基于细粒度构件的关联度量*)

戴立伟[1,2] 张为群[1,2]
(西南大学计算机与信息科学学院 重庆 400715)[1]
(重庆市智能软件与软件工程重点实验室 重庆 400715)[2]

摘 要 介绍了基于多构件软件的稳定性度量的一种模式,包括对其中各个构件及其类型的度量,并且利用该模式替换关键构件,找出合理可重用的构件组。

关键词 软件度量模式,稳定性,风险管理,构件提取

A Metrics Suite for Measuring the Relationship between Software Fine-grained Components

DAI Li-Wei[1,2] ZHANG Wei-Qun[1,2]
(College of Computer and Information Science, Southwest University, Chongqing 400715)[1]
(Chongqing Intelligent Software and Softwre Engineering Laboratory, Chongqing 400715)[2]

Abstract Introduce a model to measure the relationship between software fine-grained components. Using this model, the relationship and the key component can be measured. And the reasonable components can also be detected.

Keywords Model of metrics, Software stability, Risk management, Components

1 引言

基于构件的软件开发已经越来越广泛地应用起来,并且实践证明是一种行之有效的工业化方法。基于构件的软件开发强调利用已有的构件对软件进行设计和构建,这是降低软件成本、提高软件可靠性的很好的手段。从狭义上讲,构件可以看成是整个软件的组成模块,一种可以用来读取和定制的模块。然而,这种模块对我们来讲,大多是封装好的,不知道源代码的。

随着构件在市场上的大量应用,有效度量构件的不同特性变得越来越重要。在现有的一些软件度量手段中,例如Chidamber&Kemerer提出的C&K度量[1]、Abreu提出的MOOD度量[2]等,也有其相应的度量手段,但是我们可以看到这些手段都是基于源代码的,对于我们现有的基于构件的软件是不适用的,还有基于构件接口的研究也有少量的一些[3]。另外,我们可以看到在现有的软件开发中,构件是独立发布的封装体,其内部实现对外不可见,其接口是构件与其他构件以及周围环境进行交互的惟一方式[4],这样在不清楚构件源码的情况下,构件之间的相互影响已经变得日益突出,如果没有控制好这种影响的话,可能会带来不可预知的后果。本文就是基于这种情况,提出一种基于多构件软件的稳定性度量的一种模式,包括对其中各个构件及其类型的度量,并且利用该模式替换关键构件,找出合理可重用的构件组。

2 软件构件

软件构件是指应用系统中可以明确辨识的构成成分[3],是一种已经定制好的模块,它利用自身的接口和外界交流。对于使用者而言,构件是已经封装好的有特定功能的程序,而其内部具体的实现是我们无法得知的,就像大家熟知的黑箱一样。

2.1 构件的粒度

构件的粒度可以分为粗粒度、中等粒度、细粒度[5]。对于粗粒度的构件来讲,往往含有非常复杂的业务逻辑。而中等粒度的构件是由细粒度的构件组成的,其中含有一些不复杂却特殊的业务逻辑。细粒度构件往往是有一些RAD工具提供的,它具有很清晰的逻辑结构。

2.2 细粒度构件

细粒度构件的广泛应用得益于RAD工具的成功,比如大家熟知的ActiveX[6]和JavaBean0[7]等都可以被看成细粒度的构件。这种细粒度的构件往往有以下重要特性:

属性:是细粒度软件的基本定义,它的值可以用来读取和存写。那些可以用来读取的属性,称之为"可读属性";而那些可以用来存写的属性,称之为"可写属性"。

读取方法:是在构件内部实现的,向外部提供的读取内部可读属性的方法,该方法是构件的读接口。

存写方法:是在构件内部实现的,向外部提供的存写内部可写属性的方法,该方法是构件的写接口。

业务逻辑方法:是在构件内部实现的,除去读取方法和存写方法外的方法,它用来实现构件的业务逻辑。

接下来我们就利用细粒度构件的这几个重要特性,给出一个判断基于构件开发的软件稳定性的方法。

3 构件关联度量

软件在构件的基础上的定制,对于软件的快速开发,降低软件成本是非常有意义的,但是在现有的软件开发中可以看到许多产品是非常脆弱的,经常会出现由于定制环境不同而

*)本论文是重庆市科委自然科学基金重点项目:软件测试技术与方法研究;重庆市信息产业发展基金项目:软件过程度量与质量度量综合平台建设(200401021)以及重庆市科委自然科学基金项目:软件非功能性需求的多模型测试与外部度量(CSTC2004BB0146)的部分成果。**戴立伟** 硕士,主要研究领域为面向对象技术,软件度量,软件建模;**张为群** 教授,主要研究领域为形式化软件工程,软件测试。

无法使用的情况，所以软件稳定性和适应性在软件开发和维护过程中非常重要，它对于降低软件开发风险有着十分重大的意义。

在现有的软件开发中，构件被大量地应用，而构件之间的交互也变得非常的频繁，一个构件往往需要从另外一些构件中获取参数，甚至获得业务逻辑结果。这说明一个构件往往要关联于其他构件才能起到作用。但是一个构件如果太关联于其他的构件，也就是说该构件会关联到其他许多构件，这样是十分不稳定的。良好的稳定性被认为是有着尽可能少的关联的[2]，由此我们提出了一种度量多构件软件的稳定性的模式。

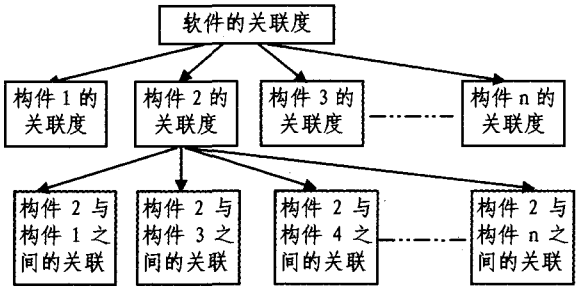

图 1　基于多构件软件的稳定性度量的模式图

3.1　构件关联模型

我们现在从构件使用者的角度定义一个基于多构件软件的构件关联模型。

一个基于构件的软件的稳定性来自于各个构件的稳定，如果各个构件都非常的稳定，在不同的环境下都能保持原有的功能，那毫无疑问，这个软件也会是非常稳定的。但是在现实中，这种情况是极少发生的。即使对于一些非常成熟的构件，在由我们定制的不同的情况下也可能出现丢失功能的情况。比如：在 Web 开发中的非常成熟的一个构件：displayTag，在 MyEclipse 开发中设定由 action 进入页面的时候，其分页功能就会丧失。而恰恰其他很多构件都是关联于该构件的，导致这些构件都无法使用，使整个软件瘫痪。

很多因素都会影响软件的稳定性，但是在我们这个模型中，我们从构件之间的关联来度量基于构件的软件稳定性。在不考虑其他因素的情况下，构件之间的关联越强，软件的稳定性就越差，软件失效的风险就越大。

3.2　软件关联度定义

首先，我们分析一下构件之间的关联性。我们已经知道构件是通过以上三种方法：读取方法、存写方法、业务逻辑方法，来完成相互交流的。而构件之间的关联就是由这些方法来决定的。

定义 1（当前构件集合）　该度量定义了当前软件中所有构件的集合。

$C \in C$，其中 C 表示当前软件中所有构件的集合。

定义 2（可读方法集合）　该度量定义了构件中所有可读方法的集合。

$M_R(c)$，其中 $c \in C$，$M_R(c)$，表示当前构件的所有可读方法的集合。

定义 3（可读方法数总和）　该度量定义了构件中所有可读方法数目的总和。

$S_R(C)$，其中 C 表示当前构件集合，$S_R(C)$，表示当前构件所有可读方法数目的总和。

定义 4（存取方法集合）　该度量定义了构件中所有存取方法的集合。

$M_W(c)$，其中 $c \in C$，$M_W(c)$，表示当前构件的所有存取方法的集合。

定义 5（存取方法数总和）　该度量定义了构件中所有存取方法数目的总和。

$S_W(C)$，其中 C 表示当前构件集合，$S_W(C)$，表示当前构件所有存取方法数目的总和。

定义 6（业务逻辑方法集合）　该度量定义了构件中所有业务逻辑方法的集合。

$M_B(c)$，其中 $c \in C$，$M_B(c)$，表示当前构件的所有业务逻辑方法的集合。

定义 7（业务逻辑方法数总和）　该度量定义了构件中所有业务逻辑方法数目的总和。

$S_B(C)$，其中 C 表示当前构件集合，$S_B(C)$，表示当前构件所有业务逻辑方法数目的总和。

定义 8（构件交流方法集合）　该度量定义了两个构件之间交流所有方法的集合。

$M(a,b)$，其中 $a \in C, b \in C$，$M(a,b)$，表示当前两个构件之间交流所有方法的集合。

定义 9（可读方法关联）　该度量定义了两个构件是否通过某一个可读方法产生了关联。

$$RMD(a,b,m)=\begin{cases}1, & \text{if}(a\in C \text{ and } b\in C), m\in M_R(b)\\ 0, & \text{otherwise}\end{cases}$$

该情况表示构件 a 通过可读方法 m，对构件 b 产生了可读方法关联。

定义 10（存取方法关联）　该度量定义了两个构件是否通过某一个存取方法产生了关联。

$$WMD(b,a,m)=\begin{cases}1, & \text{if}(a\in C \text{ and } b\in C), m\in M_W(a)\\ 0, & \text{otherwise}\end{cases}$$

该情况表示构件 a 通过存取方法 m，对构件 b 产生了存取方法关联。

定义 11（业务逻辑方法关联）　该度量定义了两个构件是否通过某一个业务逻辑方法产生了关联。

$$BMD(a,b,m)=\begin{cases}1, & \text{if}(a\in C \text{ and } b\in C), m\in M_B(b)\\ 0, & \text{otherwise}\end{cases}$$

该情况表示构件 a 通过存取方法 m，对构件 b 产生了业务逻辑方法关联。

3.3　构件方法关联分析

以上我们对软件关联度中的基本概念进行了定义，在这些定义的基础上，有关软件关联度的核心，也就是如何进行度量的分析将进一步展开。

两个构件之间可以通过某一个方法产生可读方法关联，但是究竟这两个构件之间通过多少个可读方法产生了该关联呢？这显然对我们度量构件间的关联有着非常重要的意义，由此我们利用以下方法度量构件之间的可读关联：

$RD(a,b)=\sum_{i=1}^{n} RMD(a,b,m)$，if($a \in C$ and $b \in C$), $m \in M_R(b)$

其中，$RD(a,b)$ 表示构件 a 通过多少个可读方法 m，对构件 b 产生可读方法关联。

那么同样的方法，我们可以利用以下方法度量构件之间的存取关联和业务逻辑关联：

$WD(b,a)=\sum_{i=1}^{n} WMD(b,a,m)$ ，if($a \in C$ and $b \in C$), $m \in M_W(a)$

$BD(a,b)=\sum_{i=1}^{n}BMD(a,b,m)$,if($a\in C$ and $b\in C$),$m\in M_B(b)$

其中,$WD(b,a)$表示构件 a 通过多少个存取方法 m,对构件 b 产生存取方法关联,$BD(a,b)$表示构件 a 通过多少个业务逻辑方法 m,对构件 b 产生业务逻辑方法关联。

从以上的分析我们可以得出两个构件之间的方法关联,在这里可以很直观地观察到构件之间通过哪种方法、多少个这样的方法产生了关联,这对于我们把握两个构件之间的关联是非常有用的,但是这三种方法关联的度量结果都是量化成个数的,而不是一个百分比,所以我们很难观测到两个构件之间方法度量的确切比重。

3.4 构件方法关联度分析

如果不知道构件中各个方法的关联在本方法中所占的比率,就无法知道这两个构件的方法关联是不是在此种方法的关联中比重。在这里我们在构件中某种所有方法的总和的基础上计算方法关联的比率。以可读方法关联为例,可读关联度为可读关联在构件所有可读方法数目的总和所占比率:

$RDP(a,b)=\sum_{i=1}^{n}RMD(a,b,m)/\ S_R(C)$,if($a\in C$ and $b\in C$),$m\in M_R(b)$

其中,$RDP(a,b)$表示构件 a 通过可读方法 m,对构件 b 产生可读关联度。

那么同样的方法,我们可以利用以下方法度量构件之间的存取关联和业务逻辑关联:

$WDP(b,a)=\sum_{i=1}^{n}WMD(b,a,m)/S_W(C)$,if($a\in C$ and $b\in C$),$m\in M_W(b)$

$BDP(a,b)=\sum_{i=1}^{n}BMD(a,b,m)/S_B(C)$,if($a\in C$ and $b\in C$),$m\in M_B(b)$

其中,$WDP(b,a)$表示构件 a 通过存取方法 m,对构件 b 产生存取关联度,$BDP(a,b)$表示构件 a 通过业务逻辑方法 m,对构件 b 产生业务逻辑关联度。

以上三个方法关联度定量地反映了可重用构件在不知道内部代码的情况下主要是通过哪种方法与其他构件关联,关联的程度又是多少。但是从更抽象的角度去看,一种两个构件总体之间的关联度则更为需要。通过前面的定义和分析,我们可以得出两个构件各个方法的关联,我们在此基础上观测到两个构件总体之间的方法关联。但是有个情况需要特别注意,那就是业务逻辑方法的关联性要远远大于可读方法和存取方法的关联性,所以使用一个因子 k 来协调比重。

$\mu(a,b)=(1-k)\sum_{i=1}^{m_r}RMD(a,b,m_r)+(1-k)\sum_{i=1}^{m_w}WMD(b,a,m_w)+k\sum_{i=1}^{m_b}BMD(a,b,m_b)$,if($a\in C$ and $b\in C$),$m_r\in M_R(b)$,$m_w\in M_W(b)$,$m_b\in M_B(b)$

从 μ 的构成来看,它的取值区间是$[0,\infty)$,这对于我们度量构件之间的关联度是非常不利的,所以我们利用下面的方法将其映射到$(0,1]$区间:

$$BCP(a,b)=\lim_{x\to\mu\varepsilon}\frac{\varepsilon}{\varepsilon\cdot x+1}$$

从以上分析,我们就得到了一个位于$(0,1]$的两个构件 a 和 b 之间关联度 $BCP(a,b)$,因为区间在$(0,1]$之间,我们就可以很好地反映出两个构件之间的关联比重。

3.5 构件关联度分析

通过构件方法关联度分析,我们可以得出单个构件和所有其他构件之间的关联,这样我们就可以对单个构件进行评价:

$CP(a)=\sum_{i=1}^{n}BCP(a,b_i)/n$, if($a\in C$ and $b_i\in C$),

其中,n 表示 C 中除了 a 之外的其他所有构件的数目。

当然在现有的软件中是不止有一个构件的,这样如果要知道这个软件中构件直接的关联程度,我们就可以利用各个单个构件的关联度来计算整个软件的关联度。同时,在这里我们要注意一个事实:各个构件的重要程度是不同的,所以我们利用一个重要性因子定义各个构件的重要性是很有必要的。

$S=k_1*CP(a_1)+k_2*CP(a_2)+\cdots\cdots+k_i*CP(a_i)+\cdots\cdots+k_n*CP(a_n)$,if($a_i\in C$,$k_1+k_2+\cdots\cdots+k_i+\cdots\cdots+k_n=1$),$n$ 表示 C 中所有构件的数目。

通过以上的定义和计算,我们就可以得到软件的所有构件的关联度 S,很显然 $S\in(0,1)$,同时我们可以看到,S 越小,该软件内部构件之间的关联就越小。在不清楚构件源码的情况下,构件之间的相互影响就可以很好地度量出来,在进行风险评估尤其是在引入新技术的时候就可以作为很好的参考,避免带来不可预知的后果。

4 候选关键构件

很显然,关键构件的失效对于整个软件是致命的,所以在我们设计之初就应该提供一个备用的构件,也就是在软件开发过程中做 DAR,在适当的时候可以替代到该构件。这个备用构件要求能够完成关键构件的功能,并且尽量保持接口一致。

Zaremski 和 Wing 提出了构件替换的准则,如图 2。其中,Plug-in 这种方式提出了最严格的满足条件:(i)任何合法的问题领域的输入都是该构件的一个合法输入;(ii)该构件的任何合法输出都是问题领域的一个合法输出,即$(I_P\Rightarrow I_c)\wedge(O_C\Rightarrow O_P)$。同时,一种限制稍弱的 Weak Plug-in 的方式也被提出:(i)任何合法的问题领域的输入都是该构件的一个合法输入;(ii)该构件的输入导出的任何合法输出都是问题领域的一个合法输出,即$(I_P\Rightarrow I_c)\wedge(I_c\wedge O_C\Rightarrow O_P)$。

文[6]中已经证明,只要满足 Weak Plug-in 的方式,就可以用该构件来解决当前的问题。那对应于我们对关键构件的替换,关键构件的输入就是问题领域的输入,关键构件的输出,就是问题领域的输出,而备用构件如果满足 Weak Plug-in 的方式就可以对关键构件进行替换,而不会造成对整个软件的影响了。

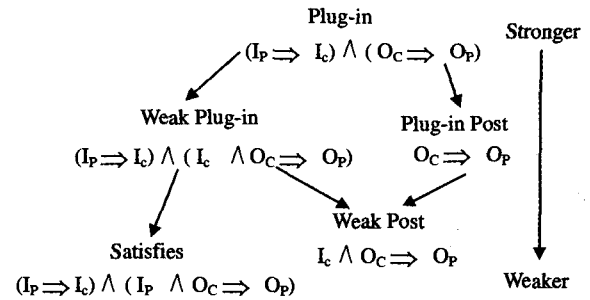

图 2 Lattice of satisfaction criteria as defined by Zaremski and Wing

当然,这是我们的理想情况,但是,在实际软件开发过程当中,很有可能无法满足这样的条件,我们在使用备份构件的同时,其关联的构件也会进行变动。如果变动导致整个软件关联度增加过大,至少会造成软件开发负荷突然增加过大,那么显然这样一个候选构件是不合适的,下面我们通过定义以

下公式来度量候选构件对整个软件系统的影响：

$$\text{BackupQuality}=(1-\frac{CP(bc)-CP(kc)}{CP(kc)})\times 100\%$$

其中，bc 表示候选构件，而 kc 表示关键构件。

很显然，如果 BackupQuality 过大的话，我们将重新考虑做 DAR，所以我们在为关键构件进行候选构件的选取的时候必须满足这样的条件：

$BC=\{bc|(I_{kc}\Rightarrow I_{bc})\wedge(I_{kc}\Delta O_{kc}\Rightarrow O_{kc})$OR $satisfy$ (BackupQuality)$\}$

5 提取构件组

通过前面构件之间的度量，我们可以得出单个构件和所有其他构件之间的关联 CP，很显然我们会得到一个 CP 值最大的构件，我们可以关联度上认为该构件为关键构件，也就是说这个构件于其他构件发生的关系最多，一旦此构件出现问题，我们软件遭受致命性损害的可能性也就最大。保证这个构件运行良好，是我们在规避以后软件开发中发生重大风险的很好的手段。

在我们度量构件之间关联度的同时，我们可以看到一张以构件为点，构件之间的关联度为边的构件关联图的形成。

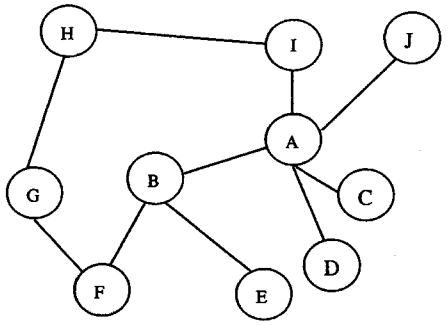

图 3 构件关联图

在图 3 中，我们可以看到边是由之前所度量出来的 BCP 来构成的，比如 A 和 B 的边，也就是 BCP(A,B)，那么在这样一个构造出来的图中，我们一定就可以找到关键构件，利用该关键构件我们可以自动构造出一个构件组，这个构件组可以在软件 SQA 审核之后重用，具体这个构件组识别的算法如下：

算法描述：

```
CS Identification (Graph g)
begin
C kc=getMax(g);
//C 表示构件，以上是获取关键构件
CS cs=getInitialCSByRadom(kc);
//从 kc 的关联构件随机选取一个构件作为初始构件组
float sumcs=sumExtenal(cs);
//获取 cs 中与其他构件关联总和
CS cstemp=getComponentByRadom(cs);
//获取与 cs 相关联的一个构件，将其加入构件组临时集合 cstemp
float sumcstemp=sumExtenal(cstemp);
//计算 cstemp 与其他构件关联的总和，将其存入一个变量 sumcstemp
While(sumcstemp<sumcs)
begin
cs=cstemp;
sumcs=sumExtenal(cs);
CS cstemp=getComponentByRadom(cs);
sumcstemp=sumExtenal(cstemp);
end
return cs;
end
```

通过以上的算法我们可以找出构件组 CS，通过 SQA 的审核之后，这个构件组就可以拿来进行重用了，当然可以在原图中去处 CS 集合的成员，继续寻找新的构件组。

同样，我们可以度量构件组的关联度 S，如果 S 越小，我们就认为该构件组越稳定。

结论 在基于构件的软件开发应用越来越广泛的情况下，软件的稳定性更是应该得到十分的注意。通过本文的方法，我们不仅可以看出软件的稳定性，对软件的风险分析和制定有很好的辅助作用，同时我们还可以看出软件对哪个模块最有关联，最具有哪种类型的关联，在软件设计时，需要对哪个模块进行良好分析和设计。当软件产生问题的时候，可以迅速对问题模块进行替换。运用该方法可以找出合理可重用的构件组，可以对现有的风险制定给予补充和支持，更有力地辅助项目过程的实施。

参 考 文 献

1 Chidamber R, Kemerer F. A Metrics Suite for Object-Oriented Design. IEEE Trans. Softw. Eng., 1994,20(6):476～479

2 Harrison R,Counsell SJ, Nithi RV. An Evaluation of the MOOD Set of Object-Oriented Software Metrics[J]. IEEE Trans. Softw. Eng.,1998,24(6):491～496

3 杨芙清，梅宏，李克勤. 软件复用与软件构件技术. 电子学报，1999,27(2):68～75

4 Szyperski C. Component Software: Beyond Object Oriented Programming. Harlow: Addison-Wesley Longman, Inc., 1997. 1～70

5 Washizaki H, Yamamoto H, Fukazawa Y. A Metrics Suite for Measuring Reusability of Software Components 1530-1435/03 2003 IEEE

6 Denning A. ActiveX Controls Inside Out. Microsoft Press, 1997

7 Hamilton G. JavaBeans Specification 1.01, Sun Microsystems, 1997

8 Penix J, Alexander P. Using Formal Speci_cations for Component Retrieval and Reuse 1060-3425/98 1998 IEEE

(上接第 263 页)

效的方法，大大提高了软件应用的灵活性和可扩展性。通过在某通信设备制造商归属位置寄存器开发中的应用，较好地满足了设计的需求，缩短了开发业务的周期，具有较好的灵活性，取得了不错的应用价值。

参 考 文 献

1 Aho, A V, Sethi R. Compilers: Principles, Techniques, And Tools[J]. Addison-Wesley, Reading, MA, 1986

2 Lee, Peter. Topics in Advanced Language Implementation, MIT Press, Cambridge, MA, 1991

3 Aho A V, Sethi R, Ullman J D. 李建中译. 编译原理[M]. 北京：机械工业出版社，2005

4 Levine J R, Mason T, Brown D. 杨作梅译. lex 与 yacc(第二版)[M]. 北京：机械工业出版社，2003

5 IBM 中国技术支持. 电子商务运行环境模型[EB/OL]. http://www-900.ibm.com/cn/support/guide/blueebook4_1.shtml. 2005-10-8

6 郑先容，黄杰. 基于 CORBA 构件模型的编译器的研究与实现[J]. 计算机应用，2005(25):91～92