软件适应性度量方法研究 *

柳顺政, 李 琳, 张 莉
(北京航空航天大学 计算机学院 软件工程研究所,北京 100191)

摘 要: 为了能够通过软件的源代码度量面向对象软件的适应性,提出了一个包含七个具体度量指标的软件适应性度量模型,并开发了在 Java 软件中提取各度量指标的支持工具 FlexTool;为了避免通过手动调整大量软件源代码的方式度量软件适应性,使研究人员能够在不太了解软件实现细节的情况下对适应性进行度量,提出了将该适应性模型应用于软件版本的演化从而度量软件适应性的方法,并通过著名开源软件 Ant 的版本演化对该方法的合理性进行了验证。

关键词: 软件适应性; 度量指标; 度量模型; 版本演化
中图分类号: TP311 **文献标志码**: A **文章编号**: 1001-3695(2009)09-3334-04
doi:10.3969/j.issn.1001-3695.2009.09.038

Study of software metrics for software flexibility

LIU Shun-zheng, LI Lin, ZHANG Li
(*Software Engineering Institute*, *College of Computer Science & Engineering*, *Beihang University*, *Beijing* 100191, *China*)

Abstract: In order to measure software flexibility based on object-oriented software's source codes, this paper proposed a flexibility measurement model including 7 concrete measurement factors, and developed a tool which could extract all the measurement factors from a software coded by Java. Besides, in order to prevent researchers from measuring flexibility through changing source codes manually so that they could measure flexibility even though they were not very familiar with software's implementation details, this paper proposed a method which used the measurement model based on the software evolution to measure flexibility. In the end, validated that the measurement model and the method proposed were reasonable by using them to a famous open-source software called Ant.

Key words: software flexibility; measurement factors; measurement model; software evolution

当今计算机技术的快速发展、软件运行环境的不断变化,以及人们对软件的要求越来越多等因素,都要求软件具有更好地适应新变化的能力[1],因此,在软件的设计和实现中,其适应性和灵活性是需要考虑的一个非常重要的因素。软件适应性是指一个系统或构件为适应业务应用和环境的改变(不包括设计时能够确定的改变)进行变化的难易程度[2]。Bengtsson 等人[3]认为软件适应性是为了使软件系统适应需求和环境变化而付出的努力。与软件的稳定性、可操作性等质量属性一样,适应性是软件一个外部质量属性,很难直接测量。因此,如何量化地表示软件的适应性是对其进行度量的关键。

目前对于软件适应性度量的研究,存在以下两方面的问题:a)软件适应性的度量还没有正式的标准,这方面的研究也不多;b)已经提出的一些度量方法需要度量人员对所度量的软件代码实现十分熟悉,并能根据变更需求对该软件实施变更,然而应用这种方法对规模较大的软件进行度量的工作量太大,使实际可操作性不强。

针对以上两个问题,本文提出了一个以面向对象软件源代码作为考察对象的软件适应性度量模型。该模型具有易用而实际可行的特点,并提出一种应用该模型对面向对象软件进行适应性度量的方法。最后对该模型及方法的合理性加以验证。度量软件的适应性本质上是要度量软件变更的难易程度。笔者发现,虽然无法准确预知某个软件未来会发生哪些变更,但引起软件版本演化的变更是研究软件适应性的一个合理来源。由此,针对本文提出的软件适应性度量模型,开发了支持工具 FlexTool,它通过分析以 Java 为开发语言的软件前后两个版本的源代码,对度量模型的各个度量指标给出测量结果。

1 相关研究

目前,软件适应性的度量还没有正式的标准,这方面的研究也不多,另外,已经提出的一些度量方法在实际中的可操作性不强。Eden 等人[1]提出了演化复杂性的观点,并演示了在不同的编程方式(面向对象和结构化编程)、体系结构风格和设计模式中,怎样使用演化复杂性来测量和比较适应性。应用该方法,需要对于软件的各个部分进行语义上和结构上的分析及手动变更,对于大型软件来说,这样做需要巨大的工作量。Tsantalis 等人[4]给出了一种描述面向对象系统适应性度量的模型和方法,他们采用概率的手段来描述变化的不确定性,通过评价当增加一个新的功能或修改一个已有的功能对每一个类可能影响的概率来评估一个面向对象的设计变化的倾向。文献[5]在软件体系结构层次提出了软件适应性的 FCM(fac-

收稿日期: 2008-09-02; **修回日期**: 2009-02-07 **基金项目**: 国家"973"重点基础研究发展规划基金资助项目(2007CB310803);国家自然科学基金资助项目(60773155)

作者简介:柳顺政(1983-),男,甘肃兰州人,硕士研究生,主要研究方向为软件适应性度量(liushunzheng@163.com);李琳(1984-),女,博士研究生,主要研究方向为软件体系结构、软件度量;张莉(1968-),女,教授,博导,主要研究方向为软件工程、需求工程、软件体系结构.

tor-criteria-model)模型,该方法无法应用于从软件源代码度量适应性。有人提出,测量适应性的一种更加精确的方法依赖于对软件变化的影响进行计算的算法或者测量,如用一个针对面向对象系统的变化—影响模型。Chaumun 等人[6]得出了一些实验结果。

2　软件适应性度量模型

2.1　软件变更成本

根据软件适应性的定义[2],其好坏体现在它适应变更的难易程度,所以,适应性和软件的变更是紧密联系的。针对软件变更的原因及变更如何发生,Eden 等人[1]提出了演化步骤的概念,认为软件是对于一个问题集的实现。当这个问题集变化为一个新的问题集时,软件为了实现新的问题集,就需要进行相应的调整,从而发生了变更。借鉴这个观点,以软件的源代码为考察对象,那么软件的源代码就是对一个问题集的实现,当问题集发生变化后,源代码就将作出调整以适应这种变化,如图 1 所示。

基于源代码度量软件适应性,就是要度量源代码发生变更的难易程度。进一步说,就是度量发生变更的成本的多少。软件变更的成本可以从两方面考虑:一方面,它可以是软件变更所消耗的人力、物力和时间等;另一方面,如图 1 所示,它可以是软件原有源代码向变更后的源代码进行调整的难度,如源代码的变更量和变更范围等。在本文中,软件变更成本指的是后者。如果为了适应新的变化,变更源代码的成本很大,也就是说变更非常困难,那么它的适应性就不好;反之,如果软件只需要较小的成本就可以满足新的变化,那么它就具有较好的适应性。因此,基于源代码度量软件的适应性问题就归结为如何度量源代码变更成本的问题。为此,必须考察软件源代码的特点,以及源代码的变更发生在哪些方面。考虑到软件开发技术的发展趋势以及目前大部分软件开发都采用面向对象技术,也为了进行更有针对性的研究,本文将面向对象软件的源代码作为研究对象。

2.2　度量模型

在面向对象软件中,软件元素包括类、方法和代码行等[7]。从源代码的粒度上来讲,类、方法和代码行的粒度从高到低,是三个粒度层次。软件变更在这三个层次上都会产生一定的影响,但影响又不尽相同。所以,单纯考虑类层次,或者方法层次,或者代码行层次的变更是不全面的,本文将对各个层次上的变更成本进行考虑。

另一方面,在类、方法和代码行等不同粒度层次上,软件的变更成本又可以从两方面考虑:a)变更元素数量的多少,即变更的规模大小;b)变更发生在软件的多大范围之内。前者从整体的数量上考察软件变更的难易程度,即认为如果变更量占软件的比例很大,则软件的适应性是比较不好的。后者则是从变更所牵扯的范围这个角度考虑,即认为如果变更导致软件进行范围很大的改动,则软件的适应性是比较不好的,相反,如果变更比较集中,则变更易于实现,从而适应性较好。

概括以上的观点,将度量面向对象软件源代码的变更成本进行归纳,如表 1 所示。

对于面向对象软件变更成本的度量包括类的增加、删除和修改,方法的增加、删除和修改以及代码行的增加、删除和修改。然而,由于各个类、方法之间相互依赖,使得变更之间并不是独立的,而是存在一定的影响关系。本文从整个软件范围内度量变更的成本,则已经将变化的影响计算在内,从而建立了度量软件变更成本的度量模型,如图 2 所示。

表 1　面向对象软件变更成本的度量

度量层次	度量内容
类层次的变更	类的增加、删除和修改
方法层次的变更	方法的增加、删除和修改
代码层次的变更	软件整体代码行的增加、删除、修改;类代码行的增加、删除、修改;方法代码行的增加、删除、修改

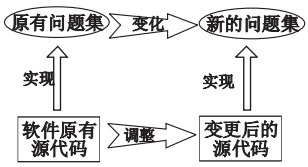

图1　软件演化步骤

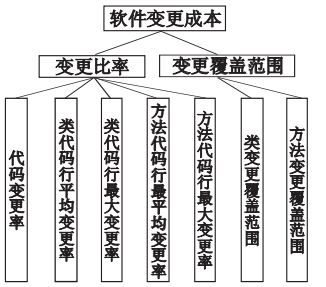

图2　软件变更成本的度量模型

先将软件的变更成本分为变更比率和变更覆盖范围两个考虑方面,然后细分它们各自具体的度量指标:

a)变更比率。它考察由变更引起的软件元素变化的比例,即考察软件源代码的相对改动量是很大的还是较小的。一般而言,大量的改动意味着较大的变更成本,此时认为软件的适应性较差;相反,如果软件变更引起的源代码变动量很小,则变更成本就小,软件的适应性就较好。定义了五个度量指标来量化地表示变化比率,它们分别是代码变更率、类代码行平均变更率、类代码行最大变更率、方法代码行平均变更率和方法代码行最大变更率。这里使用了相对量,从而排除了软件规模的影响。2.3 节将给出这五个度量指标的定义和形式化的计算表达式。

b)变更覆盖范围。它考察软件发生的变更是否集中在少部分软件元素中,即源代码的改动是集中在少部分类或方法中还是涉及到了广泛大量的类和方法。对于前者,因为实现变更所关注的软件元素很集中,所以变更成本更小,此时认为该软件的适应性较好;对于后者,变更的实现需要维护人员理解每个软件元素的含义并顾及各个软件元素之间的依赖关系,这样的变更成本较大,则该软件的适应性较差。定义两个度量来量化地表示变化覆盖范围,即类变更覆盖范围和方法变更覆盖范围。2.3 节将给出这两个度量指标的定义和形式化的计算表达式。

2.3　度量组的定义

在给出各个度量指标的定义之前,先对其中用到的符号及子表达式的含义给出说明:

在以下各定义中,LoC 表示代码行,software 表示某个软件,class 表示类,$class_i$表示某个类,method 表示方法,$method_j$表示某个方法。下标 old 表示原有数量,下标 added 表示增加数量,下标 changed 表示修改数量,下标 removed 表示删除数量。在形如 $LoC_x(y)$ 的表达式中,y 可以是 software 或 $class_i$ 或 $method_j$;x 可以是 old 或 added 或 changed 或 removed,比如,LoC_{added}(software)表示软件增加的代码行数。形如 $class_x$ 或 $method_x$ 的表达式表示类数或方法数,如 $class_{removed}$ 表示删除的

类数。

以下分别给出了图 2 中列出的七个度量指标的定义,并定义了它们的形式化计算表达式。

定义 1　代码变更率 LoC_ChangeRate 度量了软件变更的代码行数占软件变更后代码行总数的比率:

$$\text{LoC_ChangeRate} = \text{变更的代码行数}/\text{变更后代码总行数} \quad (1)$$

其中:变更的代码行数 = $\text{LoC}_{\text{added}}(\text{Software}) + \text{LoC}_{\text{changed}}(\text{software})$,变更后的代码总行数 = $\text{LoC}_{\text{old}}(\text{software}) + \text{LoC}_{\text{added}}(\text{software}) - \text{LoC}_{\text{removed}}(\text{Software})$。

定义 2　类代码行平均变更率 ClassLoC_MeanChangeRate 度量了软件中类的代码变更率的平均值:

$$\text{classLoC_MeanChangeRate} = (\sum_{i=1}^{m} \text{类的 } i \text{ 的代码行变更率})/m \quad (2)$$

其中:$m = \text{class}_{\text{old}} - \text{class}_{\text{removed}}$。类 i 的代码行变更率 classLoC_ChangeRate(class_i)度量了某个类 i 变更的代码行数占变更后该类代码行总数的比率:

$$\text{classLoC_ChangeRate}(\text{class}_i) = (\text{LoC}_{\text{added}}(\text{class}_i) + \text{LoC}_{\text{changed}}(\text{class}_i))/(\text{LoC}_{\text{old}}(\text{class}_i) + \text{LoC}_{\text{added}}(\text{Class}_i) - \text{LoC}_{\text{removed}}(\text{class}_i))$$

定义 3　类代码行最大变更率 classLoC_MaxChangeRate 度量了软件中类的代码变更率的最大值:

$$\text{ClassLo}C\text{_MaxChangeRate} = \max(\{\text{类 } i \text{ 的代码行变更率} \mid 1 \leqslant i \leqslant m\}) \quad (3)$$

其中:类 i 的代码行变更率 classLoC_ChangeRate(class_i)的定义见定义 2。

定义 4　方法代码行平均变更率 methodLoC_MeanChanged-Rate度量了软件中方法的代码变更率的平均值:

$$\text{methodLoC_MeanChangeRate} = (\sum_{j=1}^{n} \text{方法 } j \text{ 的代码行变更率})/n \quad (4)$$

其中:$n = \text{method}_{\text{old}} - \text{method}_{\text{removed}}$。方法 j 的代码行变更率 methodLoC_ChangeRate(method_j)度量了某个方法 j 变更的代码行数占变更后该方法代码行总数的比率:

$$\text{methodLoC_ChangeRate}(\text{method}_j) = (\text{LoC}_{\text{added}}(\text{Method}_j) + \text{LoC}_{\text{changed}}(\text{method}_j))/(\text{LoC}_{\text{old}}(\text{Method}_j) + \text{LoC}_{\text{added}}(\text{method}_j) - \text{LoC}_{\text{removed}}(\text{method}_j))$$

定义 5　方法代码行最大变更率 methodLoC_MaxChange-Rate度量了软件中方法的代码变更率的最大值:

$$\text{methodLoC_MaxChangeRate} = \max(\{\text{方法 } j \text{ 的代码行变更率} \mid 1 \leqslant j \leqslant n\}) \quad (5)$$

其中:方法 j 的代码行变更率 methodLoC_ChangeRate(method_j)的定义见定义 4。

定义 6　类变更覆盖范围 class_ChangeRange 度量了软件中变更的代码在类层次上的集中程度:

$$\text{class_ChangeRate} = \text{代码变更的类数}/\text{变更后的总类数} \quad (6)$$

其中:代码变更的类数 = $\text{class}_{\text{added}} + \text{class}_{\text{changed}}$,变更后的总类数 = $\text{class}_{\text{old}} + \text{class}_{\text{added}} - \text{class}_{\text{removed}}$。

定义 7　方法变更覆盖范围 method_ChangeRange 度量了软件中变更的代码在方法层次上的集中程度:

$$\text{method_ChangeRate} = \text{代码变更的方法数}/\text{变更后的总方法数} \quad (7)$$

其中:代码变更的方法数 = $\text{method}_{\text{added}} + \text{method}_{\text{changed}}$,变更后的总方法数 = $\text{method}_{\text{old}} + \text{method}_{\text{added}} - \text{method}_{\text{removed}}$。

2.4　度量举例:Maze Game

Maze Game 的原型来自于 GoF(Gang of Four)的经典著作[8]。它解决如何灵活地为一个迷宫游戏(Maze Game)构造迷宫(Maze)的问题。对于 Maze Game,下面有两种实现方案:第一种是没有使用设计模式的简单实现,第二种则应用了 Builder 和 Abstract Factory 两个设计模式并作了一些抽象。Builder 设计模式分离了复杂对象的构造和它的表示,Abstract Factory 模式为创建一组相关的对象提供了接口。这两个模式的应用为软件提供了更多的灵活性,因此,基于第二种设计方案的代码实现具有更好的适应性。

本文用 Java 语言实现了这两种设计方案,并由源代码得到了它们的类图。图 3 和 4 分别是应用设计模式并进行抽象前后 Maze Game 的类图。

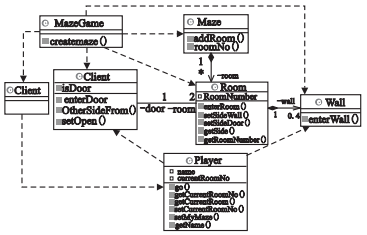

图3　应用Builder和Abstract Factory设计模式前Maze Game的类图

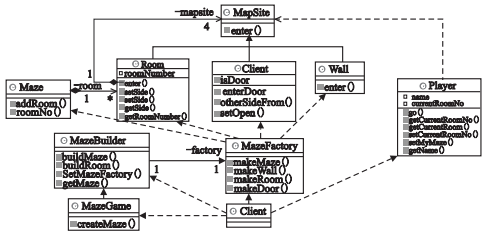

图4　应用Builder和Abstract Factory设计模式后Maze Game的类图

为了使这个迷宫游戏更加丰富,假设在其后的版本中,它需要提供对装有炸弹的(bombedRoom,bombedWall)和具有魔法的房间(enchantedRoom,enchantedDoor)的支持,并可以在某些房间之间用水沟(channel)代替墙壁,如图 5 所示。

对于图 3 和 4 的两种设计,为了将图 5 中的类加入游戏,分别对两种设计施加了变更,并进行了代码实现。对每种实现,使用 FlexTool 分别统计变更前后两个版本代码的变化比率和变化覆盖范围等九个度量指标,如图 6 所示。

从图 6 中可以看出,未使用模式实现的七个变更成本度量指标均高于使用了模式实现的相应度量指标,其中有四个度量指标前者是后者的 2 倍左右,最高的达到 2.73 倍。与使用了模式的实现相比,未使用模式的实现在代码变更率、类代码平均变更率、类代码最大变更率和方法代码变更率方面都较大,这说明未使用模式的实现为了适应变更,需要修改大量的源代码,从而造成其适应性较差,相比之下,使用了模式的实现则要理想得多。在类变更覆盖范围与方法变更覆盖范围这两项度量指标上,使用了模式的实现也比未使用模式的实现略好。因此,由度量模型得到的数据与预期相符,从而反映了度量模型的合理性。

3　由软件版本演化度量软件适应性

在第 2 章中,提出了度量软件变更成本的度量模型,给出了各个度量指标的形式化定义,并用一个实例验证了度量模型的合理性。但是对于一个实际的软件,这种获取它的两种不同的代码实现并根据问题集的变化变更软件源代码,从而通过测

量变更成本度量软件适应性的方法却不易实现。一方面是因为对于每一个待度量适应性的软件,很难获得它的多个不同实现方案以供比较;另一方面,对于大型的软件,针对问题集的每个变化而一一对源代码作出调整(如 Maze Game 示例那样),以此考察其变更成本的方法工作量非常巨大以至于难以实现。

3.1　从软件版本演化度量软件适应性

软件的适应性是由变更的难易程度来反映的,即变更成本,变更成本则由各种可能的变更对软件的综合作用来确定。如果认为一个软件是基于当前的需求而建立的,那么它未来可能的变更可以认为是个不确定因素。一个软件在演化的过程中,可能的变更是很多的,其中一些变更发生的概率很小。考虑所有可能会发生的变更情况是不切实际也是没有必要的,而那些发生概率较大的变更对变更成本起着主要作用。在概率论的统计推断理论中,对于参数估计问题,极大似然估计法认为一个事件在一次观察中出现,应该认为这个事件出现的可能性是很大的[9]。借鉴这个观点,即已经发生的事件具有比较大的概率发生。而软件版本的演化中涉及的变更作为已经发生的变更,可以认为是软件最有可能发生的变更。因此,从软件版本的演化中去研究软件的适应性,是个比较合理的方式。

综上所述,由于软件的版本演化过程中包含了软件最有可能发生的变更,可以通过度量软件版本的变更成本来衡量该软件的适应性。对于开源软件,可以比较容易地得到其源代码在版本间发生的改动,从而研究其变更成本。具体来讲,将第 2 章中建立的变更成本的度量模型应用于软件版本的演化过程,对该演化过程中的每个度量成本指标进行测量。对于一个软件,设它有版本 1 和 2,在由版本 1 向 2 演化的过程中,分别测量其代码变更率、类代码行平均变更率、类代码行最大变更率、方法代码行平均变更率、方法代码行最大变更率、类变更覆盖范围和方法变更覆盖范围,由式(8)来得到变更成本的度量:

$$\text{变更成本} = \sum_{i=0}^{5} a_i \times F_i \tag{8}$$

其中:F_i表示度量指标 i,F_1表示代码变更率,F_2表示类代码行平均变更率,F_3表示方法代码行平均变更率,F_4表示类变更覆盖范围,F_5表示方法变更覆盖范围;a_i是 F_i的权值,它可以根据研究者的侧重点不同而进行合适的选取,这里简单地取 $a_1 = a_2 = a_3 = a_4 = a_5 = 0.2$,即变更成本是以上五个度量指标值的平均。类代码行最大变更率和方法代码行最大变更率可以作为变更成本的额外参考指标。如果变更成本很小,则说明该软件的适应性较好;反之,如果变更成本很大,则该软件的适应性较差。

3.2　实例验证及分析:Ant 的版本演化

以开源软件 Ant 的版本演化来对本文度量模型,以及第 2 章提出的通过版本演化度量软件适应性的方法加以验证。

Ant 是 Apache 组织开发的一种基于 Java 的著名构建工具[10],具有优秀的跨平台性和操作上的简易性,并被集成到了众多的集成开发环境中。作为软件的一个重要质量属性,一般期望随着版本的演化和软件的不断改善,软件的适应性越来越好[11]。因此,期望 Ant 的适应性在版本的演化过程中不断增强。

收集了 Ant 软件的从 1.1 到 1.7.0 的 17 个版本,并通过适应性分析工具 FlexTool,度量了软件变更成本度量模型中的七个度量指标。本文挑选了软件变更较大的八个版本演化进行比较,为了简便,以下将其编号为版本演化 1 ~ 8,它们按照演化发生的时间依次往后。引起 Ant 变更的主要因素有对以往特性的改变、增加功能和修改 Bug 等。一般而言,与其他变更相比,修改 Bug 对软件的影响较小。另外,很难确定各个变更原因对软件的变更具体产生多大的影响,因此,假设除修改 Bug 外的变更原因对软件的影响都相同,将每个版本的变化原因数定义为 0.4 × 对以往特性的改变数 + 0.4 × 增加功能数 + 0.2 × 修改 Bug 数,得到了各个度量指标相对于变更原因的平均值,如表 2 所示。其中类最大代码变更率和方法最大代码变更率不需要进行平均处理。

表 2　Ant 各版本软件变更成本度量

度量指标	演化 1	演化 2	演化 3	演化 4	演化 5	演化 6	演化 7	演化 8
代码变更率	0.3927	0.4294	0.2686	0.5654	0.2396	0.0444	0.0394	0.2396
类平均代码变更率	0.1817	0.3940	0.0946	0.3535	0.0877	0.0198	0.0206	0.0816
类最大代码变更率	1.0000	1.0000	0.9167	1.0000	0.7264	0.5490	0.9444	1.0000
方法平均代码变更率	0.1283	0.3511	0.0620	0.3377	0.0583	0.0111	0.0197	0.0650
方法最大代码变更率	0.9250	0.9333	0.9643	1.0000	1.0000	0.9722	1.0000	1.0000
类变更率	0.7778	0.9762	0.6444	0.9291	0.6714	0.2610	0.2473	0.6576
方法变更率	0.6436	0.7835	0.4431	0.8258	0.4145	0.0981	0.1289	0.4226

对于每个版本演化,本文对表 2 中的数据进行了归一化处理(类最大代码变更率和方法最大代码变更率除外),即以表格中每一个单元格的数据除以相应列的最大值作为其归一化后的值;然后通过归一化后的数据,根据式(8)得到变更成本。因为变更成本较大意味着适应性较差,所以,以变更成本的倒数来表示适应性,并将各个版本适应性的变化趋势绘制成折线图,如图 7 所示。

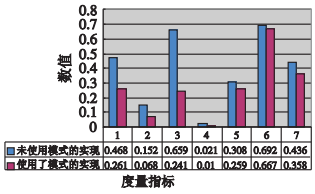

图6　未使用模式的简单实现与使用模式的实现的变更成本比较图

图中,度量指标1表示代码变更率, 2表示类代码行平均变更率, 3表示类代码行最大变更率, 4表示方法代码行平均变更率, 5表示方法代码行最大变更率, 6表示类变更覆盖范围, 7表示方法变更覆盖范围。

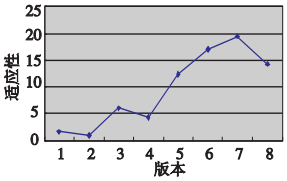

图7　Ant的适应性随版本演化变化趋势图

从图 7 中可以看到,用本文度量模型得到的 Ant 的适应性随着其版本的演化总体上呈现出不断变好的趋势。当然也注意到,并非每次变更反映出的软件适应性都得到提高,如通过版本演化 4 测量的适应性比通过版本演化 3 测量的适应性略低,这是符合直观认识的。因为软件版本的每次演化并非只致力于提高其适应性等质量属性,而是由于各种原因造成的,这些综合因素影响的结果使得软件适应性并非随着版本的演化单调地提高,但其总体趋势却是不断变好的。实验的结果与笔者的预期相符,即 Ant 的适应性在版本的演化过程中不断增强。

4　FlexTool 介绍

支持工具 FlexTool 实现了对软件变更成本模型中七个度量指标的自动计算。它以软件变更前后两个版本的源代码作为输入,经过预处理、变更分析、源代码比较和结果计算等五个主要分析步骤,最终得到代码变更率、类代码行平均变更率、类代码行最大变更率、方法代码行平均变更率、方(下转第 3344 页)

```
- 〈outputPort group = "yes" 〉
  〈name〉DecapOut〈/name〉
  〈synopsis〉Ethernet decapsulation output. 〈/synopsis〉
-〈product〉
  ……
```

3 结束语

国内外对路由器为代表的关键网络设备的研究,已经从单纯追求提高设备性能发展到同步研究提高性能和扩展功能,这是实现多网融合、业务融合的迫切需要。本文提出了可重构路由单元中软件架构的初步技术方案。这些方案是建立在对软件构件研究的基础上,围绕建立在面向新型网络应用的可重构性、可复用性研究后形成的,并对方案路由器软件部分中软件构件的生成、控制、加载等作了研究。后期将围绕提高可重构路由单元性能,将部分软件构件功能硬件化及对相关标准的制定进行更多详细研究。

参考文献:

[1] HIDELL M, SJÖDIN P, HAGSAND O. Control and forwarding plane interaction in distributed routers[C]//Proc of the 4th International IFIP-TC6 Networking Conference. Waterloo: Springer, 2005: 1339-1342.

[2] TAKOUROUT R N, RIERRE S, MARCHAND L. Separation of the control plane and forwarding plane in next-generation routers[J]. Journal of Computer Science, 2006,2(11):815-823.

[3] Juniper networks delivers industry's first high-performance control plane scaling platform[EB/OL]. (2008-02-25). http://www.juniper.net/company/ presscenter/pr/2008/pr-080225.html.

[4] HALEPLIDIS E, HAAS R, DENAZIS S, *et al*. A Web service- and ForCES-based programmable router architecture[C]//Proc of International Working Conference on Active and Programmable Netwoks. Berlin:Springer,2005:108-120.

[5] SCHUBA C, GOLDSCHMIDT J, KALAJAN K, *et al*. Integrated network service processing using network devices[EB/OL]. (2005-05). http://research. sun. com/techrep/2005/ smli_tr-2005-138. pdf.

[6] FEAMSTER N, BALAKRISHNAN H, REXFORD J, *et al*. The case for separating routing from routers[EB/OL]. (2004). http://www.research. att. com/ ~kobus/ docs/rcp-fdna. pdf.

[7] LI YUN-chen,CHIANG Mei-ling. lyraNET:a zero-copy TCP/IP protocal stack for embedded operation systems[C]//Proc of the 11th IEEE International Conference on Embeded and Real_time Computing Systems and Applications. Washington DC:IEEE Computer Society, 2005:123-128.

[8] COMER D E. 用 TCP/IP 进行网际互联,第 1 卷:原理、协议与结构[M].4 版.北京:电子工业出版社,2001:128-130.

[9] ZHAO Feng, SU jin-shu, CHENG Xiao-mei. OpenRouter: a TCP-based lightweight protocol for control plane and forwarding plane communication[C]//Proc of the 3rd International Conference on Networking and Mobile Computing. Berlin, Heidelberg: Springer, 2005: 1032-1041.

[10] 王宝生,夏毅,陈晓梅,等.基于 ForCES 体系结构的 IPv6 路由器的研究与实现[J].国防科技大学学报,2006,28(3):44-48.

[11] IYER S, MCKEOWN N W. Analysis of the parallel packet switch architecture[J]. IEEE/ACM Trans on Networking, 2003,11(2): 314-324.

(上接第 3337 页)法代码行最大变更率、类变更覆盖范围、方法变更覆盖范围七个变更成本度量指标,大大方便了数据的收集过程。其中:预处理模块删除了源代码中存在的注释和空行;变更分析模块分别构造了软件前后两个版本的抽象语法树,并据此对源代码中类、方法的变更点和相同点进行分析和记录;源代码比较模块根据变更分析模块得到的中间结果,生成源代码比较文件,并调用文件比较工具对源代码进行比较;结果计算模块综合变更分析模块和源代码比较模块得到的中间结果,计算出七个变更成本度量指标。本文中 Maze Game 实例和 Ant 实例的变更成本数据均通过 FlexTool 获得。

5 结束语

已有的软件适应性度量方法由于过于依赖度量人员对于软件源代码的熟悉程度,从而变得不具有很好的实际操作性。根据软件适应性的定义并在借鉴了前人研究的基础上,提出以软件的变更成本来表征软件的适应性,由此提出了软件变更成本的度量模型,并对其中的每个度量指标的计算方法进行了定义。通过 Maze Game 这个实例,验证了度量模型的合理性。然后,提出应用本文度量模型,以考察软件版本的演化来对适应性进行度量,并论述了度量方法。最后,通过对著名开源软件 Ant 的版本演化进行数据分析,验证了度量模型以及度量方法的合理性。另外,本文度量数据的提取均通过本文开发的 FlexTool 支持工具对源代码的分析得到。

参考文献:

[1] EDEN A H, MENS T. Measuring software flexibility[J]. Software IEE Proceedings, 2006, 153(3):113-125.

[2] IEEE 610.12-1990 standard glossary of software engineering terminology[S]. Los Alamitos: IEEE Press, 1999.

[3] BENGTSSON P, LASSING N, BOSCH J, *et al*. Architecture-level modifiability analysis (ALMA)[J]. The Journal of Systems and Software, 2004, 69(1-2):129-147.

[4] TSANTALIS N, CHATZIGEORGIOU A, STEPHANIDES G. Predicting the probability of change in object-oriented systems[J]. IEEE Trans on Software Engineering, 2005, 31(7):601-604.

[5] 高晖,张莉.软件体系结构层次的适应性度量技术研究[J].计算机科学, 2008, 35(4):51-57.

[6] CHAUMUN M A, KABAILI H, KELLER R K, *et al*. A change impact model for changeability assessment in object-oriented software systems[C]//Proc of the 3rd European Conference on Software Maintenance and Reengineering. Washington: IEEE Press, 1999: 130-138.

[7] ECKEL B. Java 编程思想[M].陈昊鹏,译.北京:机械工业出版社,2007:354-355.

[8] GAMMA E, HELM R, JOHNSON R, *et al*. Design patterns elements of reusable object-oriented software[M].[S.l.]:Addison Wesley, 1995:46-52.

[9] 韩於羹.应用数理统计[M].北京:北京航空航天大学出版社, 2004:132-133.

[10] The Apache Software Foundation. Apache Ant introduction[EB/OL]. (2006-11-6)[2008-07-04]. http://ant.apache.org.

[11] BANSIYA J, DAVIS C G. A hierarchical model for object-oriented design quality assessment[J]. IEEE Trans on Software Engineering, 2002, 28(1):4-17.