doi: 10.3969/j.issn.1005-3026.2015.12.006

基于复杂网络的类间集成测试序列生成方法

赵玉丽, 王 莹, 于 海, 朱志良
(东北大学 软件学院, 辽宁 沈阳 110819)

摘 要: 类间集成测试是面向对象软件测试的一个重要部分. 将类抽象成节点,类间调用关系抽象成边,建立面向对象软件的复杂网络模型,根据软件网络的结构特征,分析类节点的影响力和复杂性,给出一种软件重要节点的度量方法. 在此基础上,将错误被尽早发现的思想应用在类间集成测试排序中,在确保构造的测试桩复杂度较小的前提下,保证重要节点优先测试,得到了一种基于复杂网络的类间集成测试序列生成方法. 最后,在开源软件 DNS 1.2.0 上进行分析和验证,证明了该方法的有效性.

关 键 词: 集成测试;复杂网络;节点重要性;测试序列;桩复杂度
中图分类号: TP 311.5 **文献标志码**: A **文章编号**: 1005-3026(2015)12-1696-05

An Inter-Class Integration Test Order Generation Method Based on Complex Networks

ZHAO Yu-li, *WANG Ying*, *YU Hai*, *ZHU Zhi-liang*
(School of Software, Northeastern University, Shenyang 110819, China. Corresponding author: YU Hai, E-mail: yuhai@126.com)

Abstract: Inter-class integration test is a critical part of the object-oriented software testing. Representing a class by a node in a complex network, two classes are connected only if there exist an invoke relationship between them. Then, a complex network model representing object-oriented software could be constructed. According to the characteristics of the complex network structure, the influence and complexity of each class node was analyzed. Further, an approach for measure the significant nodes in the software network were provided. Moreover, the idea that error should be found as soon as possible was applied to the inter-class integration test order, an inter-class integration test order generation method was proposed, in which the test priority of the significant nodes and the lower test stub complexity were ensured. Simulation result on the open source software DNS 1.2.0 indicated the effectiveness of the proposed method.

Key words: integration test; complex network; node significance; test order; stub complexity

面向对象软件由于具有封装性、多态性和继承性等特点,在软件测试技术上与传统的面向过程软件有很大区别. 其中,类的集成测试是在类单元测试完成的基础上,对类集成过程的测试[1]. 由于类的测试顺序关系到软件缺陷发现的时间和设计测试桩造成的测试成本,类的集成测试顺序成为集成测试中的一个重要问题[2].

目前,针对类间集成测试顺序的生成方法主要有两类:1) 基于遗传算法的类间集成测试顺序生成方法. 该方法首先初始化一个种群代表类的测试顺序,然后对其进行选择、交叉、变异操作,从而得到一个满足预定条件的类间集成测试顺序[3];2) 基于图论的启发式类间集成测试顺序生成算法. 该方法从类中提取依赖关系,依据打破最多环路、不同类型边的权重等标准移除有环结构的关联边,对类图进行拓扑排序,从而得到类间集成测试顺序[4].

类间集成测试顺序问题在国内的研究相对较

收稿日期: 2014-11-01
基金项目: 国家自然科学基金资助项目(61202085, 61374178, 61402092); 中央高校基本科研业务费专项资金资助项目(N130317001, N130417004).
作者简介: 赵玉丽(1985-),女,内蒙古赤峰人,东北大学讲师,博士; 朱志良(1962-),男,辽宁沈阳人,东北大学教授,博士生导师.

少. 陈建勋等分析了对象关系图中的类间依赖关系，运用边删除规则去除环路，通过有向无环图的拓扑序列给出类的测试顺序，有效减少了测试桩的数量[5]. 柴玉梅等对 UML 模型的顺序图添加了对象约束语言，给出一种测试路径生成算法[6]. 姜淑娟等以降低测试桩复杂度为优化目标，提出一种基于耦合度量的类间集成测试顺序的确定方法[7]. 上述方法在解决类间测试排序问题时，仅仅考虑了减少测试桩数目和降低测试桩复杂度两个方面，忽略了重要的类和错误传播概率较大的类应尽早被测试的问题.

软件系统具有复杂网络特性，运用复杂网络的统计方法对软件整体分析度量的研究[8-9]已经成为软件工程领域最为活跃的一个分支. 本文将类抽象为节点，类之间的依赖关系抽象为边，构造面向对象软件系统的类级软件网络，提出一种类间集成测试序列生成方法，综合考虑类的复杂性及软件网络中类的依赖关系，给出类重要性度量指标；设计基于重要节点与测试桩复杂度的打破环路的有效算法，得到合理的类间集成测试顺序，在保证测试桩复杂度较小的前提下，使得重要类优先测试.

1　基于复杂网络的类间集成测试序列生成方法

在软件系统中，80% 的功能仅被 20% 的类所承担，20% 的类在软件系统中起着决定性的作用，应该被重点测试. 因此，如何在软件系统中找到 20% 的重要类节点是一个亟待解决的问题[10].

本文根据面向对象软件系统中类间的调用关系建立软件网络结构，当且仅当类 B 调用了类 A 的属性或方法时，网络中存在一条由类 B 指向类 A 的连边.

1.1　软件重要测试节点度量方法

在软件网络中，不仅类之间存在复杂性，同时类内部也拥有自己的属性和方法，类本身也具有一定的复杂性. 本节首先综合评估了软件类本身的复杂性及其对其他类的影响、类的复用性等因素，定义了两个软件网络重要节点的度量指标：影响因子和复杂因子.

通常情况下，类的入度越大，说明越多的类依赖于该类，即该类的影响力较大. 另外，一个类通过某个类调用第三个类在面向对象软件中也是屡见不鲜的，所以，在考虑类的影响力时，应该同时考虑类的直接依赖影响和间接依赖影响.

定义 1　特征集合 $F(A)$ 定义为类 A 的所有属性和方法所在的集合.

定义 2　可达特征集合 $\mathrm{RF}(A,B)$ 定义为类 A 中能够由类 B 中的特征直接或间接调用的特征集合.

$$\mathrm{RF}(A,B) = \mathrm{RF_D}(A,B) \cup \mathrm{RF_I}(A,B). \quad (1)$$

其中：$\mathrm{RF_D}(A, B)$ 表示类 B 直接依赖类 A 的特征集合；$\mathrm{RF_I}(A, B)$ 表示类 B 间接依赖类 A 的特征集合.

定义 3　类 A 的影响因子定义为

$$\mathrm{IF}(A) = \sum_{i=1}^{N-1} \mathrm{IF}(A,i). \quad (2)$$

其中：$i \in V, i \neq A$；$\mathrm{IF}(A,i) = \dfrac{|\mathrm{RF}(A,i)|}{|\mathrm{F}(A)|}$ 为类 A 对类 i 的影响因子；N 为软件网络的规模；V 为软件网络中类节点所在的集合. 图 1 所示为类 $A \sim G$ 共 7 个类的调用关系构成的网络结构. 类 A 中有 2 个属性和 2 个方法，即 4 个特征. 类 D 直接调用类 A 中的属性 a_2，类 D 通过类 B 间接调用类 A 中的属性 a_1，那么类 A 对类 D 的影响因子为 2/4.

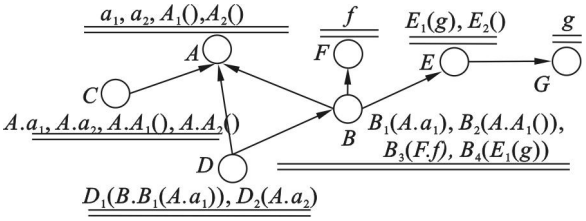

图 1　7 个类的调用关系构成的软件网络

Fig. 1　Software network constructed according to the invocation relationship among 7 classes

定义 4　类的复杂因子 CF 定义为

$$\mathrm{CF}(A) = \alpha \cdot |F(A)| + \beta \cdot \sum_{i=1}^{N-1} \frac{|\mathrm{RF}(i,A)|}{d(i,A)}. \quad (3)$$

其中：$i \in V, i \neq A, \alpha + \beta = 1$；$d(i,A)$ 表示类 A 和类 i 的平均距离，即，从类 A 经过多少层调用到达类 i. 复杂因子随着调用层次的增加而递减，可见，错误的传播概率与类间的距离成反比.

定义 5　节点重要度. 综合考虑类的影响因子和复杂因子得到类的节点重要度为

$$W(i) = \sigma \cdot \mathrm{IF}(i)' + \delta \cdot \mathrm{CF}(i)'. \quad (4)$$

其中：$\sigma + \delta = 1$；$\mathrm{IF}(i)' = \dfrac{\mathrm{IF}(i) - \mathrm{IF_{min}}}{\mathrm{IF_{max}} - \mathrm{IF_{min}}}$；$\mathrm{CF}(i)' = \dfrac{\mathrm{CF}(i) - \mathrm{CF_{min}}}{\mathrm{CF_{max}} - \mathrm{CF_{min}}}$. $\mathrm{IF_{max}}$，$\mathrm{IF_{min}}$，$\mathrm{CF_{max}}$ 和 $\mathrm{CF_{min}}$ 分别表示整个网络中节点影响因子和复杂因子的最大值和最小值.

1.2 集成测试序列生成方法

在集成测试中,如果测试类 A 时,类 B 尚未被测试,且短时间内很难构建出类 B,此时需要为类 A 模拟一个类 B 的对象. 这种用来开发或测试一个调用或者依赖桩的组件被称为测试桩[10].

构建测试桩的代价由桩复杂度估算. 本文引用 Briand 等提出的估算测试桩复杂度的方法[11],其计算方法如式(5)所示.

$$\mathrm{SCplx}(i,j) = (W_A \cdot \overline{A(i,j)}^2 + W_M \cdot \overline{M(i,j)}^2)^{1/2}. \tag{5}$$

其中:$\mathrm{SCplx}(i, j)$ 表示移除边 $e(i,j)$ 表示的依赖关系,为类 i 构造测试桩 j 的复杂度;W_A 和 W_M 表示权重,$W_A + W_M = 1$;$A(i, j)$ 表示类 i 调用类 j 的属性的个数;$M(i, j)$ 表示类 i 调用类 j 的方法的个数. $\overline{A(i,j)}$ 和 $\overline{M(i,j)}$ 分别表示归一化后的结果.

类的测试顺序需要保证每个类在测试前,其依赖的类已经测试完毕,从而确保尽量少的测试桩和集成测试的完整性. 生成的类测试顺序的目标在于保证较小的测试桩复杂度,同时满足重要节点被优先测试. 以此为基础,本文提出一种基于重要节点与测试桩复杂度的破环算法. 首先使用 Tarjan 算法深度优先遍历有向图 G 中的每一个节点,找到 G 中的所有强连通分量 SCC;由强连通分量的定义可知,有向图的环路一定存在于 SCC 中. 因此,只需要在每个 SCC 中查找环路,然后将环路中的每条边按照式(6)赋予权值,进行删边操作.

$$W_{e(i,j)} = \frac{W(i)}{\mathrm{SCplx}(i,j)}. \tag{6}$$

其中:$W(i)$ 表示节点 i 的重要度;$\mathrm{SCplx}(i, j)$ 表示删除了边 $e(i, j)$ 后,需要给节点 i 构建测试桩的复杂度.

在删除边时,遵循以下四个原则:1) 当环路中边权值最大的边只有一条时,删除环路中边权值最高的边;2) 如果边权值最大的边不止一条,且每条边的测试桩复杂度均不等,则删除其中测试桩复杂度最小的边;3) 如果边权值最大的边不止一条,且这几条边的测试桩复杂度也相等,那么删除其中节点重要度值最大的节点的出边;4) 直到有向图中再无环路时,删边结束.

定义网络中任意一条从入度为 0 出度不为 0 的起始节点出发,到达出度为 0 入度不为 0 的终端节点的所有路径为最长依赖路径 $\mathrm{Path_D}$. 除孤立节点外,节点处于最长路径中的位置由变量依赖深度 Depth 描述. 假设任意一条最长依赖路径的终端节点的依赖深度等于 1,那么该路径上任意节点 i 的依赖深度等于节点 i 到终端节点的距离 d_i 加 1. 通常经过非孤立节点 i 的最长依赖路径有 N_i 条($N_i \geqslant 1$),$\mathrm{Depth_{max}}$ 表示节点 i 在所有最长依赖路径中依赖深度的最大值.

基于上述定义,分析无环网络数据,得到类间集成测试的顺序,主要过程为:1) 过滤孤立节点,计算剩余每个节点的最大依赖深度 $\mathrm{Depth_{max}}$;2) 将节点按照 $\mathrm{Depth_{max}}$ 升序排序,对于其中存在两个或多个 $\mathrm{Depth_{max}}$ 值相同的节点按照其重要度值排序;3) 孤立节点随机插入到已排序列的前面;4) 输出集成测试序列 O.

2 结果与讨论

为了验证本文提出方法的有效性,将其应用于软件 DNS 1.2.0,得到该软件的类间集成测试顺序,并将其与文献[11]所得结果进行对比分析.

2.1 软件重要节点实验分析

根据类间依赖关系,DNS 1.2.0 的软件网络结构如图 2 所示.

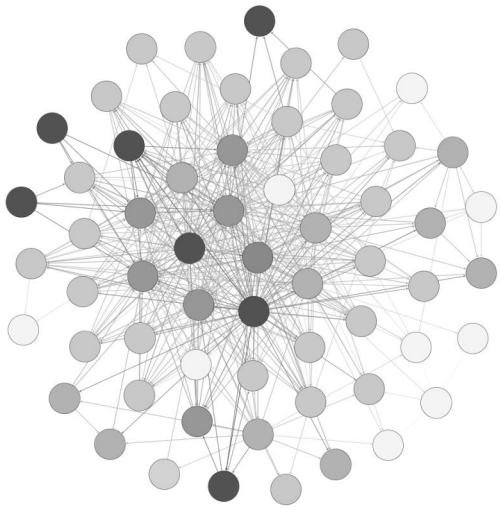

图 2 DNS 1.2.0 软件的网络结构

Fig. 2 The complex network of DNS 1.2.0

以图 2 为基础,采用式(4)可得 DNS 1.2.0 的 61 个类的节点重要度,如图 3 所示. 由图 3 可知,DNS 1.2.0 软件的类节点重要度值区间为[0.029 6, 0.513 4],其中,重要度值超过 0.3 的节点有 7 个;重要度值在 0.2 以下的节点有 48 个,约占总节点数的 78.7%. 少数重要度值较高的类节点具有较高的复杂性或较高的重用概率,成为软件的核心类节点. 错误发生在这些核心节点上所引起的严重性将远远高于其他节点引起的严重性,因此,这些少数重要度值较高的节点应该

尽早被测试.

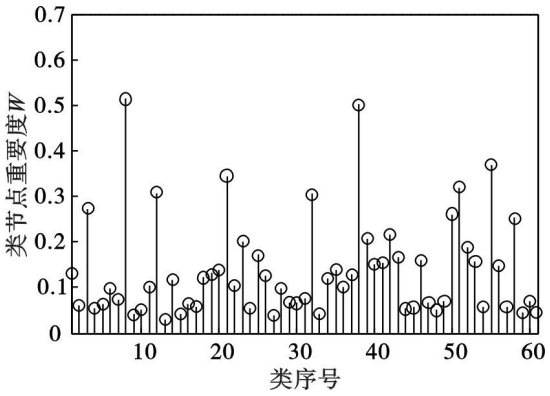

图 3　DNS 1. 2. 0 的类节点重要度
Fig. 3　The significance of each class node in DNS 1. 2. 0

表 1 列出了重要度值排名前 5 和后 5 的类节点的特征总数 Φ, 入度 d_{in}, 特征被调用次数 t, 出度 d_{out}, 调用其他类特征次数 φ 等统计特性.

表 1　类节点统计特性
Table 1　The statistics characteristics of class nodes

排名	序号	Φ	d_{in}	t	d_{out}	φ	IF	CF	W
1	8	5	29	329	1	3	1. 000	0. 03	0. 513 4
2	38	26	2	8	18	75	0. 007	1. 00	0. 503 5
3	55	38	9	79	12	106	0. 062	0. 68	0. 369 0
4	21	33	40	135	5	48	0. 300	0. 39	0. 345 3
5	51	34	1	49	17	157	0. 011	0. 63	0. 319 9
57	13	7	3	4	0	0	0. 039	0. 02	0. 029 6
58	27	8	2	3	0	0	0. 046	0. 03	0. 039 5
59	9	13	3	11	0	0	0. 042	0. 04	0. 041 1
60	33	6	2	2	5	14	0. 030	0. 05	0. 042 0
61	15	12	4	4	0	0	0. 038	0. 05	0. 042 5

分析表 1 可知, 重要度排名前 5 的类节点均具有较高的被调用次数或调用其他类特征的次数, 相应地这些类也拥有较高的复杂因子或影响因子, 进一步验证了重要度值大的节点具有较高的复杂性和影响力, 应该尽早测试. 而重要度排名后 5 的节点, 缺乏由于调用其他类特征产生的复杂性, 并且调用该类的特征次数也较少, 因此, 这些类发生错误传播到软件系统中其他类的概率相对较小, 具有较低的测试优先级.

2. 2　生成类间集成测试序列

进一步应用本文提出算法对软件网络中存在的环路进行破环操作, 并按类的最大依赖路径深度及重要度值排序, 给出 DNS 1. 2. 0 软件的类间集成测试序列.

DNS 1. 2. 0 软件网络包含两个 SCC, 16 个环路. 假设式(5)中 $W_A = W_M = 0.5$, 破环过程如表 2 所示, 其中 NC 为删边前 SCC 中包含的环路个数.

表 2　SCC 破环过程
Table 2　The process of breaking in SCC

序号	处理的 SCC	NC	删边	$W(i)$	$W_{e(i,j)}$	SCplx (i,j)
1	SCC_1	16	21→11	0. 345	8. 422	0. 041 0
2	{58,48,32,	6	21→8	0. 345	2. 468	0. 139 9
3	25,11,8,	5	32→48	0. 305	2. 182	0. 139 9
4	21}	4	32→58	0. 305	0. 502	0. 607 7
5	SCC_2	3	38→33	0. 504	1. 860	0. 270 7
6	{33,38,52}	1	52→33	0. 188	0. 695	0. 270 7
7	End	0		—		

破环结束后, 可得一个有向无环软件网络. 根据网络拓扑结构, 计算每个类节点的最大依赖深度, 并结合类节点的重要度, 得到类间集成测试的顺序, 如图 4 所示.

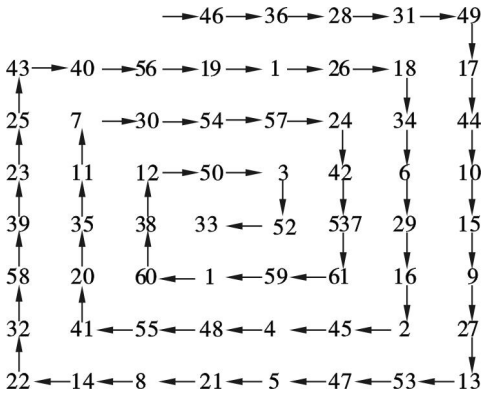

图 4　类间集成测试顺序
Fig. 4　The class integration test order

根据式(5)计算可知, 本文提出方法的平均桩复杂度为 0. 245, 总测试桩复杂度为 1. 47. 在相同参数条件下, 本文提出算法与文献[11]提出算法的平均桩复杂度和总测试桩复杂度相同. 另外, 本文得到的测试序列, 当测试完节点总数的 50% 时, 14 个重要节点已经测试完毕, 而文献[11]附录 E8 给出的 10 条测试序列中, 在测试完成 50% 时, 重要度排名前 20 的节点, 平均只测试完成了 6. 7 个重要节点. 由此可见, 本文提出算法的生成测试序列在桩复杂度不变的前提下, 使得重要节点能够优先测试.

3 结 论

本文运用复杂网络理论对面向对象软件的类间依赖进行建模,根据类间调用关系及类本身的特征,给出类节点重要度的衡量指标. 并以此为基础,设计了一种综合考虑桩复杂度和节点重要性的破环算法,从而得到依据节点最大依赖深度及节点重要度的类间集成测试顺序. 本文方法在保证桩复杂度较小的前提下,使得软件中的重要类节点优先被测试.

参考文献:

[1] Jorgensen P C. Software testing: a craftsman's approach [M]. Boca Raton: CRC Press Inc, 2013.

[2] Abdurazik A, Offutt J. Using coupling-based weights for the class integration and test order problem [J]. *Computer Journal*, 2009, 52(5): 557 – 570.

[3] Jorgensen P C, Erickson C. Object-oriented integration testing [J]. *Communications of the ACM*, 1994, 37(9): 30 – 38.

[4] Le Traon Y, Jeron T, Jezequel J M, et al. Efficient object-oriented integration and regression testing [J]. *IEEE Transactions on Reliability*, 2000, 49(1): 12 – 25.

[5] 陈建勋,肖亦然. 基于动态依赖的类间测试顺序研究[J]. 传感技术学报, 2014, 27(1): 64 – 69.
(Chen Jian-xun, Xiao Yi-ran. Class-integration testing sequence research based on dynamic dependency[J]. *Chinese Journal of Sensors and Actuators*, 2014, 27(1): 64 – 69.)

[6] 柴玉梅,冯秋燕,王黎明. 基于 UML 模型和 OCL 约束的类间交互测试用例生成方法研究[J]. 电子学报, 2013, 41(6): 1242 – 1248.
(Chai Yu-mei, Feng Qiu-yan, Wang Li-ming. Research on methods for generating test cases of inter-classes interaction based on UML models and OCL constraints [J]. *Acta Electronica Sinica*, 2013, 41(6): 1242 – 1248.)

[7] 姜淑娟,张艳梅,李海洋,等. 一种基于耦合度量的类间集成测试序的确定方法[J]. 计算机学报, 2011, 34(6): 1062 – 1074.
(Jiang Shu-juan, Zhang Yan-mei, Li Hai-yang, et al. An approach for inter-class integration test order determination based on coupling measures [J]. *Chinese Journal of Computers*, 2011, 34(6): 1062 – 1074.)

[8] Concas G, Marchesi M, Pinna S, et al. Power-laws in a large object-oriented software system [J]. *IEEE Transactions on Software Engineering*, 2007, 33(10): 687 – 708.

[9] Valverde S, Sole R V. Network motifs in computational graphs: a case study in software architecture [J]. *Physical Review E*, 2005, 72(2): 026107.

[10] Ammann P, Offutt J. Introduction to software testing [M]. Cambridge: Cambridge University Press, 2008.

[11] Briand L C, Feng J, Labiche Y. Software engineering with computational intelligence [M]. New York: Springer, 2003.