一种基于演进原则度量的软件架构持续演进效果评估方法

王 桐,廖 力,李必信
(东南大学计算机科学与工程学院,江苏南京 211100)

摘 要: 本文提出一种基于演进原则度量的软件架构持续演进效果评估方法(Software Architecture Evolution Principles,SAEP),首先该方法给出与软件架构演进相关的四个代表性的软件架构演进原则;然后通过对这四个架构演进原则的度量结果评估软件架构的演进效果;最后把本文的方法应用于8个代表性的开源项目,并分别进行有效性验证实验,实验结果表明:无论是单个演进原则还是四个演进原则结合起来,均能有效反映软件架构演进效果的好坏.单个原则用来关注软件架构特定方面的演进效果,四个原则结合起来关注软件架构的综合演进效果.

关键词: 软件架构; 软件架构演进原则; 演进原则度量; 演进效果评估
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2019)07-1475-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2019.07.010

An Approach to Evaluate the Sustainable Evolution Effect of Software Architecture Based on the Measurements of Evolution Principles

WANG Tong,LIAO Li,LI Bi-xin
(*School of Computer Science and Engineering,Southeast University,Nanjing,Jiangsu* 211100,*China*)

Abstract: In this paper,we propose software architecture evolution principles(SAEP)to evaluate the evolution effect of software architecture based on the measurements of evolution principles. In SAEP,we firstly propose four representative principles for software architecture evolution. Secondly,the evolution effect of software architecture is evaluated based on the measurement of architecture evolution principles. Finally,eight representative open source projects are performed by experiments for verifying the effectiveness respectively. The experimental results show that a single evolution principle or the combination of four evolution principles can effectively reflect the evolution effect of the software architecture. A single principle focuses on a specific aspect of the evolution effects of the software architecture. The combination of four principles focuses on the comprehensive evolution effects of software architecture.

Key words: software architecture;software architecture evolution principle;measurement of evolution principles;evaluation of evolution effect

1 引言

随着软件演进,常常会发生软件架构腐蚀现象,软件架构腐蚀会带来一些较大的危害:增加软件演进成本、降低软件性能、降低软件质量等[1].因此,评估软件架构演进效果对及时发现软件架构腐蚀和促进架构向好演进具有十分重要的意义.

当前人们对软件架构演进原则的研究主要集中于设计阶段演进原则.例如Wermelinger等人于2008年基于软件架构无环依赖原则、开闭原则、共同重用原则、共同封闭原则和稳定依赖原则评估了Eclipse项目的演进效果[2,3],但上述原则没有关注软件架构演进特点,为了评估软件架构在演进过程中的架构变化及其演进效果,本文提出一种基于演进原则度量的软件架构演进效果评估方法,简称SAEP(Software Architecture Evolution Principles).

收稿日期:2018-06-25;修回日期:2018-09-26;责任编辑:孙瑶
基金项目:国家重点研发计划(No.2018YFB1003902);国家自然科学基金(No.61572126,No.61872078,No.61402103)

2 SAEP 方法

SAEP 方法包含如下三个主要步骤:①基于软件架构恢复的信息提取;②软件架构演进原则度量;③软件架构演进效果评估. SAEP 方法具体流程如图 1 所示.

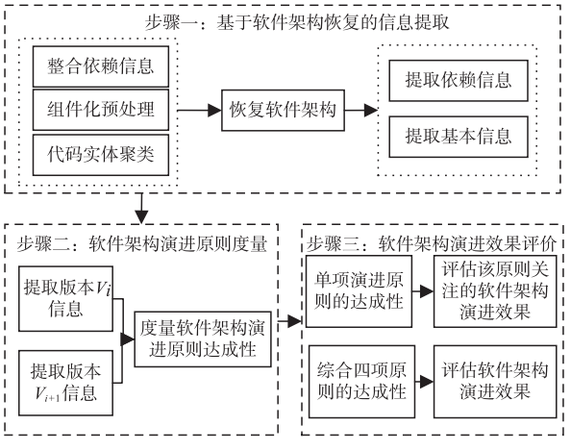

图 1 SAEP方法流程图

2.1 基于软件架构恢复的信息提取

本文所采取的架构恢复方法包含以下步骤:①依赖信息整合,通过抽象语法树中的限定名和属性提取代码实体间的依赖关系,构造各个粒度的依赖图;②组件化预处理,将具有双向依赖关系、环依赖关系、开环依赖关和传递依赖关系的实体聚类为新的实体[4];③代码实体聚类,依赖强度(Dependency Intensity)是衡量代码实体间依赖关系紧密程度的指标. 代码实体 a 和代码实体 b 之间的依赖强度 $DI(a,b)$ 计算公式如下:

$$DI(a,b)=\frac{\sum_{k=1}^{N}W(C_{ak},C_{bk})*\mathrm{Num}(C_{ak},C_{bk})}{|a|+|b|} \tag{1}$$

其中,$|a|$ 和 $|b|$ 分别是 a 和 b 中变量和方法的总数,N 是 a 和 b 间具有依赖关系的实体对,$C_{ai}(i=1,2,\cdots,k)$ 是 a 中的变量或方法,$\mathrm{Num}(C_{ak},C_{bk})$ 是实体 C_{ak} 依赖于实体 C_{bk} 的次数,$W(C_{ak},C_{bk})$ 是 C_{ak} 与 C_{bk} 之间的依赖类型的权重. 依据实体粒度的大小,各依赖类型的权重如表 1 所示.

表 1 各依赖类型权重

权重	依赖类型
0.3	目录间调用依赖,目录间组合依赖,聚合依赖
0.4	引用依赖
0.5	目录内调用依赖,声明定义依赖,目录内组合依赖,继承依赖,实例化依赖,参数类型依赖

通过以上三个阶段完成基于源码的软件架构恢复,然后基于软件架构提取软件架构基本信息和依赖信息.

2.2 软件架构演进原则度量

软件架构是组件以及组件之间交互的作用关系的高层抽象[5]. 演进对象类别包括规模演进和依赖关系的演进;演进对象粒度包括软件架构整体演进和某一组件演进. 本文将演进对象类别和演进对象粒度相交叉结合,提出四项软件架构演进原则:关注于软件架构整体依赖关系变化的主体稳定原则、关注于组件间依赖关系变化的独立演进原则、关注于软件架构整体规模变化的平滑演进原则和关注于组件内部规模变化的组件精简原则. 下文基于表 2 中所列参数介绍各软件架构演进原则及其度量方法.

表 2 参数简介

符号	含义
V_i	第 i 个版本
E_i	版本 V_i 的软件架构图中依赖边集合
$\lvert E_i\rvert$	版本 V_i 的软件架构图中依赖边总数
N_i	版本 V_i 的软件架构图中组件集合
$\lvert N_i\rvert$	版本 V_i 的软件架构图中组件总数

(1)主体稳定原则(Main Body Stability Principle, MSP)

软件架构由组件和组件间的依赖关系组成,因此组件和组件间依赖关系构成了软件架构的主体. 组件和组件间依赖关系大幅度的变化会引入大量不确定因素,例如组件功能是否稳定等. 本文通过组件和组件间依赖关系的变化情况作为度量软件架构主体是否稳定的指标[6],度量公式如下:

$$MSP=\frac{2|D|-|C|+2|B|-|A|}{|A|+|C|} \tag{2}$$

其中,

$$A=E_i\cup E_{i+1} \tag{3}$$

$$B=E_i\cap E_{i+1} \tag{4}$$

$$C=N_i\cup N_{i+1} \tag{5}$$

$$D=N_i\cap N_{i+1} \tag{6}$$

其中,MSP 是从 V_i 演进为 V_{i+1} 的主体稳定原则的度量结果,A 和 C 分别是 V_i 和 V_{i+1} 的所有依赖边和所有组件构成的集合,B 和 D 分别是 V_i 和 V_{i+1} 的相同依赖边和相同组件构成的集合,$|X|$ 为集合 X 中的元素总数.

(2)独立演进原则(Independent Evolution Principle, IEP)

演进过程中应尽量降低组件间耦合度,促使组件相对独立的演进,避免对其他组件造成影响,本文以平均关联的组件数作为衡量组件是否独立的指标[7],度量公式如下:

$$IEP=\left(\frac{E_i}{N_i}-\frac{E_{i+1}}{N_{i+1}}\right)\Big/\left(\frac{E_i}{N_i}+\frac{E_{i+1}}{N_{i+1}}\right) \tag{7}$$

其中,IEP 是从 V_i 演进为 V_{i+1} 时的独立演进原则的度量

结果.

(3)平滑演进原则(Smooth Evolution Principle, SEP)

为避免引入潜在威胁和降低软件演进成本,利益相关者以尽量小的变更满足新需求实现平缓演进[8],本文以各组件规模变化比例的平均值作为度量软件架构规模是否平滑演进的指标,度量公式如下:

$$SEP = 1 - 2\frac{\sum_{c=1}^{N+1}\frac{|LOC(i,c) - LOC(i+1,c)|}{\max\{LOC(i,c),LOC(i+1,c)\}}}{|N_{i+1}|} \tag{8}$$

其中,SEP 是从 V_i 演进为 V_{i+1} 的平滑演进原则的度量结果,c 表示组件的编号,$LOC(i,c)$ 表示 V_i 中组件 c 的规模,$LOC(i+1,c)$ 表示 V_{i+1} 中组件 c 的规模.

(4)组件精简原则(Simple Component Principle, SCP)

引入新代码可能会引起组件规模和组件间依赖关系的复杂度上升等问题,因此应对内聚度较高的组件解耦,组件精简的主要特征是组件规模降低[9],本文以组件规模的平均值作为度量组件精简程度的指标,度量公式如下:

$$SCP = \left(\frac{LOC_i}{N_i} - \frac{LOC_{i+1}}{N_{i+1}}\right) \Big/ \left(\frac{LOC_i}{N_i} + \frac{LOC_{i+1}}{N_{i+1}}\right) \tag{9}$$

其中,SCP 是从 V_i 演进为 V_{i+1} 的组件精简原则的度量结果,LOC_i 表示 V_i 的代码行总数,LOC_{i+1} 表示 V_{i+1} 的代码行总数.

通过以上四个演进原则的度量公式及其中参数含义可知,这四个演进原则的达成性值域均是[-1,1]. 度量值大于0时,说明达成了该项原则;度量值小于0时,说明没有达成该项原则. 度量值越大,原则达成性越好;反之,原则达成性越差.

2.3　软件架构演进效果评估

本文所提出的四个演进原则分别关注于软件架构演进中的不同方面,单个演进原则达成性用来评估其关注点的演进效果,四个演进原则达成性结合起来用于评估软件架构的综合演进效果.

(1)基于单个演进原则达成性评估其关注点的演进效果. 本文提出的四个软件架构演进原则从四个不同方面评估软件架构演进效果. 主体稳定原则关注软件架构整体约束即结构的变化,评估软件架构框架是否发生大幅度变动. 独立演进原则关注软件架构中组件间约束的紧密程度的变化,评估软件架构中各组件间耦合度的变化. 平滑演进原则关注软件架构整体规模的变化,评估软件架构规模是否发生了大幅度的变更. 组件精简原则关注组件的规模的变化,评估软件架构中组件是否逐步趋于精简.

(2)四个演进原则达成性结合起来评估软件架构的综合演进效果. 单个演进原则的达成性仅反映软件架构某一方面的演进情况,并不能反映软件架构整体的演进效果. 因此,我们将四个演进原则达成性结合起来,综合软件架构各方面的演进情况,进而评估软件架构综合演进效果.

3　实验研究

3.1　实验对象

我们从 Github 中随机选取了 8 个开源项目,并分别选 master 分支上的最新的25 个 release 版本作为实验对象(实验版本的选取时间为 2017/9/8). 这 8 个开源项目的基本信息如表 3 所示.

表 3　开源项目基本信息

项目名称	发布者	软件规模(LOC)
tablesaw	jtablesaw	[10681,22795]
caffeine	ben-manes	[31289,43383]
FastAdapter	mikepenz	[5942,9394]
guava	google	[22184,419113]
Hystrix	Netflix	[26854,45442]
metrics	dropwizard	[8156,14993]
realm-java	realm	[60177,71258]
async-http-client	AsyncHttpClient	[25911,31368]

3.2　实验结果与分析

RQ1:单个演进原则的度量结果是否能有效的反映该原则所关注的软件架构演进效果?

首先统计各版本的演进原则,然后分析演进原则异常的演进过程,探究异常原因,并进一步分析演进问题是否与演进原则度量结果一致,从而验证演进原则达成性是否能有效的反映其关注的软件架构演进效果.

如图 2 所示,大多数项目原则达成性集中于[-0.06,0.06]之间,但 guava 数据分布较广. guava 的第2、第4 和第8 演进过程的演进原则达成性较低,其组件的平均出度由 1 上升至3,导致组件间依赖关系复杂度上升,因此独立演进原则达成性的波动与组件间依赖关系的变化一致. 演进日志表明第2 个演进过程至第8 个演进过程中,该项目相继发布向后兼容版本与非向后兼容版本,且非向后兼容版本相比于已重构的向后兼容版本的组件间依赖关系复杂度更高,因此当非向后兼容版本时,独立演进原则达成性较低.

综上所述,独立演进原则能有效反映组件间依赖关系的变化情况. 演进时应借鉴优秀的软件架构设计重构软件架构,避免由于组件间耦合度上升而造成软件架构维护困难的问题.

如图 3 所示,原则达成性集中在[0.9,1]之间,说明开发者会尽量保持软件架构主体框架的稳定,但 metrics 数据分布较广,其最小值远低于其他项目. metrics

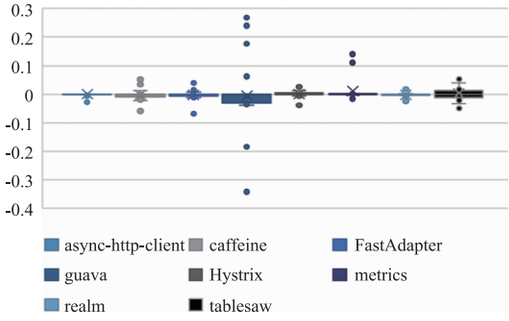

图 2 独立演进原则

的第 7 个演进过程中原则达成性急剧下降. 其架构图表明在该次演进过程中组件所包含的代码实体的变化导致组件间依赖关系变化,软件架构主体出现了不稳定的问题,因此主体稳定演进原则达成性波动与架构整体依赖关系的变化一致. 演进日志表明第 7 个演进过程的演进内容是版本 2. 1. 5 演进到版本 2. 2. 0,软件架构被大规模的重构,导致软件架构中的组件和组件间关系发生剧烈变化,导致无法保证组件接口兼容性、软件架构合理性.

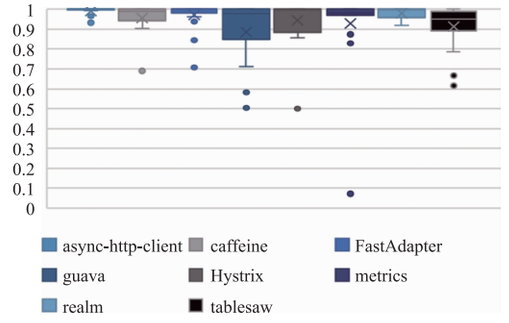

图3 主体稳定原则

综上所述,主体稳定原则能有效反映软件架构主体框架的变化情况. 演进时应对软件架构的演进应是从部分组件开始,避免由于剧烈变动而引入软件架构隐患.

如图 4 所示,大多数项目的原则达成性集中于[-0. 1,0. 2]之间,但 tablesaw 数据分布较广. tablesaw 的第 5 个和第 10 个演进过程达成性大幅度降低,第 8 个和第 18 个演进过程的原则达成性较高. 其架构图表明达成性降低是由于开发者对一个组件大幅度扩充,组件代码行的数量显著提高,但组件数目不变,所以组件平均规模升高. 而原则达成性升高是由于开发者对组件解耦,组件平均规模下降,组件精简度较高,因此组件精简演进原则达成性波动与组件规模的变化一致. 演进日志表明该项目发布时间较晚,最初的 3 个版本是内测版本,且在软件开发初期,开发者频繁扩充组件和组件解耦,导致精简度出现了波动.

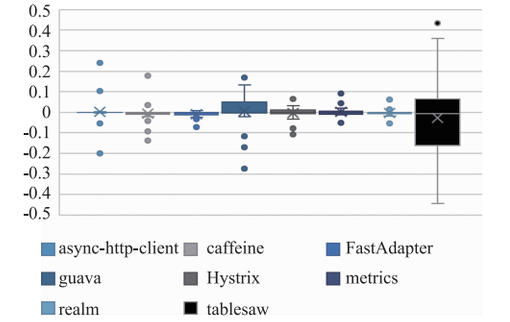

图4 组件精简原则

综上所述,组件精简原则能有效反映组件规模的变化情况. 在演进的过程中及时对低内聚的组件进行解耦,避免由于组件规模过大而导致维护困难、复用性差等问题.

如图 5 所示,大部分版本都达成了该项原则. guava、metrics 和 tablesaw 的数据分布较广. 架构图表明这三个项目频繁的拆分组件和合并组件,导致了原有组件的消失和新组件的出现. 平滑演进原则达成性的波动与架构整体规模的变化一致,因此本原则达成性有效反映其关注点的架构演进效果. 演进日志表明各组件功能复杂且规模过大,并增加了大量新组件. 基于该原则发现架构规模变更剧烈是由于开发初期没有划定各组件的功能,导致演进中不断对组件进行合并和拆分.

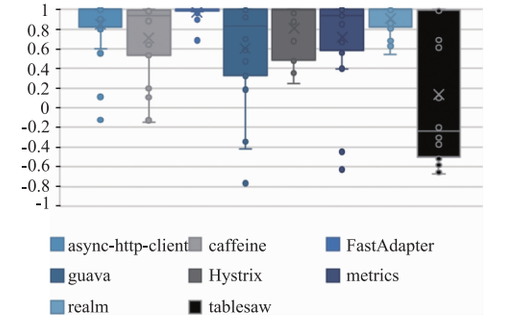

图5 平滑演进原则

综上所述,平滑演进原则能有效反映软件架构规模的变化情况. 应提前规划各组件的功能,从而避免由于组件频繁的合并和拆分导致软件架构规模剧烈变动.

RQ2:四个演进原则结合起来的度量结果是否能有效的反映软件架构综合演进效果?

我们将各版本的四项演进原则达成性的平均值记为其综合达成性,通过综合达成性与演进成本和软件可维护性之间的关系验证综合达成性是否能有效反映软件架构的综合演进效果.

演进效果较好的软件架构由于它更有利于变更,因此其可维护性较好,同时其下次演进所需的演进成本更低. Coleman 等人[10]提出的可维护性度量方法具有较高

的认可度,因此本实验中采用该度量方法计算软件可维护性. 代码行是估算软件成本的重要指标,因此以演进过程中产生的代码变更量作为衡量演进成本的指标.

实验步骤如下:①统计各版本的可维护性和下一次演进的代码变更量;②按综合达成性由高到低将版本排序,按版本数量平均分成五个等级,其中一级最优,并计算各等级综合达成性的平均值;③计算各级别代码变更量平均值和可维护性平均值.

各综合达成性等级的代码变更量如图 6,其中横坐标是综合达成性等级,纵坐标是各级别的综合达成性均值和代码变更量均值.

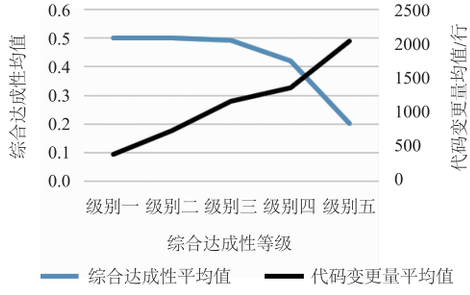

图6　综合达成性平均值和代码变更量平均值

如图 6 所示,两者相关系数是 -0.8924,即综合达成性与演进成本负相关,说明演进效果越好的版本在下一次演进过程中的演进成本越低,因此综合达成性能有效反映软件架构的演进效果.

将四项原则达成性与可维护性平均值进行线性拟合,拟合信息如表 4 所示.

表 4　可维护性平均值及其线性拟合信息

项目名称	斜率	可决系数
async-http-client	2.391	0.921
caffeine	1.197	0.7168
FastAdapter	1.022	0.6158
guava	1.172	0.424
Hystrix	0.056	0.688
metrics	14.809	0.9275
realm	0.644	0.623
tablesaw	1.856	0.416

如表 4,可决系数说明拟合的线性公式足以表现数据间关系. 斜率说明综合达成性与可维护性是正相关关系.

通过综合达成性与演进成本和软件可维护性之间的关系说明综合达成性能有效反映软件架构的综合演进效果.

RQ3:哪个演进原则的达成性与软件架构综合演进效果具有更高的相关性?

我们以演进成本即代码变更量作为衡量软件架构演进效果的指标. 基于各演进原则达成性和下一次演进过程中的代码变更量建立多元线性模型,线性回归模型的具体参数如表 5 所示.

表 5　演进原则与代码变更量的回归统计表

	Coefficients	P-value
Intercept	2123.54	7.15E-17
主体维持原则	-95.44	1.12E-3
组件精简原则	-48.13	6.47E-06
独立演进原则	-43.70	5.81E-14
平滑演进原则	-1128.24	5.23E-12

如表 5,四项原则均与代码变更量负相关,说明原则达成性越低所需的代码变更量越多,原则达成性越高所需的代码变更量越少,且由于平滑演进原则的回归系数绝对值最大,说明该原则达成性对代码变更量的影响最大.

四项软件架构演进原则达成性均与软件架构演进效果显著相关,其中平滑演进原则与演进成本的关系最显著.

RQ4:在演进过程中,哪些演进活动有利于演进原则的达成?哪些演进活动阻碍了演进原则的达成?

开发者通过各种各样的演进活动达成演进目标[11]. 为了验证各项演进活动对演进原则达成性的影响,本实验以具有详细演进日志的 tablesaw 项目的第 1 至第 23 个版本为实验用例. 将各个版本按发布时间排序,度量结果如表 6,其中 1 号演进过程是第一个版本演进到第二个版本的过程.

表 6　项目 tablesaw 演进原则达成性的度量结果

演进过程	主体稳定原则	组件精简原则	独立演进原则	平滑演进原则
1	0.8799	-0.1271	0.0122	0.1009
2	0.9680	0.2023	-0.0060	0.6911
3	0.7986	-0.2171	0.0537	-0.3445
4	0.8801	-0.2726	-0.0245	-0.6547
5	0.9607	-0.1213	-0.0085	-0.3041
6	0.9868	-0.0057	-0.0047	0.9964
7	0.9953	-0.0210	0.0047	0.7744
8	0.9487	0.1214	0.0131	0.4323
9	0.9767	0.0326	-0.0046	0.9976
10	0.6879	-0.0044	0.0020	-0.2104
11	0.8907	-0.0875	0.0165	-0.6717
12	0.8967	-0.0077	0.0000	0.2104
13	0.6793	0.0121	0.0344	-0.0323
14	0.8999	-0.0068	-0.0232	0.5435
15	0.9363	0.0667	0.0431	0.7122
16	0.9506	-0.0034	-0.0494	0.9984
17	0.8900	0.0210	0.0055	0.9848
18	0.9964	0.0124	-0.0024	-0.2157
19	0.8790	-0.3701	0.0024	-0.3235
20	0.6788	0.3431	0.0213	-0.3337
21	0.8825	-0.0035	-0.0429	-0.0314
22	0.7856	-0.1247	0.0313	0.6102

该项目的演进活动主要包括三类:①功能增加,加入新的功能;②功能改善,完善或提高性能;③故障修复,修复目前存在的错误. 表 7 是基于演进日志提取的演进活动记录,其中 1 号演进过程表示 1 号版本演进到 2 号版本,Y 表示进行了该项演进活动,N 表示没有进行该项活动.

表 7 演进活动记录

演进过程	功能增加	功能改善	故障修复
1	N	Y	Y
2	Y	N	N
3	Y	Y	Y
4	N	N	Y
5	Y	N	Y
6	Y	N	N
7	N	Y	N
8	Y	Y	N
9	Y	N	N
10	N	Y	N
11	N	N	Y
12	N	N	Y
13	N	Y	N
14	N	N	N
15	Y	Y	N
16	Y	N	N
17	Y	Y	N
18	N	N	N
19	N	Y	N
20	N	Y	N
21	N	Y	N
22	Y	Y	Y

我们对演进活动变量进行标准化处理,Y 记作 1,N 记作 0,计算演进活动与演进原则达成性之间的相关系数,如表 8. 为避免各种演进活动相互影响,仅选取只包含一项演进活动的演进过程为例分析演进活动与原则达成性之间的关系.

表 8 演进活动与原则达成性的相关系数

	功能增加	功能完善	故障修复
主体稳定原则	0.3393	-0.5311	-0.0962
组件精简原则	0.1405	0.0016	-0.5158
独立演进原则	0.1480	0.5205	0.2267
平滑演进原则	0.5826	-0.0577	-0.4622

主体稳定原则 第 7 个演进过程仅包含功能完善,其任务目标是提高性能,架构变化说明开发者用新代码替换了原有代码,导致架构构成发生了变化,对主体稳定原则造成了负面影响. 因此,替换原有代码会降低主体稳定,在今后演进中可在原有代码上进行完善促使软件架构主体的稳定.

组件精简原则 第 4 个演进过程仅包含故障修复,其演进任务目标是修复数据遗漏,其架构变化说明开发者完善某组件中的功能,架构整体规模上升但组件时不变,导致组件平均规模上升. 因此,对组件进行大规模扩充不利于达成组件精简原则,在今后演进中可在扩充功能的同时将组件进行进一步的解耦.

独立演进原则 有四个演进过程仅包含功能完善活动,且它们的演进目标是故障修复,其架构变化说明开发者在修复过程中减少了组件间不必要的依赖关系. 因此,功能完善有利于独立演进原则的达成,因此在今后演进中,在实施故障修复时应简化组件间依赖,重点关注于组件内部功能的完善.

平滑演进原则 第 6 个演进过程仅包含功能增加,其演进任务是增加自动识别数据类型的功能,其架构变化说明开发者修改了少量代码. 第 11 个演进过程仅包含故障修复,其演进目标是架构重构,对比其演进前后架构发现,开发者对组件和组件间关联进行了调整,导致软件规模大幅度波动. 通过以上演进过程可知,小规模演进软件架构有利于平滑演进原则达成,故障修复导致的组件及组件间关系剧烈变动不利于平滑演进原则达成,因此在今后演进中,为了保障平滑演进原则达成应逐步演进,避免大规模的变更而导致断代演进.

基于以上分析得出,由于演进活动的内容不同,会对各个演进原则达成性造成不同程度影响,当某个演进原则达成性较低时,应重点关注与其高度相关的演进活动,通过对演进活动的分析发现演进原则达成性较低的原因,避免在今后的演进中再次出现相似问题.

4 相关工作

我们基于数据来源、评估持续演进的方法和演进原则的关注点对比现有的相关工作.

在数据来源方面,部分学者所使用的数据是软件相关的属性,并非软件架构的相关数据[12,13]. 本文方法在数据来源方面采用的是软件架构信息,更贴合软件架构,因此评估结果能有效的说明软件架构的演进情况.

在持续演进的评估方法方面,已有学者通过可演进性衡量软件架构的演进能力[14,15],而可演进性是软件架构所具有的一种质量属性. 本文方法将演进前和演进后的版本相结合,评估出演进后的软件架构是否更有利于持续演进,是软件架构相对的变化,通过对比结果说明软件架构是否持续向好演进.

在演进原则关注点方面,已有学者通过软件架构的设计原则分析软件架构演进效果,但设计原则关注的是演进后的软件架构是否符合软件架构设计[2,3]. 本文方法基于软件架构演进过程中演进对象类别和演进对象粒度提出四项软件架构演进原则,这些原则关注软件架构在演进中所涉及的方面.

以上三个方面的对比分析说明,本文方法在以上

三个方面适用于评估软件架构演进效果.

5　总结与未来工作

本文从演进原则是否达成角度出发来评估软件架构的演进效果,设计和选择了四种代表性的软件架构演进原则,实验结果表明单个演进原则还是四个演进原则结合起来,均能有效反映软件架构演进效果. 本文的实验对象均采用的是 Java 项目,而C ++ 语言具有多继承特性且 C 语言不具有类之间的继承依赖、实例化依赖等,以上语言特性的区别对架构恢复和演进原则达成性的影响需要进一步探讨,同时演进原则的设计和选择本身就是一个难点,如何设计和选择最合适的演进原则成为下一步研究的重点.

参考文献

[1] BRITTO R, SMITE D, DAMM L O. Software architects in large-scale distributed projects: an ericsson case study[J]. IEEE Software, 2016, 33(6): 48 – 55.

[2] WERMELINGER M, YU Y, LOZANO A. Design principles in architectural evolution: a case study[A]. IEEE International Conference on Software Maintenance[C]. US: IEEE, 2008. 396 – 405.

[3] WERMELINGER M, YU Y, LOZANO A, et al. Assessing architectural evolution: a case study[J]. Empirical Software Engineering, 2011, 16(5): 623 – 666.

[4] ALKHALID A, ALSHAYEB M, MAHMOUD S A. Software refactoring at the package level using clustering techniques[J]. Journal of Research & Practice in Information Technology, 2011, 5(3): 276 – 284.

[5] 王映辉,王立福. 软件体系结构演化模型[J]. 电子学报, 2005, 33(8): 1381 – 1386.
WANG Y, WANG L. Research about model and RiPPle effect analysis of software architecture evolution[J]. Acta Electronica Sinica, 2005, 33(8): 1381 – 1386. (in Chinese)

[6] 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.

[7] CAI Y, XIAO L, KAZMAN R, et al. Design rule spaces: a new model for representing and analyzing software architecture[J]. IEEE Transactions on Software Engineering, 2018, PP(99): 1 – 27.

[8] LI T. An approach to modelling software evolution processes[A]. An Approach To Modelling Software Evolution Processes[M]. China: Tsinghua University Press, 2008. 337 – 354.

[9] BJUHR O, SEGELJAKT K, ADDIBPOUR M, et al. Software architecture decoupling at ericsson[A]. IEEE International Conference on Software Architecture Workshops[C]. US: IEEE, 2017. 259 – 262.

[10] COLEMAN D, ASH D, LOWTHER B, et al. Using metrics to evaluate software system maintainability[J]. Computer, 1994, 27(8): 44 – 49.

[11] SUN X, LI B, LI B, et al. SE-FCA: a model of software evolution with formal concept analysis[J]. Chinese Journal of Electronics, 2015, 24(1): 13 – 19

[12] FERNANDEZ-RAMIL J, LOZANO A, WERMELINGER M, et al. Empirical studies of open source evolution[A]. Software Evolution[M]. Berlin: Springer Berlin Heidelberg, 2008. 263 – 288.

[13] GODFREY M, TU Q. Growth, evolution, and structural change in open source software[A]. International Workshop on Principles of Software Evolution[C]. US: ACM, 2001. 103 – 106.

[14] BODE S, RIEBISCH M. Impact evaluation for quality-oriented architectural decisions regarding evolvability[A]. Software Architecture[C]. US: ECSA, 2010. 182 – 197.

[15] BREIVOLD H P, CRNKOVIC I, LARSSON M. Software architecture evolution through evolvability analysis[J]. Journal of Systems & Software, 2012, 85(11): 2574 – 2592.

作者简介

王　桐　女,1990 年生于黑龙江省绥化市. 现为东南大学计算机科学与工程学院博士研究生. 主要研究方向为软件维护与演化.
E-mail: prudens@163.com

廖　力　女,1976 年生于陕西省宝鸡市. 现任东南大学计算机科学与工程学院讲师. 主要研究方向为软件工程,软件维护与演化.
E-mail: lliao@seu.edu.cn