diwl0e

[10] 刘念真. 利用 Oracle 信息模型驾驭大数据[EB/OL]. [2014-06-22] http://wenku.baidu.com/view/abfb3a1552d380eb62946d9d.html

[11] 克里斯·安德森. 长尾理论[M]. 北京: 中信出版社, 2006