分 类 号 TP 311　　密　级
U D C　　编　号 10486

武　汉　大　学
博　士　学　位　论　文

基于软件网络的软件静态结构分析及应用研究

研　究　生　姓　名：　潘伟丰

指导教师姓名、职称：　李　兵　教授

学 科 专 业 名 称：　计算机软件与理论

研　究　方　向：　软件工程

二〇一一年 五月

Software Networks-Based Analysis of Software Static Structure and Its Applications

A Dissertation

Submitted to Wuhan University

in Partial Fullment of the Requirements for

the Degree of Doctor of Philosophy

in Computer Science

By Weifeng Pan

SUPERVISED BY PROF. BING LI

WUHAN UNIVERSITY

May, 2011

郑 重 声 明

本人的学位论文是在导师指导下独立撰写并完成的,学位论文没有剽窃、抄袭、造假等违反学术道德、学术规范和侵权行为,否则,本人愿意承担由此而产生的法律责任和法律后果,特此郑重声明.

学位论文作者(签名):

年　　月　　日

论文的主要创新点

通过软件工程与复杂系统的学科交叉研究，以软件网络为研究对象，从复杂网络的角度认识软件结构及其动力学行为，有助于我们深入理解和认识软件的本质，从而开辟一条软件工程学研究的新途径，并为软件工程实践提供指导。本文工作的创新点主要体现在以下两个方面：

（1）**方法创新**：将复杂网络相关理论引入软件工程中，构建软件系统不同粒度的网络模型——软件网络，进而从一个较高的层次抽象和研究软件系统的静态结构，探索软件的结构特性及其演化规律。同时，建立软件内部结构属性与外部质量属性（稳定性）的联系，并通过软件结构的度量指导软件工程实践。

（2）**应用创新**：以软件度量为研究取向，践行软件网络观在软件工程中的价值，验证了基于软件网络的软件度量方法在软件过程的设计、构造、测试和维护等各阶段中的应用。具体包括：首次提出了软件在特征（方法和属性）粒度的加权软件网络模型，分析了变更在该网络上的传播过程，并借以评价软件的稳定性，并基于稳定性的度量实现软件结构优选；首次提出了软件在特征粒度的软件网络模型（属性方法网络和方法耦合网络），并通过节点走查和对软件网络社区结构的分析指导软件类重构；首次提出了软件在类粒度的加权软件网络模型，并通过该网络分析各类对软件测试的重要性，进而对软件回归测试的测试用例进行排序。

摘　　要

软件系统已经深入到了人们生活的方方面面，发挥着日益重要的作用。应用领域的不断拓展加之人们对软件功能和质量需求的不断提高，使得软件的规模和复杂性不断增长，软件质量无法得到保障。因此，如何认识、度量、管理、控制乃至降低软件复杂性，是软件工程学亟待解决的一个重要问题。软件结构被认为是影响软件复杂性的重要因素之一。因此，要对软件系统的复杂性进行深入、细致的研究，就必须对软件的结构信息进行合理的描述和有效的量化。

传统的结构度量方法（面向过程/对象度量方法）更多的考虑的是软件元素的属性度量（如有多少个类、方法等），但是对于软件结构本身包含的复杂信息还没有充分地挖掘。由于缺少相应的方法和理论，研究人员很少从整体的角度来审视软件的结构及其演化，导致对软件的本质缺乏清晰的认识。

近年来，一些研究者将复杂网络相关理论引入软件工程中，构建软件的复杂网络（软件网络）描述软件系统的骨架，从较高层次描述系统的结构，为我们研究软件结构及其动力学行为提供了有力的工具。但是，软件网络的研究目前仍处于起步阶段。纵观软件网络 8 年多的发展，软件网络的研究已经取得了一些可喜的成果，但是仍存在下面的一些问题: 缺乏从不同粒度和演化的角度来分析和理解软件的组织结构、结构的演变和由此产生的行为特征; 缺乏软件整体结构特性与软件质量（稳定性、可靠性、可维护性等）之间关系的认识; 缺乏软件结构度量如何指导工程实践（软件重构、软件测试等）的研究。

针对上述问题，本文基于软件的网络结构，采取软件工程和复杂网络相关理论相结合的研究方法，以分析软件结构特征为着眼点，以理解软件的结构特性，为软件开发和维护提供支持为目标，以目前应用最广泛的面向对象软件为研究对象，从表征、分析、度量和应用四个方面来分析和研究软件静态结构，具体的研究内容和取得的成果包括:

（1）软件多粒度演化分析 (**详见博士期间发表论文 3、4 和 5)**

针对现有软件演化分析中的不足，提出了多粒度软件网络模型，从方法、类和包三个粒度抽象面向对象软件结构，并采用复杂网络的理论与方法，从时间（多版本演化）和空间（方法、类和包三个粒度）两个角度系统的研究软件系统结构特征及其演化规律。通过真实面向对象软件的实证研究揭示了面向对象软件在结构上的一些特征和演化规

律，并比较了不同粒度软件网络的异同。这些规律的发现可以辅助理解软件系统设计的演化过程及趋势，并指导软件项目管理、成本预测以及软件测试、软件维护等工作。同时，对于构建软件演化模型也具有重要的指导意义。

（2）软件稳定性分析及结构优选 (**详见博士期间发表论文 6**)

软件稳定性对软件维护和质量的评估都有重要影响。针对现有软件稳定性研究中存在的问题，提出加权特征依赖网络模型抽象面向对象软件的结构，提出了一种仿真的方法分析变更在该软件网络上的传播过程，并定义软件稳定性指标评价软件稳定性。理论和实证研究验证了度量指标的有效性，同时还表明，本文的方法可以实现结构优选；随着软件元素间的耦合增强，软件的稳定性不断下降。这些结论对决策者作出正确的决策具有指导意义，同时也为软件结构设计和质量评估提供了理论依据。

（3）面向对象软件类重构 (**详见博士期间发表论文 2**)

我们将复杂网络社区发现算法应用于面向对象软件的类重构，识别方法移动的重构。构建面向对象软件系统在特征（方法和属性）粒度的网络模型，借鉴复杂网络社区发现相关研究成果，在尊重设计模式的前提下优化软件的类结构，并通过对比优化前后软件类结构和节点走查的方法发现软件的重构点。实例研究验证了本文方法的有效性。该方法可以为开发者实施重构工作提供指导。

（4）面向对象软件回归测试用例排序 (**详见博士期间发表论文 1 和 7**)

从软件复杂性和静态结构出发，对面向对象软件的回归测试用例排序技术进行了研究。提出加权类依赖网络模型抽象类粒度面向对象软件系统，通过分析类的复杂性来度量类的错误倾向性，通过仿真方法评价类产生错误后对系统的影响大小，并在此基础上提出类测试关注度衡量类的测试重要性。最后根据测试用例覆盖的类的测试关注度总和对测试用例进行排序。实例研究验证了本文方法的有效性。该方法可以为实施软件回归测试工作提供支持。

以软件网络为研究对象，从复杂网络的角度认识软件结构及其动力学行为，有助于我们深入理解和认识软件的本质，从而开辟一条软件工程学研究的新途径，并为软件工程实践提供指导。

关键词：软件结构；复杂网络；重构；测试；复杂性

ABSTRACT

Software has been used in every walk of life, playing increasingly important role. The ever-increasing expansion of applications and users' requirements makes a steep rise in the scale and complexity of software, which results in the decrease in the software quality. So it is a great challenge in software engineering to understand, measure, manage, control, and even to low the software complexity. Software structure is one of the factors influencing software quality. So if we want to deeply investigate the software complexity, the information enclosed in the structure should be effectively measured.

The traditional structural metrics (*i.e.*, process- and object-oriented (OO) software metrics) mainly focus on the attributes of software elements, *e.g.*, the number of classes, the number of methods, *etc.* But they fail to deeply explore the rich information in software topological structure. Due to the lack of suitable tools and theories, people seldom investigate the software structure as a whole, making themselves be in dark about the nature of software.

Complex network theory provides an effective tool to study the system structure and its dynamics. In recent years, a few researchers introduced the complex network theory to software engineering domain, and used complex networks in software, hereafter referred to as software networks, to represent software structure from a higher level. Based on software networks, a lot of work have been carried out to study the software structure and its dynamics. By reviewing the research work on software networks, we find there are still some problems: lack of work to analyze the structure organization, structure evolution, and the corresponding behavior properties from a multi-granularity and evolution perspective; lack of work to uncover the relationships between the software structure as a whole and the attributes of software quality such as stability, reliability and maintainability; and lack of work to use structure metrics to support software engineering practices such as software refactoring and software testing.

To address the above-mentioned problems, this dissertation, based on software networks, takes an integration way of software engineering and complex network theory to study the software static structure, with characterizing the software structure as its focus, and understanding structure properties and supporting software engineering practices as its objectives. The static structure of software will be explored from four aspects, representation, analysis, measurement and application. The primary contents and achievements of this dissertation can be summarized as:

(1) Multi-granularity evolution analysis of software

Considering the shortcomings of existing software evolution analysis work, we pro-

posed a multi-granularity software networks model to represent the software structure from method, class, and package level, and used complex network theory to analyze the evolution of OO software. The proposed method characterized the software structure and its evolution law from time (evolution) and space (multi-granularity) dimensions. An empirical study conducted on real world OO software have disclosed the underlying evolution dynamics and laws existing in software structures. All these discoveries can help us understand the software design and its evolution, and support the software project management, cost prediction, software testing, software maintenance, etc.

(2) Software stability analysis and optimal structure selection

The stability of a software system is one of the most important quality attributes affecting the maintenance effort. Considering the shortcomings of existing software stability analysis work, in this dissertation we presented a novel metric to measure the stability of OO software by software change propagation analysis using a simulation way in software networks at feature (*i.e.*, methods and attributes) level. The effectiveness of the proposed metric is analytically and empirically validated. Empirical studies also showed that the proposed metric can be used to select the optimal software from a bunch of similar software systems and the increase in the coupling of software elements will decrease the stability of software. These discoveries can help a lot in decision-making and provide a theoretical basis for designing software and measuring software quality.

(3) Class refactoring of OO software

Considering the shortcomings of existing software refactoring methods such as complex and resource-consuming, in this dissertation, we presented an approach to recondition the class structures of OO software, *i.e.*, to detect the 'move method' refactorings. The proposed approach used software networks at feature level to represent the software structure, and used a guided community detection algorithm while respecting design patterns to obtain the optimized class structure. By comparing the class structure before and after optimization, and node inspection, a list of refactorings will be provided. The proposed method can serve to help software developers in performing refactoring task.

(4) Test case prioritization for regression testing of OO software

This dissertation proposed a new test case prioritization technique, which takes into consideration software complexity and software structure properties that have not been used in previous work. It used weighted software networks at class level to represent OO software, defined some metrics to describe bug proneness of classes, bug severity and finally the testing importance of classes. Empirical studies on real world OO software systems showed its effectiveness. The proposed method provide a new way to do software testing and can serve to help software developers in testing software.

Based on the software network model, using complex network theory to study the software structure and its dynamics will deepen our understanding of the nature of the software complexity, provide a new insight into software engineering, and support our software engineering practices.

Key words: Software structure; Complex networks; Refactoring; Software testing; Complexity

目　　录

摘要 **i**

ABSTRACT **iii**

目录 **I**

第一章　绪论 **1**

1.1　研究背景 1

1.1.1　软件复杂性问题 1

1.1.2　软件度量的重要性 3

1.1.3　结构度量的重要性 4

1.1.4　复杂网络的研究 4

1.1.5　软件网络的研究 6

1.2　问题的提出及意义 7

1.3　主要工作内容及研究体系 8

1.4　论文的组织结构 10

第二章　相关研究 **11**

2.1　引言 11

2.2　传统软件工程中软件结构的研究 13

2.2.1　面向过程软件结构度量研究 14

2.2.2　面向对象软件结构度量研究 14

2.3　基于软件网络的软件结构研究 15

2.3.1　软件网络拓扑结构分析 16

2.3.2　软件网络的演化建模 18

2.3.3　基于软件网络的软件复杂性度量 19

2.4　本章小结 20

第三章　软件多粒度演化分析 **22**

3.1　引言 22

3.2　相关研究工作 24

3.3 SMGEA-CNT 方法 25
3.3.1 面向对象软件 26
3.3.2 数据收集 27
3.3.3 软件网络 27
3.4 实验设计 28
3.4.1 系统简介 28
3.4.2 实验分析 29
3.5 本章小结 39

第四章 软件稳定性分析及结构优选 **41**
4.1 引言 41
4.2 相关研究工作 42
4.3 SoS-SN 方法 44
4.3.1 数据收集 47
4.3.2 软件稳定性分析 47
4.4 *SoS* 的理论评价 52
4.5 *SoS* 的实验评价——结构优选 55
4.5.1 数据源 55
4.5.2 实验结果 55
4.6 本章小结 58

第五章 面向对象软件类重构 **59**
5.1 引言 59
5.2 相关研究工作 60
5.2.1 软件重构 60
5.2.2 社区发现算法 61
5.3 CR-CDSN 方法 62
5.3.1 重构的类型 62
5.3.2 数据收集 63
5.3.3 数据预处理 64
5.3.4 软件网络 65
5.3.5 模块度评价指标 66

5.3.6 有导向的社区发现算法 68
5.3.7 节点走查 68
5.4 实验设计 70
5.4.1 系统简介 70
5.4.2 实验及结果 71
5.4.3 实验分析 73
5.5 本章小结 75

第六章 面向对象软件回归测试用例排序 **76**
6.1 引言 76
6.2 相关研究工作 77
6.3 TCP-WCDN 方法 78
6.3.1 数据收集 79
6.3.2 软件网络 79
6.3.3 类测试关注度计算 83
6.3.4 测试用例优先级计算 86
6.4 实验设计 87
6.4.1 系统简介 87
6.4.2 实验结果与分析 88
6.5 本章小节 91

第七章 总结与展望 **94**
7.1 论文总结 94
7.2 存在的问题和进一步工作 97

参考文献 **99**

攻博期间发表的文章、参与的研究项目及获得的奖励 **114**

致谢 **116**

插图目录

1.1 Windows 源代码行增长情况 2
1.2 以“复杂网络”为关键词的 SCI（左）/ EI（右）论文发表情况统计 . . . 5
1.3 研究内容的体系 9

2.1 度量的实体及属性 12
2.2 软件复杂性分类 13
2.3 软件范型转变 14
2.4 传统结构度量关注点剖析图 16
2.5 JDK 1.2 软件网络（左）和累积度分布（右） 17
2.6 研究现状小节 21

3.1 传统软件项目中的两个阶段 22
3.2 SMGEA-CNT 方法框架 26
3.3 Java 源代码片段及其对应的软件网络 28
3.4 节点（左）/边（右）的演化 30
3.5 双对数轴（log-log plot）中 $< k > vs.$ 节点数 31
3.6 (a)-(c)：Azureus 4.5.0.2 版本的 $P_{cum}(k)$；(d)：度分布指数 α' 的演化 . . 33
3.7 Azureus 不同粒度软件网络 L 和 C 的演化信息 35
3.8 Azureus 不同粒度软件网络 L 和 C 的演化信息（黑色）及同等规模（相同节点数和边数）随机网络 L 和 C 的演化信息（红色） 35
3.9 Azureus 类粒度软件网络模块度的演化 37
3.10 Azureus 不同粒度软件网络同配系数的演化 38
3.11 Azureus 的 4.0.5.2 版本不同粒度的富人俱乐部连通性 40

4.1 一个简单的例子 45
4.2 一个简单的例子 45
4.3 SoS-SN 方法框架 46
4.4 代码片段及其相应的 WFDN 48
4.5 SoS vs. t 的曲线。其中：左图对应 $cr = 1/575$，$maxT = 50{,}000$ 的设置；右图对应 $cr = 6/575$，$maxT = 50{,}000$ 的设置 53

4.6 使用 Adapter 设计模式前（左）/后（右） Java 程序的 WFDN 56

5.1 CR-CDSN 方法框架 62
5.2 代码片段及其相应的 AMN 65
5.3 代码片段及其相应的 MMN 66
5.4 方法移动示意图 67
5.5 节点走查示例 70
5.6 JHotDraw 5.1 版本的 AMN 71
5.7 AMN 中包含重构点的小连通图（重构点为绿色节点） 72
5.8 JHotDraw 5.1的 MMN（每一种颜色代表一个连通图） 73
5.9 MMN 中的重构点（边上的文字是相应节点对应的方法名） 74
5.10 模块度 Q 随时间的变化曲线 74

6.1 植入错误的代码片段（红色部分是植入的错误） 78
6.2 TCP-WCDN 的框架图 79
6.3 代码片段及其对应的 WFDN 80
6.4 代码片段及其对应的 WFDN 81
6.5 WCDN 例子 85
6.6 JTopas-0.5.1 的 WFDN（上）和 WCDN（下） 90

表格目录

3.1 Azureus 的统计数据 29

4.1 变更种类 49
4.2 JUnit 3.4 统计数据 52
4.3 八组 Java 程序的统计数据 56
4.4 八组 Java 程序的 SoS （$maxT = 50,000$, $cr \times |N| = 1$, $mp = 0.2\alpha$ ($\alpha = 1, 2, 3, 4, 5$)） 57

5.1 JHotDraw 5.1 统计数据 70
5.2 AMN 的一些统计数据 71
5.3 AMN 中的重构点 72
5.4 MMN 中的重构点 73

6.1 实验 OO 软件的统计信息 89
6.2 测试用例优先级排序方法 92
6.3 各种排序技术 $APFD_c$ 值表 92

第一章　绪论

“规模改变一切[1]。”

—— *L. Northrop et al.*

“系统的结构会影响其功能、性能和可靠性等其它系统指标[2]。”

—— 钱学森

“没有度量就没有控制[3]。”

—— *T. Demarco*

1.1　研究背景

1.1.1　软件复杂性问题

自 1964 年第一台计算机 ENIAC 诞生之后，便有了程序的概念，可以认为它是软件的前身。在随后的几十年中，计算机软件获得了快速的发展，并经历了程序设计、程序系统和软件工程三大发展阶段[4]。如今，计算机软件已经渗透到了各个领域，成为我们日常生活不可分割的一个重要部分。应用领域深度和广度上的不断拓展以及人们对软件功能和质量需求的不断提高，使得软件规模激增，复杂性急剧增长。从最初的几十行代码，发展到现在的几十万、几百万行代码。以常见的 Windows 操作系统为例（如图 1.1 所示）[5]：Window 3.1 的源代码行数约为 300 万行，而到了 Windows Vista 源代码行数超过了 5,000 万行，代码行随时间呈“超线性”增长。加之市场竞争日趋激烈、软件需求不断变化、软件开发人员高流动性等因素，使得软件开发成本居高不下、开发进度难以控制、修改和维护困难，软件质量无法得到保障。

在过去的 40 多年里，软件工程领域获得了蓬勃发展，各种新技术、新标准层出不穷，但是“软件危机”的问题依然没有得到有效的解决。著名计算机科学奖、图灵奖获得者 F. P. Brooks 早在 1986 年其著名的《没有银弹：软件工程中根本和次要问题》中指出：“不存在任何一种技术或管理上的新成果，能够单独许诺在 10 年内使软件生产率、简洁性或可靠性获得大幅提高”[6]。如今 20 多年过去了，那颗传说中的“银弹”也一直没有出现。根据 2009 年美国 Standish Group[1] 对当年软件项目的调查报告显示[7]：“只

[1]Standish Group 是美国专门从事跟踪 IT 项目成功或失败的权威机构。自 1994 年以来，这个机构每两年发布一次对全球数千个 IT 项目的调查报告 CHAOS Report。

	Year(X)	SystemName(Y)
1	1990	Win 3.1
2	1995	Win NT
3	1997	Win 95
4	1998	Win NT 4.0
5	1999	Win 98
6	2000	Win NT 5.0
7	2001	Win 2k
8	2002	Win XP
9	2003	Win Vista

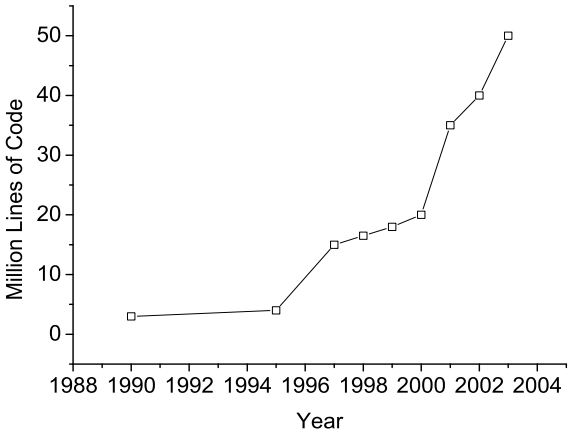


图 1.1　Windows 源代码行增长情况

有 32% 的软件项目是真正成功的，其余的项目全都是失败的或是有问题的，24% 的项目是完全失败的”。尽管软件项目成功率较 1994 年的 16.2% 已有所提高，但是总的来说还是很低的。关于此问题的原因，大家说法不一，但是越来越多软件工程领域的学者认为，造成这种窘境的根源在于软件固有的复杂性，软件复杂性使得软件的质量难以进行有效的量化、软件设计和开发难以进行有效的控制和管理[6, 8]。

尽管软件复杂性问题由来已久，但是对于什么是软件复杂性，至今还没有公认的定义。通常认为，软件复杂性是分析、设计、测试和修改软件的困难程度，并且这种复杂性日益增长[9, 10]。软件的复杂性一般有三种常见形式：计算复杂性（computational complexity）、心理复杂性（psychological complexity）和表征复杂性（representatiional complexity）[11]。其中：计算复杂性指解决一个问题所需要耗费的计算资源的数量；心理复杂性指人们认识软件和对软件进行操作的复杂程度；表征复杂性指个体对任务所含关系的表征能力，也可以看成软件的信息量。造成软件复杂性问题的原因不胜枚举，但是一般认为主要有以下几个方面：（1）软件环境的复杂性；（2）软件需求的复杂性；（3）软件模型的复杂性；（4）软件过程的复杂性；（5）软件项目管理的复杂性；（6）软件体系结构的复杂性；（7）软件测试复杂性；（8）非形式化方法的复杂性。在所有这些因素的综合作用下，软件复杂性已成为软件的基本属性。软件复杂性增加了软件开发和维护的难度，降低了开发的效率，并极大地影响了软件的可靠性。研究表明，软件复杂性是软件缺陷产生的重要根源，当复杂性超过一定界限时，软件缺陷数会急剧上升，软件可靠性急剧下降[12, 13]。因此，如何认识、度量、管理、控制乃至降低软件复杂性，是软件工程学亟待解决的一个重要问题[8, 14]。

1.1.2 软件度量的重要性

度量在软件工程中具有重要的地位。T. DeMarco 指出:“没有度量就没有控制”[3]。N. E. Fenton 等指出:“没有度量就没有预测和控制”[15]。管理学泰斗 H. J. Harrington 指出:“不能量化就不能理解，不能理解就不能控制，不能控制就不能改进”。C. Bill 指出:“不能度量，就无法理解；不能理解，就无法管理”[16]。为了管理和控制软件复杂性，并对软件质量进行科学评价，软件度量（学）（software measurment 或 software metrics[2]）应运而生，其实质是依据一些定义明确的规则，将数字或符号赋予实体（系统、构件、过程等）的特定属性，将属性进行量化，以帮助我们理解软件的状态（开发、维护等的状态），控制软件开发过程，改进软件过程和提高软件质量。软件度量的概念最早由 R. J. Rubey 和 R. D. Hartwick 在 1968 年提出[17]，经过 50 多年的发展，目前已经成为软件工程一个重要的研究方向。

复杂性对软件的稳定性（stability）、可维护性（maintainability）、可靠性（reliability）、可理解性（understandability）、可修改性（modifiability）和可测试性（testability）等外部质量都有重要影响。在过去的 50 年中，研究人员对软件复杂性展开了深入的研究，提出了各种软件度量指标对软件开发的各个阶段进行评估和控制。例如针对需求分析阶段的复杂性度量，A. J. Albrecht 和 J. E. Gaffney 提出功能点分析方法来度量系统的功能[18, 19]，M. Arnold 和 P. Pedross 提出了软件规模（size）和生产率（productivity rate）的度量方法[20]；针对设计阶段的复杂性度量，L. Briand、S. Morasca 和 V. R. Basili 提出设计内聚性和耦合性的度量方法[21]，J. Babsiya 和 C. G. Davis 提出一个评价设计质量的层次模型[22]；针对实现阶段的复杂性度量，R. Chidamber 和 F. Kemerer 提出了 CK 度量组[23]，M. Lorenz 和 J. Kidd 提出了 Lorenz 度量[24]，F. Brito 和 E. Abreu 提出了 MOOD 度量集[25]。据统计，现在已有各种度量 1,000 多个，发表的关于软件度量的文章 5,000 余篇，这些工作在一定程度上丰富了软件度量学的内容，加深了我们对软件的认识。但是正如 N. E. Fenton 指出的那样，这些度量在软件质量预测方面依然存在很大的缺陷，究其根源在于对软件复杂性的认识还不够全面[26, 27]。软件的复杂性是无法用一个单一的维度来描述的[28]，这犹如无法用长方体的长来表述其体积一样。因此，要对复杂性有一个全面的认识，我们必须从各种维度综合考虑。

[2]国内也有学者将 software measurement 和 software metrics 进行区分，即：前者译为软件度量，后者译为软件量度。在本文中我们不加区分，都表示软件度量。其实目前学术界还没有明确这两个术语的区别。

1.1.3 结构度量的重要性

软件结构被认为是影响软件复杂性的重要因素之一[29]，它是由软件元素（数据对象、方法、模块、类、构件、子系统等）在软件内的组织而形成的，主要包括组成软件的元素和元素之间的关系两个部分[30, 31]，展现出一种拓扑结构。软件结构的形成是软件设计、实现和维护阶段各种决策综合作用的结果。随着软件复杂性的激增，软件结构的重要性突显。系统科学的观点也认为，系统的结构会影响其功能、性能和可靠性等指标[2]。软件代表了一类人工的复杂系统，其结构也是决定其质量的重要因素之一[32, 33, 34]。软件工程学的奠基人 E. W. Dijkstra 曾指出："软件结构组织对设计的逻辑验证和实现的正确性检验都具有重要的作用[35]，我们在实现软件功能的同时也应该注重软件结构的理解"。D. P. Darcy 等人指出："软件的结构会影响软件维护，软件结构越复杂，人们理解软件就越困难，所需的费用也越高"[31]。R. Subramanyan、T. Gyimóthy 和周毓明等的研究显示，软件结构的复杂性与软件错误的分布、错误的数量都具有很大的影响[13, 36, 37, 38]。因此，要对软件系统的复杂性进行深入、细致的研究，就必须对软件的结构信息进行合理的描述和有效的量化。

结构信息的量化一直是软件工程中一个比较困难的问题。在 ACM 杂志 50 周年专刊上，F. P. Brooks Jr 将结构信息的量化（quantification of structural information）、软件估算（software estimation）和计算机系统用户界面设计（user interface design for computer systems）列为半个世纪以来计算机科学的三大挑战[39]。由于缺少相应的方法和理论，研究人员很少从整体的角度来审视软件的结构及其演化，导致对软件的本质缺乏清晰的认识[14, 39]。卡内基 · 梅隆大学（CMU，Carnegie Mellon University）软件工程研究所（SEI，Software Engineering Institute）在其 2006 年 7 月题为《Ultra-Large-Scale System: The Software Challenge of the Future》的报告中指出："未来的软件系统其代码行将以亿计，形成所谓的"超大规模系统"（Ultra-Large-Scale System），超过了目前开发实践的极限。因此，我们需要用新的观点、思想和方法来应对将会碰到的问题。在很多情况下，这种新的观点、思想和方法可以通过传统软件工程和其它学科的交叉得到"[1]。

1.1.4 复杂网络的研究

复杂系统和复杂性科学近年来成为人们关注的热点之一，被美国圣菲研究所（Santa Fe Institute）创始人 George Cowan 列为"二十一世纪的科学"，它的基本观点是结构决定功能，强调用整体的观点研究系统[40]。伴随着计算机技术的发展及其在复杂系统研

究中的应用，人们可以采集、存储和分析空前规模的数据，以复杂网络为代表的复杂系统研究兴起（如图 1.2[3]），它们将自然和技术中的许多系统用网络模型来研究，即：用节点代表系统中的元素，边代表元素间的相互作用。

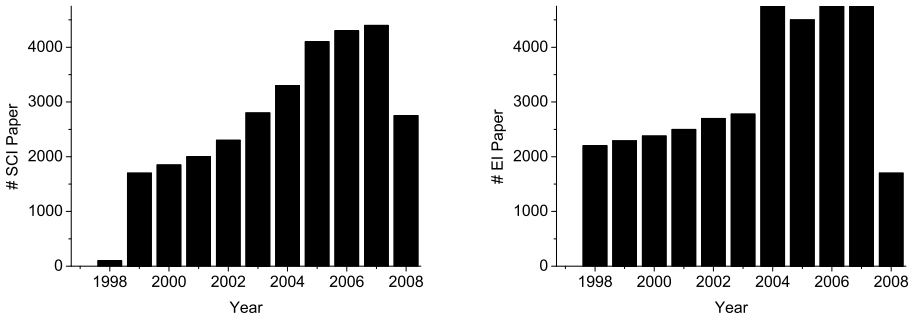


图 1.2　以“复杂网络”为关键词的 SCI（左）/ EI（右）论文发表情况统计

一般认为，推动这股研究热潮的是两篇著名的论文。1998 年美国康奈尔（Cornell）大学理论和应用力学博士生 D. J. Watts 及其导师 S. H. Strogatz 在《Nature》上发表题为《Collective dynamics of ‘small-world’ networks》[42]的论文，揭示现实世界中许多网络（电网、演员合作网络等）具有“小世界”（small world）特征，即网络具有较小的平均路径长度和较大的聚集系数。同时，他们提出小世界网络模型来解释这些特征的产生机理。一年以后，美国圣母（Notre Dame）大学物理系的 A. L. Barabási 教授和他的博士生 R. Albert 在《Science》上发表题为《Emergence of scaling in random networks》[43]的论文，揭示现实中的许多网络具有“无标度”（scale free）特征，即节点度服从幂率分布（power-law distribution）。同时，他们认为“增长”（growth）和“择优连接”（preferential attachment）是形成无标度网络的要点，并提出 BA 无标度网络模型来解释这些特征的形成原因。由于“小世界”、“无标度”等性质既不同于规则网络，也不同于随机网络，正如大多数物理学家认为的“复杂位于规则与随机之间”一样，所以大家把这些网络称为“复杂网络”[44]。

D. J. Watts、S. H. Strogatz、A. L. Barabási 和 R. Albert 的杰出工作使复杂网络研究进入了一个新时代，即“网络的新科学”[45, 46, 47]，受到了各个领域的专家的广泛关注和重视，有关复杂网络的论文和专著不断涌现。从 Internet[48, 49] 到 WWW[43, 50]、从大型电力网络[42]到全球交通网络[51, 52]、从生物体中的大脑[53]到各种新陈代谢网络[54]、从科研合作网络[55]到各种经济[56]等，甚至于语言[57]、数字[58]、乐谱[59]、地震[60]等许多在常人眼里并非网络的东西也可以从复杂网络的角度去研究。人们发现这些网络虽然来自不同的领域，代表不同的系统，但是都具有相似的“小世界”和“无标度”等特征，并

[3]该图摘自 [41]

根据揭示的拓扑结构统计特征，人们构建了很多网络演化模型，用于解释产生这些特征的内在机理。目前，复杂网络的研究内容可以归纳为[61]：(1) 分析：通过实证揭示网络结构的模式和统计性质，以及度量这些性质的合适方法；(2) 建模：建立准确的网络模型以帮助人们理解网络结构性质及其产生机理，并为研究网络上的行为奠定基础；(3) 预测与控制：在已知网络结构性质和产生机理的基础上，预测网络行为，并提出改善网络性能和设计新的网络的有效方法。可见，复杂网络已经成为研究各类开放复杂系统的（整体）拓扑结构及其动力学性质的有力工具。

1.1.5 软件网络的研究

软件系统从不同的粒度看是由数据对象、方法、模块、类、构件、子系统等构成的，并由这些元素的交互完成预期的功能，它代表了一类人工的复杂系统，当然也可以从复杂网络的角度进行研究[62]。近年来，统计物理和复杂系统科学领域的研究人员尝试将软件结构用网络（软件网络）的形式来描述，即将软件元素（数据对象、方法、模块、类、构件、子系统等）视为节点，元素之间的关系作为连接各节点的无向(有向)边。2002 年，S. Valverde 等首先研究了软件网络[63]。他们通过逆向工程方法从程序源代码得到系统的类图，并以系统类图为研究对象，采用无向图来抽象软件系统，即网络中的节点表示类，边代表类之间的继承和关联等交互关系。在多种不同类型的软件系统中，他们发现其软件网络都存在“小世界”和“无标度”等复杂网络特征，实验结果与人们原有的设想（节点度分布可能符合正态分布或泊松分布）截然不同。

软件网络可以描述软件系统的骨架，从较高层次描述软件的结构，强调软件系统的拓扑结构特征。2004 年以后，这些发现逐渐受到了国际软件工程领域研究人员的关注，并尝试用软件工程的相关原理对这种现象进行解释。相关论文见于一些较高级别的学术刊物上，如《IEEE Transaction on Software Engineering》、《Communications of the ACM》、《ACM SIGPLAN Notices》、《Physical Review E》以及 Elsevier 的《Information Science》等[62, 64, 65, 66, 67]。用复杂网络的观点来审视、研究软件系统的软件网络观，得到了越来越多软件工程领域研究者的认同和重视。通过分析 8 年来软件网络的相关研究可以发现，软件网络的研究内容主要集中于以下 3 个方面[14]：(1) 分析：发现和验证软件系统的结构特性（“小世界”、“无标度”等），探索存在于各种形式网络中的共同结构特征；(2) 建模：构建软件演化模型，探索软件生长和演化规律，分析软件系统具有“小世界”、“无标度”等结构特征的原因；(3) 度量：借鉴复杂网络理论对软件系统的结构复杂性进行度量，从结构角度评价软件质量。软件网络为我们研究软件系统的(整体)结

构及其动力学行为提供了新的视角。

1.2 问题的提出及意义

软件系统的结构对其功能、性能和可靠性等系统指标都有一定影响。要对软件系统的复杂性进行深入、细致的研究，就必须对软件的结构信息进行合理的描述和有效的量化。结构信息的分析和度量有助于对软件开发项目进行控制和管理，帮助我们把握其现状，预测其趋势，并为下一个项目提供依据。然而，由于缺少相应的方法，传统的软件结构度量研究很少从整体的角度来审视软件结构以及其演化规律，导致对软件的本质缺乏清晰的认识。复杂网络理论的出现、发展及其在软件结构度量研究中的应用，为系统的分析软件结构提供了坚实的理论基础和强大的分析手段，可以帮助我们从整体上把握系统结构及其动力学行为。

纵观软件网络 8 年多的发展，复杂网络与软件工程学相结合的研究已经取得了一些可喜的成果，但是软件结构的研究仍存在下面的一些问题：

（1）缺乏从不同粒度和演化的角度来分析和理解软件的组织结构、结构的演变和由此产生的行为特征

目前软件网络的研究还处于起步阶段，大部分研究都关注软件系统共同的结构特征及其形成规律。其思路都类似，即先通过逆向工程的方法，得到面向对象（过程）软件系统代码在某一个粒度（如面向对象中的方法、类或包，面向过程中的函数）的网络模型，再运用复杂网络基本理论发现和验证其是否具有“小世界”、“无标度”等复杂网络特征。但是它们忽略了软件的空间和时间特征。在时间上，软件是有一个生命周期的。一个成功的软件必定会经历一个自诞生（第一个版本），发展（经历多个版本），到最后淘汰的过程。在空间上，软件系统的代码从不同的粒度看是一个分层的，从微观到宏观、细粒度到粗粒度的过程。在不同粒度，软件元素包括数据对象、方法、模块、类、构件、子系统等。此外，现有的工作集中于研究软件的“小世界”和“无标度”的共性，而忽略了其它对软件性能更有意义的特征的研究，如：模块性、同配性等。因此，我们必须从多粒度和演化的角度来分析软件的结构及其演化，以及由此产生的一些行为特征，以便更加全面的地反映系统的整体性质。但是，目前这方面的研究工作比较匮乏。

（2）缺乏软件整体结构特征与软件质量（稳定性、可靠性、可维护性等）之间关系的认识

软件系统由于其自身的特点，难以像电视机、汽车等物理产品一样可以通过直接测量来评价其质量，而只能通过软件度量来间接评价。尽管人们普遍认识到软件结构对软

件质量（稳定性、可靠性、可维护性等）具有重要的影响，但是对于两者之间的具体关系仍不是太清楚。人们不清楚什么样的软件结构可以使软件稳定性和可靠性高，并且易于维护。面对具有相同功能的软件系统，人们也很难进行结构优选。因此，我们有必要探索软件结构与软件质量的具体联系，从而为重用软件开发成功经验等提供依据。但是，目前这方面的研究工作也比较匮乏。

（3）缺乏软件结构度量如何指导工程实践（软件重构、软件测试等）的研究

复杂网络和软件工程学的交叉研究不到十年，研究人员主要来自统计物理和复杂系统科学领域，他们侧重于分析和发现各类软件网络共有的结构特征及其形成规律，大部分工作仍停留于现象发现和解释阶段，离工程实践相距甚远。然而，软件开发实践者更希望得到的是在面对大规模软件设计、开发和维护问题时，能够简化其工作，指导他们开发出高质量软件的方法和工具。因此，软件网络的研究也应该朝这个目的努力。但是，目前这方面的研究工作仍很匮乏。

1.3 主要工作内容及研究体系

软件网络为分析复杂软件系统的结构及其动力学行为提供了有效的手段和工具。针对上述几个问题，在复杂网络理论和软件工程学的基础上，以软件静态结构[4]为研究对象，本文将从以下几个方面展开研究（研究内容的体系如图 1.3 所示）：

（1）软件多粒度演化分析

以多版本面向对象软件系统为研究对象，用逆向工程的方法，提取各个版本系统代码在不同粒度（方法、类和包）的网络模型。引入复杂网络基本理论，定量的分析软件网络的结构特征及其动态演化规律，分析这些特征和规律的形成机理及其对于软件设计、开发和维护等实际工作的现实意义。

（2）软件稳定性分析及结构优选

构建面向对象软件系统在特征（方法和属性）粒度的加权网络模型，并以此为基础研究软件结构和软件外部质量——稳定性之间的关系。具体包括：研究相应软件网络的构建方法；分析软件变更的不同种类，研究变更在软件网络中的传播行为，构建指标评价节点变更对系统的影响；在前两个工作的基础上研究软件稳定性度量指标，并从理论和实证两个方面验证度量指标的可靠性和有效性，并指出其在实际（软件结构优选）中

[4]这里之所以用“静态结构”一词，是为了与软件运行时表现出来的“动态结构”相区别。目前已有少量工作关注软件动态度量，即对软件运行时的动态调用关系进行分析。最近，J. K. Chhabra 和 V. Gupta 综述了该方面的研究工作[68]。

的应用价值。

（3）面向对象软件类重构

构建面向对象软件系统在特征粒度的网络模型，借鉴复杂网络社区发现相关研究成果，充分考虑设计模式对软件结构的影响，研究软件结构优化技术，为软件结构理解和优化提供支撑。具体包括：研究相应网络模型的构建方法；研究软件系统已有设计模式的识别方法；研究新的社区发现算法以适应软件重构工作的需要。

（4）面向对象软件回归测试用例排序

构建面向对象软件系统在类粒度的加权网络模型，从软件复杂性和结构出发研究软件回归测试用例排序技术。具体包括：研究相应加权软件网络的构建方法（权值的设置方法）；从类本身及类间交互复杂性出发，研究类错误倾向性、类测试重要性的度量方法；研究测试用例优先级度量方法。

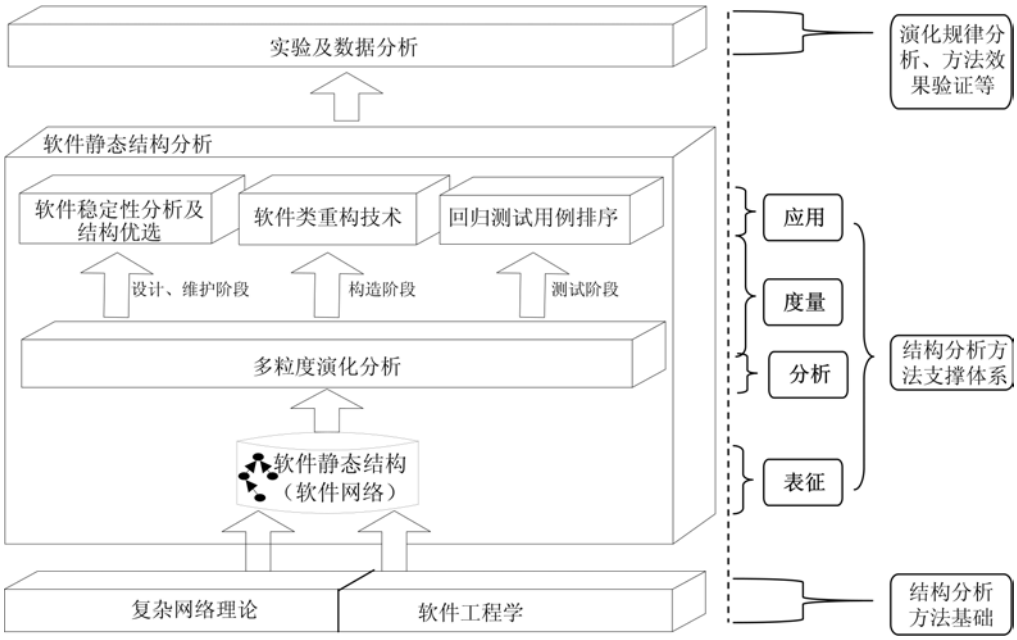


图 1.3　研究内容的体系

这四个内容围绕软件静态结构分析，从表征、分析、度量和应用四个方面来对面向对象软件静态结构进行分析，彼此之间是紧密关联的。结构的软件网络表征是分析、度量和应用的基础，我们的研究工作都是基于软件的网络模型的。分析、度量和应用是结构分析的三个方面（如上文所述）。同时，我们认为软件结构的软件网络表示并不是唯一的，研究的问题不同，相应的软件网络也会存在差异。因此，在本文的研究中，我们不分章单独讨论软件的网络表示方法，而将其贯穿于各章中，针对一个具体的问题，提出构建软件网络的方法。分析是度量和应用基础，通过分析可以发现结构中存在的特征和潜在问题，从而为度量和应用研究提供内容。此外，度量是应用的前提，我们是在软

件结构度量的基础上来解决软件工程中存在的结构优选、重构和测试问题的。

同时，多粒度演化分析的结果对软件过程的设计、构造、测试和维护阶段的工作以及软件管理都有指导意义。基于稳定性分析的结构优选、软件类重构技术和回归测试用例排序方法分别针对维护阶段、编码阶段和测试阶段的问题提供了相应的解决方案。可以说，基于软件网络的软件结构分析方法对于解决软件工程中，特别是软件过程各阶段中的问题都具有重要的意义。

1.4 论文的组织结构

本文共分七个章节对所研究内容进行阐述：

第一章　绪论：主要介绍本文的研究背景、要解决的主要问题及主要研究内容；

第二章　相关研究：介绍与本文工作密切相关的一些研究工作，包括传统软件工程中软件结构的研究和基于软件网络的软件结构研究，从而为后续章节的深入讨论打好基础。

第三章　软件多粒度演化分析：从方法、类、包三个粒度构建软件的多粒度软件网络模型，引入复杂网络基本理论刻画软件结构特征，分析不同粒度软件结构在演化过程中的变化趋势和规律。

第四章　软件稳定性分析及结构优选：构建软件特征粒度的加权软件网络模型，分析变更在该网络中的传播行为，定义软件稳定性指标，并引入一种仿真算法用于计算该度量值。通过理论和实例分析说明该度量的有效性，并研究其在软件结构优选方面的作用。

第五章　面向对象软件类重构：构建面向对象软件系统在特征粒度的网络模型，借鉴复杂网络社区发现相关研究成果，提出一种有导向性的社区发现算法，在尊重设计模式的前提下优化软件的类结构。

第六章　面向对象软件回归测试用例排序：构建软件类粒度的加权网络模型，分析软件复杂性与软件缺陷产生之间的相互关系，定义节点测试重要性指标及其计算方法，并用于软件测试用例排序。

第七章　总结与展望：归纳和总结本论文所做的研究工作，分析存在的问题，并提出今后的工作设想。

第二章　相关研究

“读史使人明智。”

—— 弗朗西斯·培根

“以人为鉴可以知得失，以史为鉴可以知兴替。”

—— 唐太宗李世民

2.1　引言

软件度量学的概念来源于 1968 年 R. J. Rubey 和 R. D. Hartwick 的一份工作《Quantitative measurement of program quality》[17]，它几乎与软件（程序）的概念同时产生。1976 年，B. W. Boehm 指出[69]：“对软件（程序）的属性不能仅有定性的研究，还必须有定量的研究”。1977 年，M. H. Halstead 指出[70]：“任何一门学科要成为科学，必须理论和实践相结合，而软件度量学正反映这种结合”。在这种形势下，软件度量方面的研究获得了稳步发展，目前已经成为软件工程一个重要的研究方向。

软件度量其实质是依据一些定义明确的规则，将数字或符号赋予实体（系统、构件、过程等）的特定属性，将属性进行量化，以帮助我们理解软件的状态（开发、维护等的状态），控制软件开发过程，改进软件过程和提高软件质量。IEEE 将软件度量（software quality metric）定义为[71]：度量是一个函数，它的输入是软件数据，它的输出是一个单一的数值，用于表示软件某一属性对软件质量的影响程度。

软件度量主要有三大应用，即：描述、评估和预测。软件度量可用于描述软件，获得对软件的认识；软件度量可以跟踪软件项目的进展，为作出正确的决策提供支持，它也可以对一个软件产品或软件过程作出评价；软件度量用于预测的时候，软件的属性常以数学模型的形式给出，并将其与软件其它属性关联。通过这种形式的度量，我们可以为特定软件项目做资源和时间的规划，还可以预测项目的规模、质量或其它属性。在以上三种情况中，软件度量都提供了量化支持，而不再只是凭借个人经验。管理者可以使用历史项目的数据来决定使用何种语言、工具或采用何种过程。

为了进行度量，我们必须识别度量的实体及其属性。R. R. Dumke 和 E. Foltin 将软件度量实体分为三类[72]：过程（process）度量、产品（product）度量和资源（resource）度量，这种观点得到了大多数软件度量研究者的认同。同时，针对一个具体的实体，其属性又分成了内部属性和外部属性。内部属性是指仅使用实体自身来直接度量的属性，

通常描述软件的复杂性，一般具有相对明确的定义，如软件规模、控制流、耦合等；外部属性指由实体及其环境（人等）共同才能度量的属性，不能进行直接度量，如可靠性、可维护性、可用性等。图 2.1[1] 所示的是软件开发各个阶段的软件实体及其属性。从图 2.1 我们可以发现，软件度量可以应用于软件开发各个阶段（需求分析、设计、编码、验证等）。一般软件度量使用的越早越有效，但是在早期，度量数据的获取比较困难，因此，目前很多的度量都是针对软件源代码的。

Entity	Internal Attributes	External Attributes
Product		
Requirements	Size, Reuse, Modularity, Redundancy, Functionality	Understandability, Stability
Specification	Size, Reuse, Modularity, Redundancy, Functionality	Understandability, Maintainability
Design	Size, Reuse, Modularity, Coupling, Cohesion	Comprehensibility, Maintainability, Quality
Code	Size, Reuse, Modularity, Coupling, Cohesion, Control Flow Complexity	Reliability, Usability, Reusability, Maintainability
Test set	Size, Coverage level	Quality
Process		
Requirements analysis	Time, Effort	Cost effectiveness
Specification	Time, Effort, Number of requirements changes	Cost effectiveness
Design	Time, Effort, Number of specification changes	Cost effectiveness
Coding	Time, Effort, Number of design changes	Cost effectiveness
Testing	Time, Effort, Number of code changes	Cost effectiveness
Resource		
Personnel	Age, Cost	Productivity, Experience
Team	Size, Communication Level, Structure	Productivity
Software	Size, Price	Usability, Reliability
Hardware	Price, Speed, Memory size	Usability, Reliability

图 2.1　度量的实体及属性

对于软件管理者来说，他们感兴趣的是软件的外部属性，如开发人员的效率是多少，软件的可靠性有多高等。但是，外部属性不能直接度量，一般只有通过内部属性间接度量。同时，对于外部属性的定义也比较困难，很少有公认的定义。从第 1 章，我们已经知道，软件复杂性是软件的基本属性，它对软件质量具有重要影响，已经成为影响软件质量的重要内部属性。可以说，复杂性包含了软件所有的方面。软件复杂性一般有三种常见形式（如图 2.2[2]）[11]：计算复杂性（computational complexity）、心理复杂性（psychological complexity）和表征复杂性（representatiional complexity）。计算复杂性指解决一个问题所需要耗费的计算资源的数量；心理复杂性指人们认识软件和对软件

[1]该图摘自 [73]

[2]该图修改自 [11]。本文将结构复杂性度量细分成元素级度量和系统级度量。

进行操作的复杂程度；表征复杂性指个体对任务所含关系的表征能力，也可以看成软件的信息量。其中，心理复杂性又包含问题复杂性（problem complexity）、程序员特征（programmer charactersitics）和结构复杂性（structural complexity）三个方面。本文主要围绕软件的静态结构展开，借鉴了复杂网络基本原理，从软件网络角度分析软件结构。为此，本章主要关注结构复杂性方面的研究，并从传统软件工程中软件结构的研究以及基于软件网络的软件结构研究两个方面进行介绍。

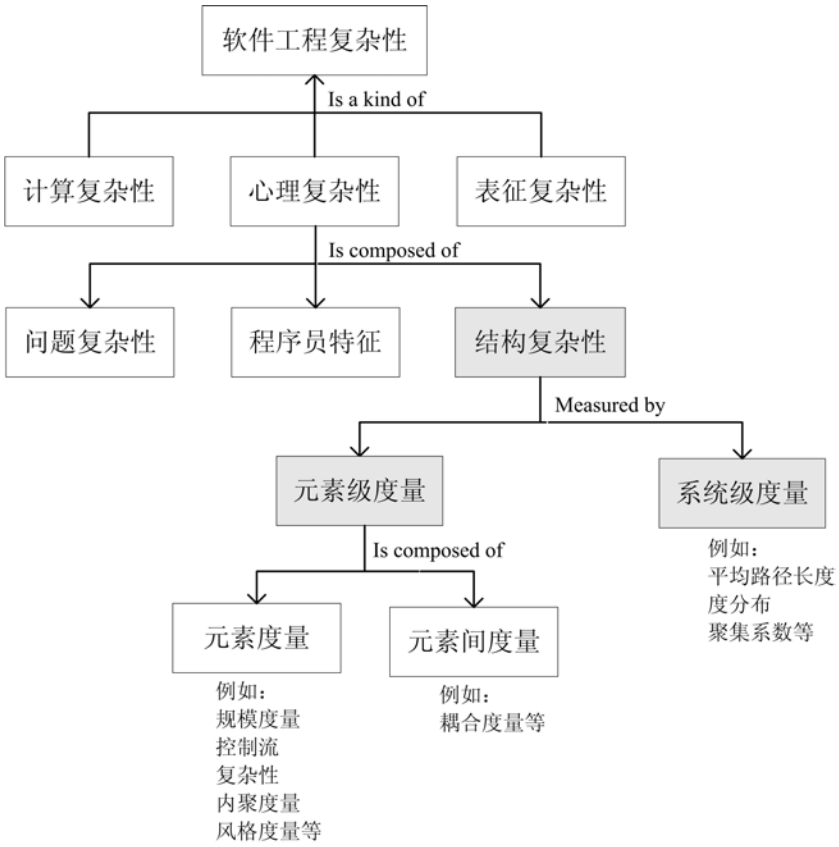


图 2.2　软件复杂性分类

2.2　传统软件工程中软件结构的研究

通常，软件开发范型大致经历了自由程序设计、面向过程程序设计、面向对象程序设计、面向构件程序设计和面向网络服务五个阶段（如图 2.3）。软件度量随着软件的产生而产生，软件范型改变了，软件度量也会作出相应转变。到目前为止，软件系统的结构度量研究主要经历了两个阶段：（1）面向过程软件结构度量研究；（2）面向对象软件结构度量研究。

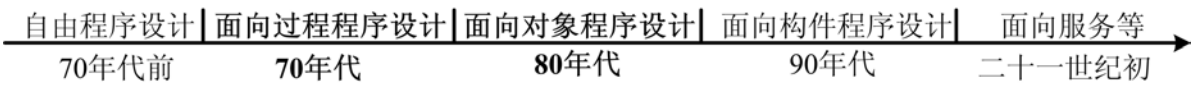


图 2.3　软件范型转变

2.2.1　面向过程软件结构度量研究

面向过程编程技术在上个世纪 70 年代盛行一时，成为主流的编程技术。那时的软件度量研究也主要是基于面向过程软件的。1974 年，R. W. Wolverton 提出用 LOC（Lines of Code）语句行度量程序员的代码率[74]。1976 年，T. J. McCabe 基于图论提出用流程图的独立路径条数，即 McCabe 圈复杂度（Cyclomatic Complexity），来度量程序的复杂性，并用 McCabe 圈复杂度作为测试方法论的基础[75]。1977 年，M. H. Halstead 通过计算程序中的运算符（operator）和操作数（operand）的数量来度量程序的复杂性，这就是著名的 Halstead 度量法[70]。1979 年，A. J. Albrecht 从需求说明书出发，提出了功能点分析方法[18]。1980 年，E. I. Oviedo 构建了一个模型将控制流和数据流合并处理，并通过计算控制和数据流复杂性来度量程序的复杂性[76]。1981 年，W. A. Harrison 和 K. I. Magel 提出了一种通过流程图分解来度量软件复杂性的方法[77]。1984 年，K. C. Tai 提出了一种基于数据流的程序复杂性度量[78]。其它用于度量面向过程软件结构复杂性的方法还包括：结（knots）度量[79]、扇入扇出（fan in-fan out）方法[80]、互连（interconnection）度量[81]等。在这些方法的基础上，又出现了很多其它新的度量方法。

2.2.2　面向对象软件结构度量研究

到了上个世纪 80、90 年代，面向对象技术获得了空前的发展，并逐渐取代面向过程程序设计成为软件开发的主流，运用面向对象技术设计和开发的软件系统越来越多。为了弥补面向过程开发方法的不足，面向对象技术引入了类、封装、继承、多态、信息隐藏、重用、消息传递等机制，以提高软件的可扩展性、可维护性等。面向对象技术强调的是对等实体间的交互而不是面向过程中的层次分解关系。因此，面向过程的软件度量方法不能很好的反映面向对象技术的这些特征，度量效果不好[82]。

为了应对这一问题，各种针对面向对象软件的度量不断出现。1989 年，Morris 和 L. Kenneth 研究了那时已有的软件度量方面的工作，筛选部分度量用于面向对象软件的度量[83]。1991 年，J. M. Bieman 提出了一个用于度量软件重用性的框架，并定义了一些评价软件重用性的度量，如公共/私有重用性（public/private reuse）等[84]。1992年，A. Lake 和 C. R. Cook 研究了 C++ 软件的度量问题[85]。1993 年，R. Sharble 和 S. Cohen

研究了面向对象软件设计的度量[86]。1993 年，M. Lorenz 提出了 11 个面向对象度量，如平均方法规模（Loc）、每个类中方法的平均数量、每个类中实例变量的平均数等，还针对性的给出了一些度量的校验规则[24]。1994 年，M. Lorenz 又对其进行了扩展，将度量指标分为方法规模、类规模等 7 大类。1994 年，R. Chidamber 和 F. Kemerer 阐述了面向对象软件度量学理论基础，提出了 6 个面向对象软件设计和复杂性度量，如加权方法数 WMC（Weighted Method per Class）、继承树深度 DIT（Depth of Inheritance Tree）、类间耦合度 CBO（CouplingBetween Objects）等一套面向对象的度量方法，这就是著名的 CK 度量组[23]。1995年，F. B. Abreu 针对面向对象软件的封装性、继承性、耦合性和多态性提出包含 6 个度量指标的 MOOD 度量集[87, 88]。

之后，有很多研究工作是对 CK 度量组和 MOOD 度量集进行评价和分析的，并在这两个度量方法的基础上提出了新的度量方法，如 W. Li 和 S. Henry 在 CK 度量组的基础上，提出了几个新的度量指标，并分析了其与软件维护之间的关系[89]；Y. Lee 等基于信息流（Information Flow）提出了用于度量面向对象程序的耦合性和内聚度的度量指标[90]等。当然还有很多面向对象软件结构度量方法，这里不再一一枚举，详见 [91]。

2.3 基于软件网络的软件结构研究

从面向结构的程序设计到面向对象程序设计，再到面向构件/服务的软件开发方式，软件的设计和实现的重点已从实现局部的编程难题，转向了如何将代码有效的进行组织。特别是对于大规模的软件系统而言，结构的组织就尤为重要。一个结构良好的软件往往易于维护。但是传统的结构度量方法（面向过程/对象度量方法）更多的考虑的是软件元素的属性度量，即一个类中有多少方法、一个方法有多少行代码等，尽管也有部分度量（CBO、 DIT 等）考虑了元素的交互，但是对于软件结构本身包含的复杂信息还没有充分地挖掘。此外，传统度量方法往往是在单一粒度下实施度量的，这对于一个由不同粒度、层次元素构成的复杂软件系统也是不足的。为了更加清楚的指出传统结构度量方法的不足，我们将以图 2.4 为例进行说明。其中：方法/属性粒度的虚线椭圆内的节点表示它们共属同一个类，类粒度虚线椭圆内的节点表示它们同属一个包。传统软件结构度量往往关注孤立的一个软件元素，如在度量包的时候，往往关注一个包内的元素（对应类粒度一个虚线椭圆内的部分）；度量类的时候，关注一个类内的元素（对应方法/属性粒度虚线椭圆内的节点）。但是忽略了元素间的交互，如包粒度各个包与其邻居，邻居的邻居，邻居的邻居的邻居·····的关系。CBO 等指标虽然关注了元素的交互，但是它只是度量与某个元素有直接关系的元素的个数，没有考虑通过关系传递产生

的间接交互关系，如图中由一个节点的可达集构成的子图的整体度量，甚至整个系统整体的度量指标。因此，我们有必要从整体和全局的角度，从不同粒度系统的挖掘软件结构中蕴含的信息。

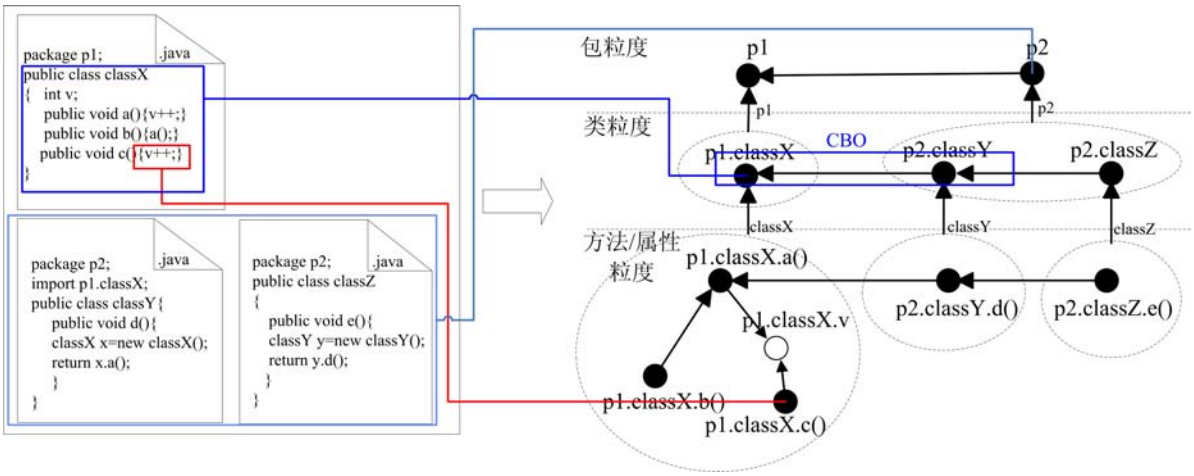


图 2.4　传统结构度量关注点剖析图

从第 1 章的描述我们已经知道，复杂网络为我们研究系统结构及其动力学行为提供了有力的工具。近年来，一些研究者将复杂网络相关理论引入软件工程中，构建软件网络模型，用于分析软件结构特征及其动力学行为。复杂网络理论强调从整体上认识、理解和控制系统，而非关注局部。研究系统在整体上表现的特征可以为我们理解软件系统提供新的维度。但是，复杂网络理论与软件工程学相结合的研究工作目前仍处于起步阶段。以软件系统静态结构为分析对象，研究人员主要从以下三个方面展开研究：(1) 软件网络拓扑结构分析（分析）：发现和验证软件系统的结构特征（“小世界”、“无标度”等），探索存在于各种形式网络中的共同结构特征；(2) 软件网络的演化建模（建模）；构建软件演化模型，探索软件生长和演化规律，分析软件系统具有“小世界”、“无标度”等结构特征的原因；(3) 基于软件网络的软件复杂性度量（度量）：借鉴复杂网络理论对软件系统的结构复杂性进行度量，从结构角度评价软件质量。下面我们将分节进行介绍。

2.3.1　软件网络拓扑结构分析

2002年，S. Valverde 等首先将复杂网络方法引入软件结构分析中[63]。他们将系统的类图用无向网络来抽象，即：网络中的节点表示系统中的类，节点间的边表示类间的关系，如继承（inheritance）、关联（association）等。他们用这种方法抽象 JDK 1.2 和

UbiSoft ProRally 2002，并用复杂网络中的相关方法分析这些软件网络（如图 2.5 [3]）的统计特性，他们发现，这两个软件的软件网络具有与其它复杂网络类似的“小世界”和“无标度”特性。同时，他们认为软件开发中的局部优化过程可能是是导致软件网络呈现这些特性的原因。

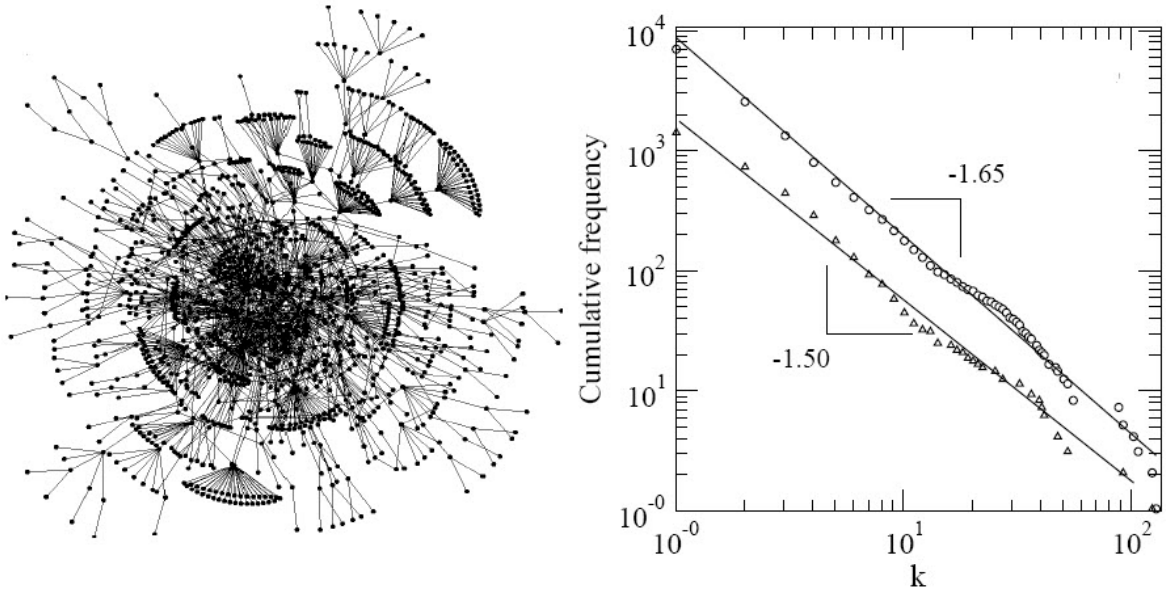


图 2.5　JDK 1.2 软件网络（左）和累积度分布（右）

在软件系统中，系统模块（类、方法等）之间的关系（协作、调用等）反映了系统的控制流。因此，软件网络中的边的方向是有意义[62]。2003 年，一些研究者使用有向网络研究软件结构。C. R. Myers 使用有向网络表示软件系统的结构，分析了 3 个面向对象软件的类协作图（class collaboration graph）和 3 个面向过程软件的静态过程调用图（static procedure call graph），他们发现：（1）这些有向软件网络尽管来自不同系统，甚至是用不同类型语言（面向过程/对象）开发的系统，但是都具有“小世界”和“无标度”特性；（2）出入度分布的幂指数存在不平衡性，即出度分布的幂指数大于入度分布的幂指数；（3）出度与入度之间存在负相关性，他们认为这可能与软件重用有关，即简单的类（出度小的类）容易被重用，因此入度大。S. Valverde 和 R. Solé 使用有向网络描述软件系统的类图（class diagram）对系统的结构进行研究。他们发现，类粒度的软件网络除了具有“小世界”和“无标度”特性外，同时具有层次性（hierarchical）和模块性（modularity）[92]。

之后，人们开始从不同粒度（方法、类、包、文件等）对大量开源软件的结构进行研究。A. P. S. Moura 等将 C 和 C++ 语言开发的系统在文件级（头文件）抽象成网络，

[3]该图引自 [63]。

即头文件为节点，头文件的共现关系抽象成边。他们发现这样的网络也具有“小世界”和“无标度”特性，并指出：“软件随时间增长是软件呈现无标度特性的原因，软件性能的优化是软件具有“小世界”特性的原因”[93]。N. LaBelle 和 E. Wallingford 在包粒度将软件系统看成网络，对一些软件的依赖网络进行分析，发现这样形成的网络同样具有“小世界”和“无标度”特性[94]。S. Valverde 和 R. V. Solé 将类和类的方法都抽象为节点，方法访问（reference）类的关系抽象成类节点和方法节点间的边，他们发现这样构建的软件网络也同样具有“小世界”和“无标度”特性[95]。国内的研究人员在这方面也做了些实证研究，如李德毅、李兵、祁国宁、赵海等人对 Java 语言编写的若干软件系统以及 Linux 等进行分析，发现了其中的“小世界”和“无标度”特性[96, 97, 98, 99, 100]。

近来，S. Jenkins、M. Shi、L. Wang 等以复杂网络理论为工具，从软件版本演化的角度分析软件结构的演化规律[67, 101, 102, 103]。除发现“小世界”、“无标度”等特性外，他们还发现了软件网络节点间的距离随演化不断增长等新特性，为研究软件结构特征提供了新的视角。

2.3.2 软件网络的演化建模

C. R. Myers 认为软件网络节点的入度代表的是软件的重用率，节点入度越大重用率越高。通过这种重用形成中枢节点从而有效提高节点间的信息传递效率[62]。同时，他还指出：“软件系统的结构不但要实现复杂的功能，同时还要支持软件的演化，使软件能以一种较小的代价实现演化，而重构（refactoring）在这个过程中扮演着重要的角色，通过重构生成一些小规模、简洁的代码片段从而促进其重用”。由此，他提出基于重构（refactoring-based）的软件演化模型。仿真实验表明，由该模型生成的网络，许多网络特性（出/入度分布、出入度幂指数差异、度相关性、聚集系数等）都与真实系统的软件网络很相似。

S. Valverde 和 R. V. Solé 等认为软件系统表现出的复杂网络特性可能与设计时的相互矛盾的约束条件（以较短的平均最短路径产生较小的代价）有关，是在综合考虑各种约束条件后优化设计的结果[63, 104]。在此基础上，R. V. Solé 和 S. Valverde 等分析了 C++ 系统的软件网络，他们发现，网络规模与子图数量间存在因果关系，软件的生长与细胞网络很相似，具有复制（duplication）和分岔（divergence）现象。根据这个发现，他们提出了基于复制（duplication）和分岔（divergence）的软件演化模型[105]。

何克清等从设计模式（design patterns）的角度出发来研究系统的生长。他们把软件模式分成“冷冻部分”（frozen spots）和“热点部分”（hot spots），提出了基于模式

冷冻部分的演化模型[106]。其中:“冷冻部分”是模式中类角色间的协调依赖关系，不可变；“热点部分”是模式扩展和变更的规范。该模型生成网络的许多特性与先前的发现非常吻合。

李兵等将复杂网络和演化算法相结合，提出了一种软件网络的演化模型 CN-EM[107]。仿真数据（如平均最短路径、聚集系数等）显示该方法能够比较好地刻画实际软件系统复杂网络特性的涌现过程。

此外，李桓等针对有向和无向软件网络分别提出了相应的演化模型，用于模拟软件网络出现的无标度特性[108]。

2.3.3 基于软件网络的软件复杂性度量

R. Vasa 等根据软件网络中节点数和边数之间的关系来研究软件结构的变化，并预测软件的规模和构造该软件所需的代价[109]。

Y. Ma 等根据节点的度定义了结构熵（structural entropy），用于量化和分析软件系统结构的有序程度，同时研究了结构熵与软件鲁棒性和通信效率间的关系，为系统结构的优化提供了依据[110]。

A. Girolamo 等以介数等指标定义了一套面向对象软件的度量指标，从不同层次（类层、网络层和设计层）识别和检测软件结构的缺陷以及有问题的类，并评价设计的质量[111]。

2006 年，Y. Ma 等以软件的基本属性——内聚和耦合为核心，融合复杂网络基本参数、面向对象度量指标和代码级度量指标，从宏观（网络）、中观（社区）和微观（节点）视角分析复杂软件系统的结构，提出了一个层次型的度量体系[112]。

S. Jenkins 等在面向对象软件系统的类粒度，提出了一个新的度量 I_{cc} 用于度量软件的稳定性[67]。

R. Vasa 等提出若干度量指标用以度量开发中软件结构的稳定性变化。他们发现类的规模及其复杂性的分布随时间变化不大，具有较大入度的类易被修改[113]。

Melton 等研究了 81 个开源系统类之间的依赖关系，他们发现，声明了从其它类访问的非私有成员的类易形成依赖环，从而使得软件的复杂度增加、稳定性降低[114]。

Y. Ma 等使用网络 Motif 来研究软件结构的稳定性，研究软件微观结构稳定性与宏观结构稳定性之间的详细关系[115]。在分析了 6 个软件系统的子图后，他们发现，统计重要性较高的子图都拥有较稳定的结构，而较稳定的结构都没有环的出现。因此，他们建议程序员在开发程序时应尽量避免环的产生。

H. Zhang 等使用 k-核（k-core）研究面向对象软件系统类粒度的拓扑结构[100]。他们发现，软件网络的核数（coreness）存在相似性（similar coreness），即软件网络的核数（coreness）相对都比较小，且核数与软件网络的规模关系不大。同时，他们使用核数度量软件的层次性（hierarchies）。他们认为核数相似性可能是软件开发中维护代价和软件性能折中的结果。

2.3.3.1 研究现状小节

图 2.6[4] 前三个部分（分析、建模和度量）总结了近几年来软件网络相关的研究工作。可以看到，软件网络的研究尽管已经取得了一些的成果，但是目前仍处于起步阶段，目前的大部分工作仍停留在发现和解释的层面，距离工程实践还有一段距离（即第四部分的工作还很少）。纵观软件网络 8 年多的发展，我们发现软件网络的研究仍存在以下一些问题（详见第 1 章 1.2 小节）：（1）缺乏从不同粒度和演化的角度来分析和理解软件的组织结构、结构的演变和由此产生的行为特征；（2）缺乏软件整体结构特性与软件质量（稳定性、可靠性、可维护性等）之间关系的认识；（3）缺乏软件结构度量如何指导工程实践（软件重构、软件测试等）的研究。而本文的研究旨在从软件结构的网络表征、分析、度量和应用四个方面系统的分析软件静态结构，帮助开发人员理解软件结构演化的特征、评价软件结构稳定性、检测重构点，以及提高回归测试效率，具有鲜明的针对性。

2.4 本章小结

本章简单的介绍了一下与本文密切相关的几个领域的研究成果。首先，回顾了现有的软件结构度量研究，包括面向过程软件的结构度量研究和面向对象软件的结构度量研究。其次，分析了现有的软件网络的研究工作。上述 2 个方面的研究内容是本文工作的基础，对本文的研究工作具有借鉴意义。

总的来看，传统的软件工程研究尽管在一定程度上提供了软件结构的一些信息，但是它们更多的考虑的是软件元素的属性度量，对软件结构信息的挖掘不够全面。将复杂网络和软件工程相结合，用网络的观点从整体和全局的角度审视软件结构，有助于我们进一步理解软件的本质，弥补了传统软件工程结构度量的不足。同时，复杂网络理论具有较好的数学基础，能为软件结构的分析提供理论支撑。

[4]该图修改自 [14]。

研究主题	研究特点		代表作
分析	研究内容：	发现和验证软件结构具有的特性，如：“小世界”、“无标度”等	[63][64][68][93-101][147-149]
	研究人员：	主要来自统计物理学和复杂系统科学领域	
	研究方法：	图论、统计分析等	
建模	研究内容：	构建模型，解释系统具有某些结构特性（“小世界”、“无标度”）等的原因	[63][64][102-106]
	研究人员：	主要来自统计物理、复杂系统科学、软件工程领域	
	研究方法：	图论、统计物理、设计模式等	
度量	研究内容：	借鉴复杂网络理论，提出软件结构度量，评价软件复杂性，进而评价软件质量	[68][101][107-113]
	研究人员：	主要来自软件工程领域	
	研究方法：	复杂网络、逆向工程等	
应用	研究内容：	通过软件结构度量指导软件工程实践，如：软件测试、代码重构、剽窃代码检测、设计模式挖掘等	
	研究人员：		
	研究方法：	复杂网络、逆向工程、数据挖掘等	

图 2.6　研究现状小节

当前，软件网络的研究还处在起步阶段，现有的大部分工作仍停留在发现和解释的层面，距离工程实践（指导软件的设计、开发和维护，预测软件质量）还有一段距离。

第三章　软件多粒度演化分析

“人们从物理、生态、地理、水文、气象等许多不同研究角度发现，在不同尺度下，事物的结构和发展变化呈现出不同的规律[116]。”

—— *C. E. Woodcock et al.*

“获得软件系统高层的演化信息，有助于理解系统的演化过程及趋势、降低软件复杂性、提高软件质量，并指导软件工程实践[117]。”

—— *H. C. Gall et al.*

3.1　引言

“物竞天择，适者生存”，生物的进化是生物对环境不断适应的结果[118]。软件系统同生物一样，在其生命周期内，也必须不断的演化，否则就有可能提前被淘汰[119]。软件系统处在一个不断变化的环境中，这个环境具有技术性和社会性[120]：(1) 软件系统运行在一个技术的支撑环境之上；(2) 人们对系统的认识和评价是基于人们对系统直接或间接的使用体验。正是软件系统与环境的这种交互决定着它的成败。因此，当环境改变时（新的用户需求、新的应用环境、性能的提高、存在安全隐患、使用不便和错误的修正等），软件系统也必须做必要的变更以适应环境的这种变化，这一过程就是软件演化（software evolution）[1]。一般认为，软件演化在软件第一个版本发布之后便开始了[122]（如图 3.1[2]），它的费用占了软件开发总费用的 90% 以上[124]。

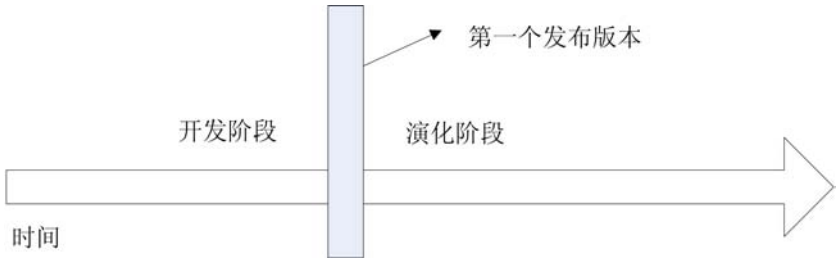


图 3.1　传统软件项目中的两个阶段

T. Mens 等在 IEEE Software 杂志 2010 年 25 卷第 4 期刊登了一期软件演化方面的专辑，收录了 K. Beck 和 B. Boehm 等著名软件工程领域科学家的文章。在这个专

[1]在谈及软件时，软件演化（software evolution）和软件维护（software maintenance）这两个概念常被看成同义词。但正如 D. L. Parnas 指出的[121]，维护（maintenance）一词常指在保持系统原有设计（design）的情况下对系统所采取的变更，而软件变更常会导致设计的变化。为了凸显软件随时间变更的动态特性，本章使用软件演化一词。

[2]此图引自文献 [123]

辑中，T. Mens 等指出[125]：“软件的演化同物理商品实体的演化不同，它主要指同一软件系统不同版本之间的差异”。软件演化反映的是软件实体或其组成部分属性的变化过程，即探讨软件系统是如何随时间演化的[126, 127]。软件演化分析有助于发现软件系统中存在的问题，预测系统的发展趋势，对于软件管理，以及软件过程中的设计、编码、测试和维护过程中的工作都有指导意义[128]。同时，软件演化分析对于构建比较合理的软件演化建模也具有重要的意义，因为建模的目的就是为了分析软件演化特性的形成机理。若我们不对软件系统采取任何的维护工作，软件演化常使得软件系统的复杂度不断增加，产品质量不断下降[126]。因此，为了应对软件复杂性的增长和质量的下降，我们必须获得软件的高层演化信息[117]。

软件结构对软件开发的成败和最终软件的质量（性能、可维护性、可靠性等）都有重要影响，近年来受到越来越多软件工程实践者和研究者的关注。软件的演化不可避免的使软件的设计（design）发生变化[129]，这种变化的直接反映是软件结构的变化，即：软件元素本身或软件元素之间交互的改变。因此，为了全面的认识软件的演化，我们必须对软件结构的演化进行深入、细致的研究，并从结构角度提供软件演化的高层信息。然而，由于缺少相应的方法（度量、支撑工具等），研究人员很少从整体的角度研究软件结构的演化，对软件演化在结构上表现出来的特性知之甚少[130]。

复杂网络作为网络科学的一个分支，已成为研究系统结构及其动力学行为的有力工具。在本章中，我们提出了面向对象软件演化分析的 SMGEA-CNT 方法（Object-oriented software multi-granularity evolution analysis using complex network theory），使用复杂网络理论来研究面向对象软件结构的演化。首先，我们提出了面向对象软件的多粒度软件网络模型，从方法、类和包三个粒度抽象面向对象软件系统的整体拓扑结构；然后，我们引入复杂网络研究中的常用度量（度分布、聚类系数、社团、同配性等）从多个粒度来量化软件结构特征；最后，通过跟踪这些度量在软件不同粒度以及不同版本之间的变化来分析软件结构的演化，研究软件系统在结构层面的演化规律及其对软件工程实践的指导意义。

本章后续内容的组织结构如下： 3.2 节介绍相关研究内容； 3.3 节介绍 SMGEA-CNT 方法，包括：数据的收集方法、多粒度软件网络模型等；3.4 节介绍针对开源 Java 软件的实验设计方法，包括：实验系统介绍和具体的实验分析方法；最后在 3.5 节对本章进行小结。

3.2 相关研究工作

早在上个世纪 70 年代，人们已认识到跟踪软件演化的重要性。M. Rochkind 提出了源代码控制系统（source code control system）来记录软件连续版本的源代码[131]。J. Hunt 和 D. Mcllroy 提出了基于文本的 Delta 算法识别系统哪些地方、在什么时候、做了什么变更[132]。这些早期的工作强调记录软件演化信息的重要性。

M. M. Lehman 是从分析和建模角度来进行软件演化研究的先驱，他和同事 L. Belady 等在过去的 40 多年中对 OS/360 操作系统进行了深入的研究，他们使用模块数来描述软件某一版本的规模，定义了一些度量来刻画软件连续版本的差异，最终揭示了软件演化的八条基本规律（持续变化、持续增长、复杂度不断增加、自律、组织稳定性守恒、熟悉度守恒、质量不断下降和反馈系统），被称为 Lehman 软件演化规律[119, 128, 133, 134]。M. M. Lehman 的学生 C. K. S. Chong Hok Yuen 之后系统的研究了 Lehman 软件演化规律，发现这八条规律并不是对所有的系统都成立的[135, 136, 137]。T. Tamai 和 Y. Torimitsu 使用问卷调查的形式对日本 5 年内商用软件的寿命（lifetime）进行研究，他们发现小规模软件的寿命会相对短些，管理类的软件（人事管理系统等）的寿命比一些商业辅助型软件（销售支持软件、制造辅助软件等）长些[138]。C. R. Cook 和 A. Roesch 跟踪了德国电话公司实时电话交换软件 18 个月的演化数据，它们发现了类似 M. M. Lehman 的结论，即：持续变化、熵不断增加和变化不一致[139]。C. F. Kemerer 和 S. Slaughter 提出了两种方法，即时间序列分析和序列分析，用于分析软件的演化，并对 23 个商用软件进行了分析[140]。A. Mockus 等研究了 Apache 服务器和 Mozilla 浏览器，他们发现 Apache 和 Mozilla 的演化主要是由一些核心开发者控制着，这些核心开发者决定着开发周期、软件结构等[141]。D. German 提出了一种基于软件轨迹（software trails）的软件演化分析方法，他利用来自邮件列表、网站、版本控制系统、文档、源代码等的信息分析软件的演化，这种方法可以提供软件项目演化的详细历史信息[142]。I. Antoniades 等提出了一个框架用于模拟开源软件演化，实验表明通过这个框架得到的软件演化的结果与真实软件演化的结果是大体一致[143]。A. Capiluppi 用一些基本的软件项目属性（规模、模块数、开发者数等）分析了 19 个开源项目的演化并推断其演化模型[144]。M. W. Godfrey 等使用代码统计的方法（code statistics）研究 Linux 内核连续 6 年的演化，发现 Linux 在系统级呈“超线性”增长模式，这与从商业系统中得到的结论相矛盾[145]。F. V. Rysselberghe 等将版本管理系统记录的变更信息进行简单、宏观的可视化，用于识别系统演化中的相关变更，如不稳定的部分，关联在一起的部分，以及设计和结构的演化[146]。

上述研究大多集中于软件版本数、系统开发/使用时间、系统规模、模块数等简单的统计，对软件演化的研究还不够深入。2007 年后，一些研究者尝试从软件系统整体结构角度研究软件的演化。S. Jenkins 等将软件系统在类粒度抽象成有向网络，并提出了一个新的度量 I_{cc} 来分析软件连续版本结构的稳定程度[67]。M. Shi 等将 JDK 的 5 个版本，在类粒度上抽象成网络，并应用复杂网络理论研究其结构特性，发现所有软件网络属于“小世界”（small world）和“无标度”网络（scale free），并且具有异配的层次结构（disassortative hierarchical structure）[101]。L. Wang 等应用复杂网络理论研究了 Linux 内核模块的演化，他们构建了 Linux 内核 233 个版本的调用图（call graph），并研究了这些调用图的诸多方面特征，他们发现这些调用图都为“无标度”和“小世界”类型网络[102]。H. Li 等将数个面向对象软件系统在类粒度抽象成软件网络模型，并应用复杂网络理论分析这些系统的演化，他们发现这些网络属于“无标度”网络，并且随着系统的演化，网络节点间距离不断增长[103]。

应用复杂网络理论研究软件演化的工作还处于开始阶段，存在以下问题：

（1）在单一粒度研究软件演化是不够全面的。软件系统具有时间和空间双重特性：在时间上，软件系统版本不断更新；在空间上，软件系统从不同的粒度可以看成是由数据对象、方法、模块、类、构件、子系统等软件元素及其交互构成的。因此，仅仅从单一粒度来研究软件演化的特点是不全面的，我们并不清楚从某一粒度得到的结论是否可以推广到其它的粒度，不同粒度软件演化存在什么区别和共同之处。因此，为了从结构角度全面、系统的认识软件演化，我们必须引入多粒度视角，从多个粒度综合研究软件演化的特性。

（2）分析的结构特征太少。现有工作大多分析了软件结构的“小世界”、“无标度”特性，对其它更有意义的结构参数，如模块度、同配系数、富人俱乐部等缺少分析，也并没有指出揭露的这些特征对软件工程实践的指导意义。

在本章中，我们提出了面向对象软件演化分析的 SMGEA-CNT 方法，该方法针对上述第一个问题，构建了软件的多粒度软件网络模型抽象软件不同粒度的结构。针对第二个问题，在验证不同粒度软件网络的“小世界”、“无标度”特征的基础上，对软件的模块度、同配系数、“富人俱乐部”等也进行了分析，并指出了这些工作对软件管理以及软件过程中的设计、编码、测试和维护过程的指导意义。

3.3 SMGEA-CNT 方法

图 3.2 是 SMGEA-CNT 方法的框架图，可以发现 SMGEA-CNT 方法主要由四个

部分构成。本节将详细介绍 ① ~ ③ 部分，第 ④ 部分将在 4.2 节中详细说明。

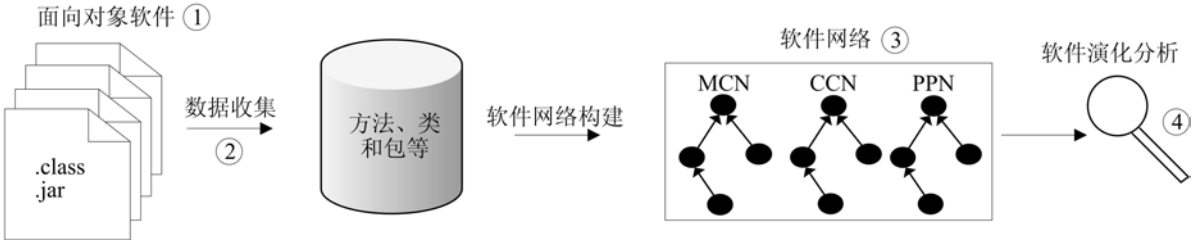


图 3.2　SMGEA-CNT 方法框架

3.3.1　面向对象软件

本章主要关注面向对象软件，并使用开源的用 Java[147] 语言编写的软件作为研究对象[3]。使用开源软件系统作为研究对象，主要考虑到开源的系统比较容易获取多个版本，可以访问其详细的变更日志（开发者日志、 CVS[148] 日志等），同时没有版权的问题，可以自由使用。使用 Java 语言编写的面向对象系统主要是基于以下几点考虑：

（1）面向对象技术自上个世纪 90 年代开始已经成为广泛使用的开发范型，网络上（如开源项目托管网站 Sourceforge[149]、 Freshmeat[150] 等）存储着大量开源的、用 Java 语言编写的软件系统，获取容易。

（2）这些软件都采用面向对象开发模式，具有相对清晰的内部结构，且易于编写工具提取软件元素(属性、方法、类、包等)以及它们之间的关联关系。

（3）选择 Java 语言编写的软件作为研究对象，主要因为目前我们的分析工具只能解析 Java 语言写的软件。

同时，从现有的文献来看，面向对象软件的软件网络主要从三个来源构建：

（1）从源代码构建：根据语法关键字的分析和过滤，得到软件网络模型[92]；

（2）从中间代码构建：通过解析中间代码，如 Java 语言的 .class 或 .jar 文件、GCC 抽象语法树文件等，从而构建软件网络模型[33, 34, 96]；

（3）从 UML 类图构建：通过逆向工程的方法得到系统的 UML 类图，然后通过对 UML 类图的简化来得到软件网络模型[110, 112]。

此外，考虑到 Java 语言是一种与平台无关的语言，软件产品经销商或开发者有时仅仅提供软件模块的字节码，在这种情况下如果我们还想分析系统的演化，只有对字节码文件进行操作了。因此，本文主要从中间代码构建软件网络模型，即从 Java 语言的 .class 或 .jar 文件构建。

[3]对于其它面向过程语言（如 C）开发的系统而言，多粒度的概念可以从语句、函数、文件、模块等体现。

3.3.2 数据收集

SMGEA-CNT 方法的第一步是收集数据。通过解析 Java 字节码文件（.class 文件和 .jar 文件）提取软件元素（方法、类、包）及它们之间的关联关系（这里考虑的关联关系见 3.3.3 节软件网络的定义）。本文所有的数据都是由我们自主开发的软件网络分析工具（Software Network Analysis Tool，简称 SNAT）自动提取的。SNAT 可以实现从面向对象软件字节码文件生成各个粒度（方法、类、包）的特定软件网络，然后对软件网络进行一系列的度量、分析和可视化等[151]，但该工具目前还不是开源的。当然，我们也可以借助一些开源的工具来获取所需的数据，如 XDepend[152]、DMS Software Reengineering Toolkit[153]、gprof[154] 等，具体的使用方式在这里不再赘述，请查看相关工具的操作手册（manual）。

3.3.3 软件网络

根据 3.3.2 节收集的数据，我们分别从方法、类[4]和包三个粒度上构建软件的网络模型（如定义 1、2、3 所示），形成面向对象软件的多粒度软件网络模型。这个多粒度软件网络模型是 SMGEA-CNT 方法的基础。下面首先给出多粒度软件网络模型各个粒度软件网络的定义：

定义1：方法调用网络（Method Call Network, MCN）

OO 软件系统的 MCN 可以用一个有向图表示，即 $\mathrm{MCN} = (N_m, E_m)$。其中：$N_m$ 为节点的集合，表示软件系统中所有方法的集合；E_m 是有向边的集合，表示方法之间的调用关系，并从调用者指向被调用者。

定义2：类关联网络（Class-Class Network, CCN）

OO 软件系统的 CCN 可以用一个有向图表示，即 $\mathrm{CCN} = (N_c, E_c)$。其中：$N_c$ 为节点的集合，表示软件系统中所有类的集合；E_c 是有向边的集合，表示类之间的关联关系。在CCN中，当一个类使用另外一个类提供的服务时，这两个类之间便发生了关联关系。边的方向从服务使用者指向服务提供者。因此，在以下 4 中情况下，类 A 和类 B 之间就会存在一条有向边 $A \to B$：

（1）如果类 A 使用关键词 extends 继承自另外一个类 B。

（2）类 A 使用关键词 implements 实现了接口 B。

（3）类 B 拥有一个类型是 A 的属性。

[4]本文将接口（interface）也当作类看待。因此，若不作特别声明，本文中提到类时其实也包括了接口。

（4）如果类 A 的方法调用了类 B 一个对象的方法。

定义3：包关联网络（Package-Package Network, PPN）

OO 软件系统的 PPN 可以用一个有向图表示，即 PPN $= (N_p, E_p)$。其中：N_p 为节点的集合，表示软件系统中所有包的集合；E_p 是有向边的集合，表示包之间的关联关系。在 PPN 中，包之间的关联关系间接来源于包所包含的类之间的关联关系。也就是说，当一个包中的一个类，与另外一个包中的一个类有关联关系，那么这两个包之间也存在关联关系，边的方向与类之间的方向一致。

图 3.3 给出了一个源代码片段及其对应的不同粒度的软件网络。

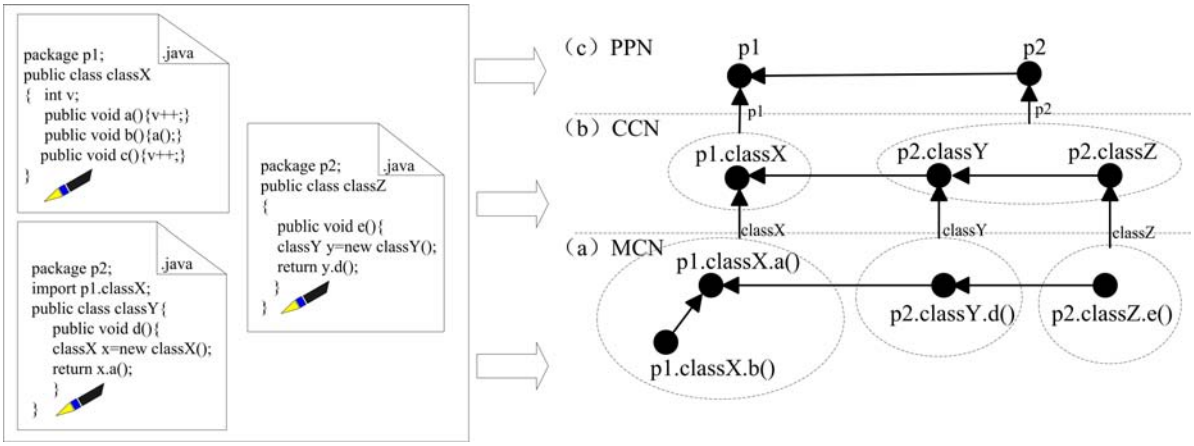


图 3.3　Java 源代码片段及其对应的软件网络

3.4　实验设计

为了说明如何使用 SMGEA-CNT 方法分析面向对象软件的演化，本节将结合开源软件 Azureus[155] 进行实例分析。3.4.1 小节将首先介绍一下实验系统 Azureus；3.4.2 小节将使用复杂网络中的参数从各个不同的角度详细地分析 Azureus 的演化。通过实例分析，我们在揭示 Azureus 演化特点和规律的同时将指出这些演化规律对软件工程实践的指导意义。

3.4.1　系统简介

Azureus 是一款著名的 BitTorrent 客户端软件，其源代码和其它相关资料可以从 [155] 处下载。在收集相关数据时，我们发现，Azureus 版本 0.9.0 至版本 3.0.0.3 其版本号和发布日期存在不一致，软件发布时间早，其版本号却大。因此，本章中我们选择版

本 3.0.0.8 到工作开展时的最新版本 4.5.0.2 共 39 个软件作为研究对象，而版本 3.0.0.8 之前的几个版本不在研究之列。

表 3.1 所示的是 Azureus 的一些统计数据，包括第一个版本（版本 3.0.0.8）和最后一个版本（版本 4.5.0.2）的方法数、类数和包数，以及项目总共的版本数和已持续的时间。

表 3.1　Azureus 的统计数据

参数名	参数值	参数名	参数值
3.0.0.8 版本方法数	25,119	4.5.0.2 版本方法数	39,711
3.0.0.8 版本类数	4.945	4.5.0.2 版本类数	7,662
3.0.0.8 版本包数	417	4.5.0.2 版本包数	476
项目版本数	39	项目已持续月数	40

3.4.2　实验分析

这一节，我们将使用复杂网络理论从不同粒度研究 Azureus 软件结构的演化。限于篇幅，我们仅选择了一些对于研究软件结构特别重要的参数，其它更多的参数请参见综述 [156]。下面将分节阐述。

3.4.2.1　节点和边的多粒度演化

软件网络的基本构成单元是节点和边，并且节点数常被看作是网络的规模。因此，研究软件节点数和边数的演化，可以了解软件规模的变化。在这一小节，我们将分析不同粒度软件网络中节点数和边数在 39 个软件版本中的演化情况，研究不同粒度软件规模的变化规律。图 3.4 所示的是节点和边的演化。其中：纵坐标是特定版本软件网络的节点/边数，横坐标是 Azureus 某一个版本的序列号。这里我们使用软件的发布序列号 RSN（release sequence number）[157] 作为软件某一个版本的标识符，而不使用软件的版本号。RSN 是一个顺序号，它根据时间的先后来依次给软件进行标识，即：第一个版本的 RSN 是 1，第二个版本的 RSN 是 2，依次类推。也就是说，Azureus 的 3.0.0.8 版本的 RSN 是 1，最后一个版本 4.5.0.2 的 RSN 是 39。

从图 3.4 可见，MCN、CCN 和 PPN 中节点数和边数都随 RSN 不断增长，但是增长速度有差异，增长速度从方法粒度到包粒度不断下降。这表明，在软件演化的过程

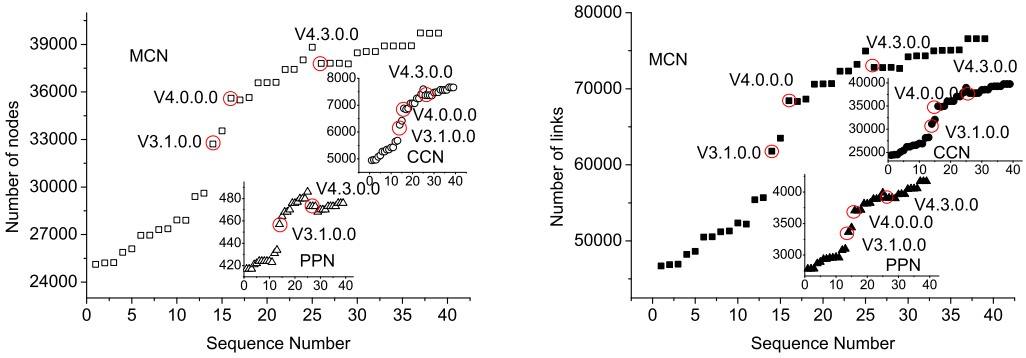


图 3.4　节点（左）/边（右）的演化

中，软件的维护活动主要集中在方法粒度（方法本身或方法之间调用关系的增加比较频繁），然而类和包粒度的改变不是很大，变化比较平缓。这比较容易理解，因为软件在开发出来以后，其整体的功能模块不会有太大的调整，只是一些小功能的变更和完善。同时，随着粒度的降低（从包粒度到方法粒度），网络的节点数（边数）不断增大：节点数（边数）从包粒度到类粒度增大了 10 多（约 9）倍；节点数（边数）从类粒度到方法粒度增大了约 5（2）倍。

从图 3.4 我们看到了几个比较特殊的演化点（红色圆圈部分），在这几个演化点节点数/边数变化比较大。边上的文字（V*.*.*.*）是这个点对应软件的版本号。为了研究这些特殊点的成因，我们查阅了它们的发布日志。我们发现在版本 3.1.0.0 的代码中共有 26 处变更，其中 16 处是新功能的添加，10 处只是性能的改善和 bug 的修复。在版本 4.0.0.0 的代码中共有 54 处变更，其中 20 处是新功能的添加，34 处是性能的改善和 bug 的修复。因为性能的提高及 bug 的修复引起的变更并不会对网络节点与边的数量形成太大的影响，而新功能的添加使得节点数和边数不断增长，所以这两个版本节点数与边数增长幅度比较大。同时，在版本 4.3.0.0 中共有 68 处变动，其中有 18 处是新功能的添加，21 处是性能的提高和 bug 的修复，但此处在性能提高与 bug 修复过程中，之前的一些功能被删除了，导致节点与边的数量小幅度减少。

此外，我们也可以看出，节点的增长与边的增长大体都呈“超线性”增长趋势，尤其是 4.3.0.0 之前的版本，这与 [102] 中的结果不谋而合。在软件项目管理与软件成本预测方面，节点与边的“超线性”增长发挥着非常重要的作用。如果随意地忽略它或是简单的用线性增长代替，则将低估软件的复杂度与开发费用，对企业造成的损失以及产生的负面影响可能是巨大的。

3.4.2.2 平均度的多粒度演化分析

节点 n 的度 $deg(n)$ 被定义为与该节点 n 相连的其它节点的数量，它是节点的一个重要特征。直观上看，一个节点的度越大就意味着这个节点在某种意义上越“重要”。基于节点的度，我们可以定义网络的平均度 $< k >= \sum_{n \in N} deg(n)/|N|$。其中：$N$ 是一个特定网络所有节点的集合，$deg(n)$ 是节点 n 的度。

A. L. Barabási 等研究过科学家合作网平均度 $< k >$ 的演化情况，发现 $< k >$ 随时间的变化趋近线性增长，这表明科学家合作网是一个“加速增长”（accelerating growth）的网络[158]。A. P. Masucci 等对人类语言网进行了研究，发现人类语言网络也是一个“加速增长”网络，且 t 时刻词语总数 $N(t)$ 与时间 t 满足关系 $N(t) \sim t^{1.8}$[159]。

平均度是描述网络连通性的一个重要参数，$< k >$ 越大，网络连通性越好。对于软件网络而言，平均度可以间接的反映软件中消息的传递效率。此处，我们将对从 Azureus 构建的软件网络的平均度 $< k >$ 做分析，检测该软件网络是否也具有“加速增长”特性。图 3.5 显示的是不同粒度软件网络平均度 $< k >$ 与节点数在双对数轴上的曲线图（log-log plot[160]），并给出了相应的线性拟合直线及评价拟合精度的指标 R^2。

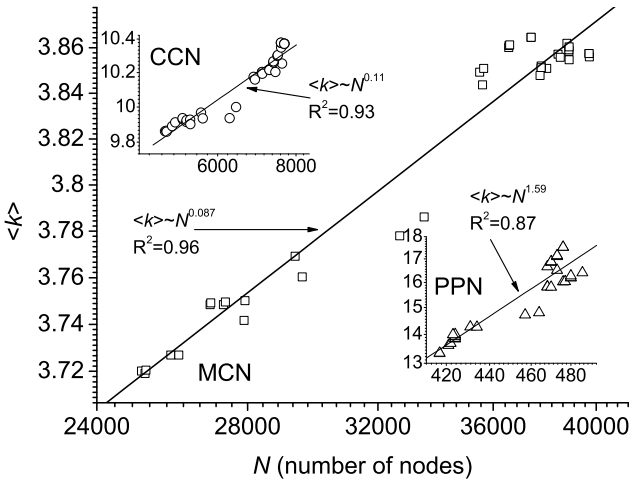


图 3.5　双对数轴（log-log plot）中 $< k >$ *vs.* 节点数

从图 3.5 我们可以发现：Azureus 不同粒度的软件网络其 $< k >$ 和节点数 N 之间的关系大致符合 $< k > \sim N^{\beta}$（见直线拟合），但是其指数 β 随粒度的不同而不同，MCN、CCN 和 PPN 的指数分别是 0.087、0.11 和 1.59。这表明，这三类软件网络都是“加速增长”网络，平均度 $< k >$ 随着 N 的增长不断增加，但是速度非常缓慢。这种情况对于 MCN 和 CCN 更加明显，尽管在这连续的 39 个版本中，节点数获得了大幅度的增长，但是 $< k >$ 的增加不到 1。加速增长现象在 PPN 中更加明显，$< k >$ 增长

了 4，但是与包数增加的 59 来说，这种增加的量还是比较微弱的。我们将这种“加速增长”现象称为“弱加速增长”（weak accelerating growth）。“弱加速增长”是软件演化的一个新特点，这个特点被很多已有的软件演化模型所忽略，如软件增长模型 DDEA 和 DAAE[161] 都没有考虑“弱加速增长”这一特性。

$< k >$ 和 N 的关系 $< k > \sim N^{\beta}$ 可用于预测项目的成本。如果可以在项目开发的早期估算系统中包、类或方法之一的数量，那么就可以根据这个公式预测系统相应粒度软件网络连接的数量。如果以前还有本项目或类似项目节点和边的平均费用，那么我们就可以预测现在这个系统的成本。同样的，跟“超线性”增长类似，对一个存在“弱加速连接”现象的庞大工程系统而言，如果随意地忽略它并按匀速连接对该工程系统进行工期预测与成本预测，事必会造成巨大的损失。

3.4.2.3 度分布的多粒度演化分析

度分布 $P(k)$ 常用来刻画网络的结构特征，用于描述网络中节点度的分布情况。度分布表示一个随机选择的节点度恰好是 k 的概率，数学上，常用公式 $P(K = k) \sim k^{-\alpha}$ 进行描述。目前，我们常用度分布来检测一个网络是否是“无标度”的，也就是说，当网络的度分布 $P(k)$ 符合幂律分布（power-law）时，那么这个网络就是“无标度”网络[43]。很多情况下，幂律也常用累积度分布来定义，即 $P(K > k) \sim k^{-\alpha'}$。这两种定义方式在本质上是等价的，并且幂指数存在关系：$\alpha = \alpha' + 1$。

对于有向网络（网络中的边是有向边），我们有必要区分出度（out-degree）和入度（in-degree），出度是指从该节点指向其它节点的边的数目；入度是指从其它节点指向该节点的边的数目。对于软件网络，一个节点的入度是对代码重用情况的一种度量，即具有较大入度的节点对应的代码会在很多情景下被重用；具有较小入度的节点对应的代码重用不高。另一方面，软件出度是软件设计复杂性的一种度量，即具有较大出度的节点其功能越强，它融合了很多其它节点的功能，因此其设计比较复杂；具有较小出度的节点相对比较简单[62]。

类似节点的度分布表示一个随机选择的节点度恰好是 k 的概率，节点的入度（出度）分布 $P^{in}(k)$（$P^{out}(k)$）表示一个随机选择的节点具有入度（出度）k 的概率。本章我们分析了各个粒度软件网络的累积入度（出度）分布 $P^{in}_{cum}(k)$（$P^{out}_{cum}(k)$），发现所有度分布均具有近似幂律特性，即遵循一个幂律 $P_{cum}(k) \sim k^{-\alpha'}$ 或具有指数截止（exponential cutoff）的幂律 $P_{cum}(k) \sim k^{-\alpha'} e^{-\lambda k}$。对于具有指数截止的幂律，我们可以对曲线做适当的截取使得其更好的符合幂律分布。考虑篇幅限定，这里我们只给出 Azureus 4.5.0.2

版本的累积入度（出度）的分布，及相应的直线拟合和 R^2，详见图 3.6(a)-(c)。

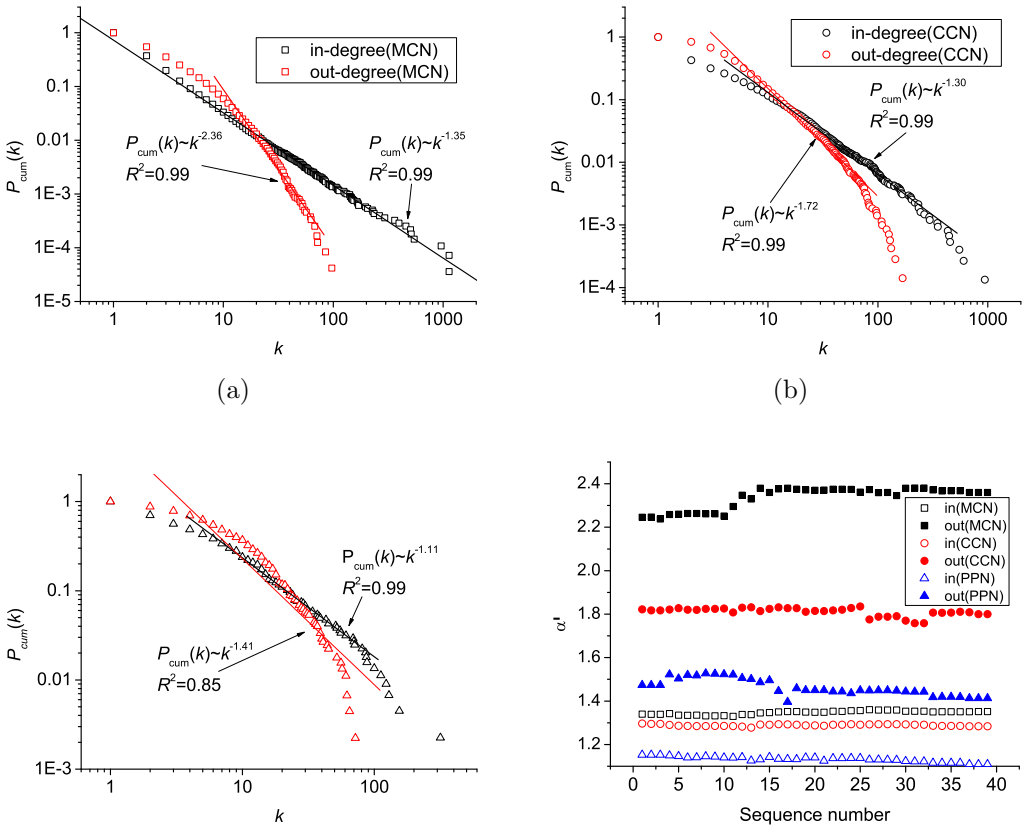


图 3.6　(a)-(c)：Azureus 4.5.0.2 版本的 $P_{cum}(k)$；(d)：度分布指数 α' 的演化

网络的无标度特性对软件系统极为重要：(1) 它与软件的鲁棒性密切相关。一个无标度网络对随机错误的容错性比随机网络更强，这正是为何许多软件在存在缺陷的情况下仍然可以正常运行的原因。但当错误发生在那些度很大的节点上时，就有可能使整个系统面临崩溃的危险。(2) 对于软件测试的意义。既然每个节点有着不同的重要性，那么在软件测试过程中，可以着重测试那些出/入度大的节点。大入度节点重用度高，涉及面广，一旦存在错误可能影响大量其它节点；大出度节点对其它节点的依赖性强，融合了很多其它节点的功能，对软件的复杂性具有重要的影响，这些节点存在错误的可能性也偏高。

图 3.6(d) 是累积出度（入度）分布幂指数的演化。从图 3.6(d) 可见，不同粒度软件网络节点的出度与入度分布是不平衡的，即对一个特定粒度而言，累积出度分布幂指数明显要比相应粒度累积入度分布的幂指数要大。在 Azureus 4.5.0.2 版本中，MCN：$\alpha'^{out}(2.36) > \alpha'^{in}(1.35)$；CCN：$\alpha'^{out}(1.72) > \alpha'^{in}(1.30)$；PPN：$\alpha'^{out}(1.41) > \alpha'^{in}(1.11)$。这种不平衡表明，在一个特定的软件网络中入度大于出度的节点比较常见，

这可能与软件工程实践中强调重用原则有关，重用次数越多，入度就会越大，因此，在软件网络中节点入度比出度大也是正常的。

此外，图 3.6(d) 显示不同粒度网络的累积出度（入度）分布幂指数各自保持相对稳定，即其演化曲线几乎平行于水平轴，也就是说，幂指数是有界的。这说明，幂指数在连续两个版本间的变化很微小，它基本与我们对软件系统所做的变更（软件网络中节点和边的变更）没太大的相关性。据我们所知，这一特性在以前的工作中未曾报道过，因此也被很多现有的软件演化模型所忽视。

3.4.2.4 平均路径长度和聚集系数的多粒度演化分析

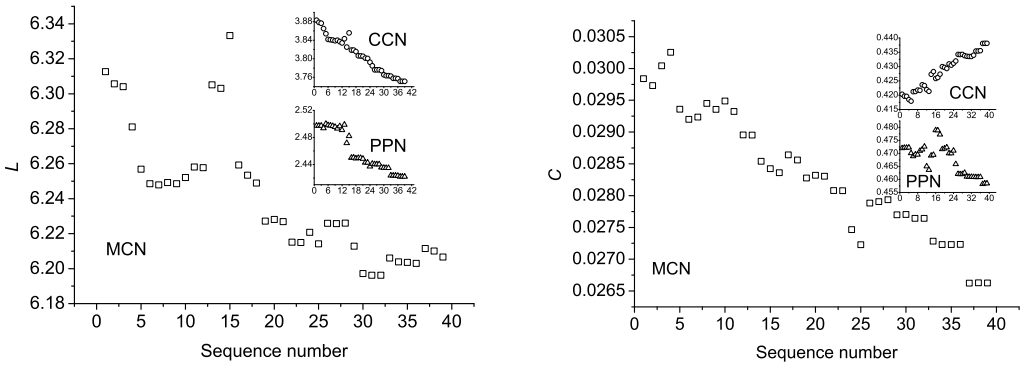
任意节点对 i 和 j 之间的最短路径 d_{ij} 是连接这两个节点的所有路径中具有最少边数的那条路径。网络的平均路径长度 $L =< d_{ij} >$ 是网络中所有节点对之间 d_{ij} 的平均值。

网络的平均路径长度 L 是网络的一个重要特征，在软件网络中也有比较重要的意义，它表征了软件中节点之间传递信息的能力，修改和调试软件的开销，以及响应能力[62]。一般，随着软件 L 的逐渐减小，消息传递变得更快，修改和调试变的容易，响应更快。图 3.7(a) 显示的是平均路径长度 L 的演化。从图 3.7(a) 可见，各个粒度软件网络 L 随演化不断减小，这表明软件系统的信息传递变的更加高效。我们认为，软件网络 L 这种减小趋势可能源于内部连接的增多，比如之前没有相连的两个点间增加了一条边使其相连，因而提高了网络的连通性。这一特性对构建软件演化模型具有指导意义，即：在添加新节点，并将新节点与已有节点相连的同时，已有节点间也会添加新的边。

节点 i 的聚集系数 C_i 反映它的邻居之间成为邻居的可能性大小。如果用 l_i 表示节点 i 的邻接节点间的实际边数，k_i 表示节点 i 的邻接节点总数，那么 C_i 可以通过 $C_i = 2l_i/k_i(k_i - 1)$ 来计算。而整个网络的聚集系数是所有节点聚集系数的平均值，即 $C =< C_i >$。

3.7(b) 显示的是 Azureus 各粒度软件网络聚集系数的演化。从 3.7(b) 可见，在 MCN 和 PPN 网络中，C 以一个很慢的速度不断下降，而在 CCN 中 C 缓慢增加。这表明，随着软件的演化，调用了相同类的类彼此间调用的可能性比方法和包粒度的更大，且不同粒度的软件网络的聚集系统变化范围各不相同。对于 Azureus 而言，各个粒度 C 的变化范围分别是：MCN：$C \in [0.02, 0.04]$；CCN：$C \in [0.41, 0.44]$；PPN：$C \in [0.45, 0.48]$。PPN 和 CCN 的 C 比较接近约为0.4，而 MCN 的 C 相对较小。

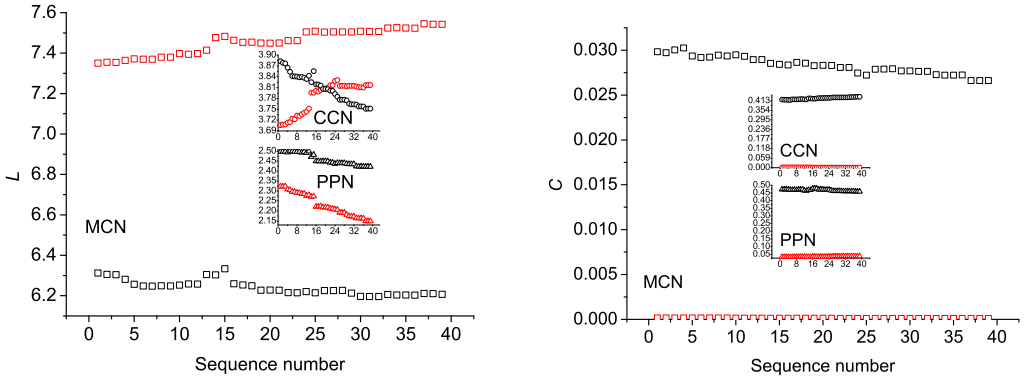
一般我们将具有较小平均路径长度和较大聚集系数的网络称之为“小世界”网络[42]。



(a) (b)

图 3.7　Azureus 不同粒度软件网络 L 和 C 的演化信息

图 3.8(a)-(b) 分别显示了 Azureus 三个粒度软件网络平均路径长度 L 和聚集系数 C 的演化信息及相同条件下（相同节点/边数）随机网络对应 L 和 C 的演化信息。从 3.8(a)-(b) 可见，各个粒度的软件网络均满足“小世界”网络特性，即它们的平均路径长度与随机网络的的很接近且比较小，但聚集系数却比相应随机网络的大很多：MCN 中 C 是随机网络的 170 多倍；CCN中 C 是随机网络的 200 多倍; PPN 中 C 是随机网络的 10 多倍。



(a) (b)

图 3.8　Azureus 不同粒度软件网络 L 和 C 的演化信息（黑色）及同等规模（相同节点数和边数）随机网络 L 和 C 的演化信息（红色）

3.4.2.5　社区结构的多粒度演化分析

在诸多现实网络中，社区结构非常普遍，如社会网络、生物网络和航空交通网络等都存在明显的社区结构。社区结构是一种不均匀的结构组织方式，社区内部节点间的连接比较紧密，社区之间的连接比较稀疏。

面向对象软件系统通常由一个个包构成，而一个包又包含许多类，一个类又是方法与属性的集合。同时，软件工程实践又鼓励“高内聚，低耦合”的设计原则。软件元素的这种组织与开发原则，形成了相应软件网络中的社区结构现象。软件网络的社区结构研究为理解网络功能与拓扑间的相互关系提供了新的视角。

M. E. J. Newman 等提出用模块度 Q（modularity）来衡量某一特定网络社区划分质量，其计算公式如下[162]：

$$Q = \sum_i (e_{ii} - a_i^2), \tag{3.1}$$

其中：e_{ii} 表示网络中连接社区 i 内任意两节点的边在所有边中占的比例；a_i 表示与第 i 个社区中节点相连的边（至少有一个端点属于社区 i 的边）在所有边中占的比例。这里我们使用 Q 来衡量软件网络的模块度，并使用 M. E. J.Newman 等提出快速算法对软件网络进行社区划分及求 Q，算法详见 [162]。

我们研究了软件系统不同粒度模块度的演化。限于篇幅，我们这里只给出了类粒度软件网络模块度的演化情况（如图 3.9）。从图 3.9 可见，Azureus 类粒度模块度在演化的过程中不断减小。演化是软件的基本特征之一，但是由于时间和各种资源的限制，软件的变更必须在很短的时间内完成，很多时候没有进行充分的考虑。软件系统变的越来越复杂，与最初的设计越离越远，软件系统的模块度不断下降，质量不断下降。因此，为了提高软件的质量，我们必须不断的重构软件系统。那么我们究竟应该在什么时候对软件进行重构呢？本章我们提出通过跟踪软件模块度的演化来确定重构时间点（在哪个版本实施重构）的方法。为了确定重构时间点，我们首先定义模块度的变化率 CRM（change rate of modularity）。软件连续两个版本 i 和 $i+1$ 间的 CRM 定义为：

$$CRM(i, i+1) = \begin{cases} (Q_i - Q_{i+1})/Q_i, & Q_{i+1} \le Q_i \\ 0, & Q_{i+1} > Q_i, \end{cases} \tag{3.2}$$

其中：Q_i 和 Q_{i+1} 分别是版本 i 和 $i+1$ 对应的模块度。我们认为，如果两个相邻版本间的模块度存在明显的下降，则表示存在一个重构时间点，本章我们定义阈值 0.02 来过滤出存在的重构时间点[5]。如图 3.9，我们已用红圈标识了找到的 4 个重构时间点，它们分别对应 Azureus 的 3.0.1.4、3.1.1.0、4.1.0.0 和 4.3.0.2 版本。同时，我们也可以

[5]过滤值的设置目前还主要是凭借经验，即如果想使重构时间点多一点，则过滤值可以设置的相对小些；如果想使重构时间点少一点，则过滤值可以设置的相对大些。

通过 Q 来优化软件的结构，即：通过移动软件元素来最大化 Q 以增强内聚，减少耦合（详见本文第 5 章）。

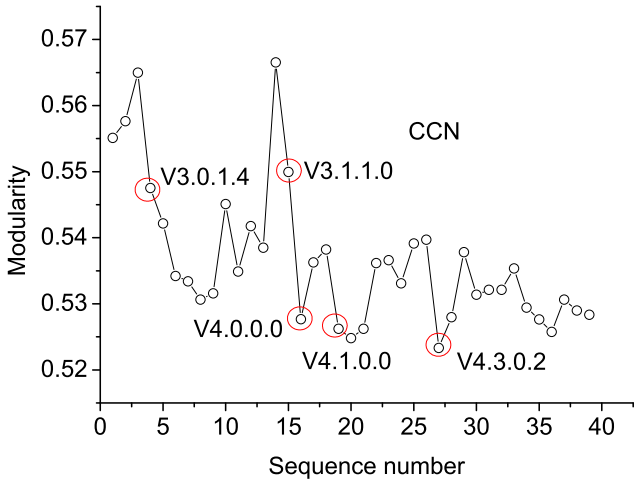


图 3.9　Azureus 类粒度软件网络模块度的演化

3.4.2.6　同配系数的多粒度演化分析

很多实际网络在度上都呈现混合模式（mixing patterns），即同配混合（assortative mixing）或异配混合（disassortative mixing）。所谓的同配混合是指大度节点趋于与其它大度节点相连，低度节点更可能与其它低度节点相连。所谓异配混合是指大度节点趋于与其它低度节点相连，或者说，低度节点趋于与其它高度节点相连。这种连接倾向性主要反映网络中节点度分布与其直接相连节点（邻居节点）度分布之间的关系。

M. E. J. Newman 首次在复杂网络中引入了同配系数 r 以测量节点间的连接倾向性，定义如下[163]：

$$r=\frac{\sum_i j_i k_i - M^{-1}\sum_i j_i \sum_i k_i}{\sqrt{[\sum_i j_i^2 - M^{-1}(\sum_i j_i)^2][\sum_i k_i^2 - M^{-1}(\sum_i k_i)^2]}}, \tag{3.3}$$

其中：j_i 和 k_i 分别是第 i 条边的入度和出度，M 是网络的总边数。若同配系数 $r>0$，则网络是同配网络；若 $r<0$，则网络为异配网络；若 $r=0$，则为随机网络。同配系数体现网络节点的连接倾向，同配网络中高度值节点倾向于和高度值节点相连，异配网络中高度节点则倾向于和低度节点相连。研究表明，社会网络大多是同配网络，而生物网络和技术网络往往是异配网络。同配系数具有很强的现实意义，如异配网络对随机失效和选择性攻击都比较脆弱，它易于病毒的传播，而同配网络则正好相反。

软件网络若为同配网络，则说明同其它软件元素（方法、类或包）联系紧密的元素节点之间更容易发生（继承或使用等）关联关系，软件元素复用率较高。据我们所知，软件网络中关于同配系数的相关研究还非常少。本节将研究 Azureus 不同粒度软件网络同配系数的演化（如图 3.10）。我们发现了一个很有趣的现象，即：所有类粒度软件网络（CCN）与包粒度软件网络（PPN）在度上都表现出很弱的异配性，而所有方法粒度软件网络（MCN）都表现出很弱的同配性。这说明在 Azureus 软件的开发过程中，复用率高的软件元素之间发生（调用或继承等）关联关系的倾向性过弱。同时，我们也发现，各个粒度 r 的演化曲线非常稳定，在总共 39 个版本中，变化比较平稳，即 MCN: $r \in [0.005, 0.009]$； CCN: $r \in [-0.051, -0.045]$； PPN: $r \in [-0.042, -0.017]$。查阅开发日志可知，在 Azureus 的整个发展过程中，其基本架构比较稳定，因此同配系数变化比较小。从图 3.10 可见，所有 r 的值都几乎接近 0 ，这与 [163] 中在包粒度的实验结果一致，表明 Azureus 软件网络在度上没有很明显的混合模式，我们把这种混合模式叫“弱混合”模式（weak mixing patterns）。对于软件系统而言，“弱混合”模式有助于分析软件元素之间的协作关系，表示复杂的软件元素更倾向于由相对简单的元素构成，体现了软件的构造性原则。软件网络的这种“弱混合”模式也被很多软件演化模型忽视。

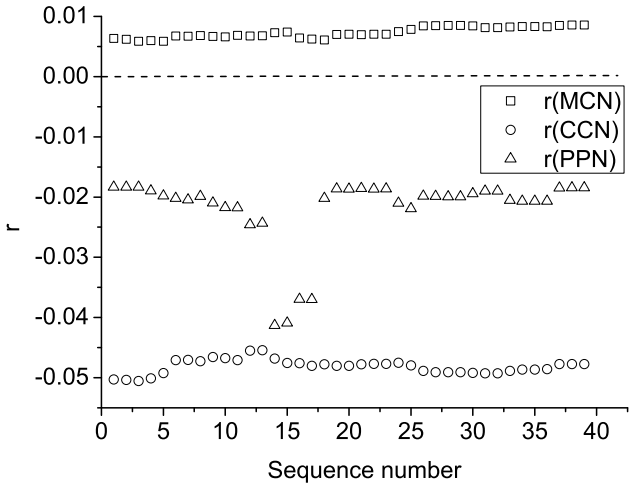


图 3.10　Azureus 不同粒度软件网络同配系数的演化

3.4.2.7　“富人俱乐部”现象的多粒度演化分析

“富人俱乐部”现象描述了少量具有大度的节点间的紧密交互，这些节点也称为“富节点”即“hubs”。我们可以使用“富人俱乐部”连通性 $\phi(r/N)$ 来量化这种现象，它表示网络中前 r 个度最大的节点之间实际存在的边数 L 与 r 个节点之间总的可能存在

的边数 $r(r-1)/2$ 的比值，即

$$\phi(r/N) = \frac{2L}{r(r-1)}. \tag{3.4}$$

其中：N 表示相应网络的总节点数。如果 $\phi(r/N) = 1$，表示前 r 个节点组成的“富人俱乐部”是完全连通的子图，如果 $\phi(r/N) = 0$，表示这些节点是彼此孤立的。“富人俱乐部”现象与网络的同配性尽管有一定的相似性，但是存在很大的区别，“富人俱乐部”现象描述的是“hubs”节点间的连接关系，而网络的同配性描述的是任一连接的节点间的度相关性。

“富人俱乐部”的连通性对信息传递效率和网络的鲁棒性等都有重要的意义。我们研究了 Azureus 各个粒度软件网络的“富人俱乐部”连通性，发现 $\phi(r/N)$ 在每个版本表现很相似，这里我们仅给出 4.5.0.2 版本获得的相关数据（如图 3.11）。从图 3.11 可见，CCN 和 PPN 网络的“富人俱乐部”现象比较明显，“富节点”间连接的非常紧密。在 CCN 中，前 1.04‰（8 个）富节点拥有 28.571% 的边；在 PPN 中，前 18.487% 个富节点拥有 30.016% 的最大可能边；在 MCN 中，“富人俱乐部”现象就不怎么明显，在前 1.1763‰的富节点中，连通性的最大值只有 0.09524。对于软件网络而言，该现象有助于分析不同类型软件实体间的协作关系。

3.5 本章小结

本章应用复杂网络的相关理论和方法，从方法、类和包三个粒度详细分析了面向对象软件系统的演化，首次从多粒度角度系统的揭示软件系统的演化动力学特性和规律。为了说明我们方法的使用方式，我们以开源 Java 软件 Azureus 的 39 个连续版本（从版本 3.0.0.8 到版本 4.0.5.2 版本）作为研究对象进行了案例分析，发现了其结构具有一些明显的规律，概括如下：(1) 各个粒度软件网络规模（节点数、边数）增长呈“超线性”增长特点，但是增长速度从方法粒度到包粒度不断下降；(2) 各个粒度软件网络平均度呈“弱加速”增长特点；(3) 各个粒度软件网络都是“小世界”和“无标度”网络，并且度分布的幂指数在一个很小的范围内波动；(4) 类和包粒度软件网络是“弱异配”网络，而方法粒度上的软件网络是“弱同配”网络；(5) 类和包粒度软件网络表现出明显的“富人俱乐部”现象，而方法粒度的不明显。此外，通过跟踪软件模块度的变化率，我们提出了一种确定软件重构时间点的方法。这些规律的发现可以辅助理解软件系统设计的演化过程及趋势，并指导软件项目管理、成本预测以及软件过程中的软件测试、

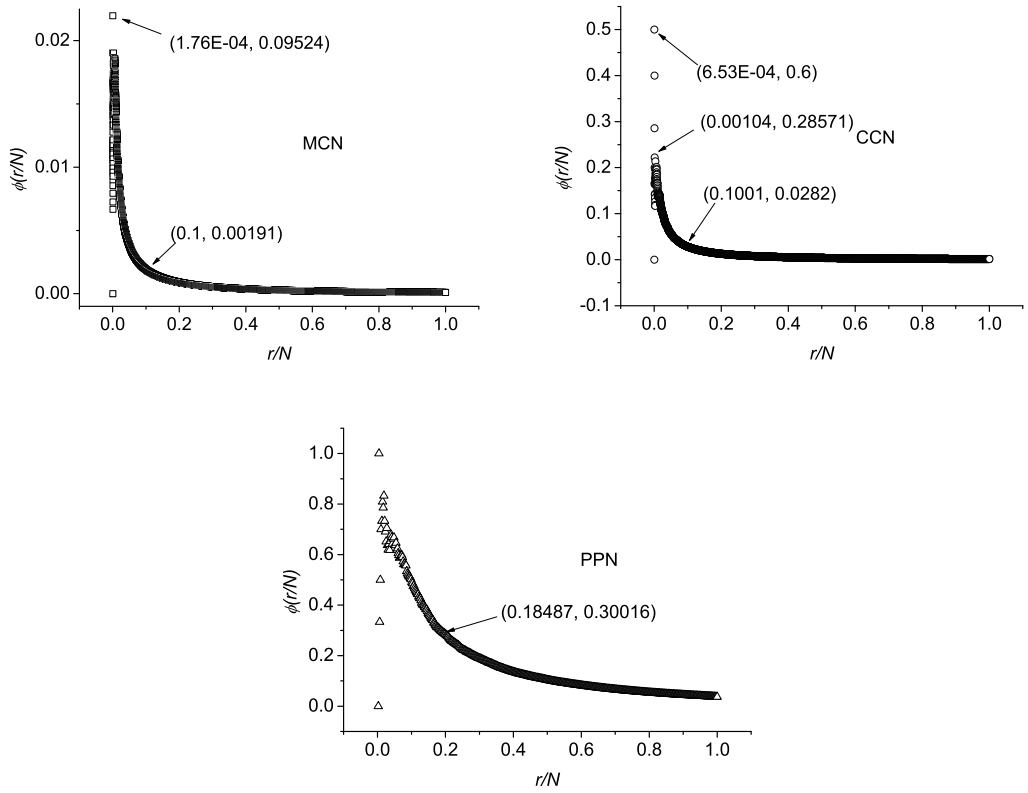


图 3.11　Azureus 的 4.0.5.2 版本不同粒度的富人俱乐部连通性

软件维护等。同时对于构建软件演化模型也具有重要意义。

下一步我们考虑在现有工作的基础上，从以下几个方面继续开展工作：(1) 选择更多的软件验证本文发现的规律是否具有普适性；(2) 构建软件网络的增长模型，解释上述规律形成的内在机制；(3) 关注一种特定的交互关系（如只考虑类之间的继承关系），然后构建相应的软件网络，分析该网络的演化特性。

本章实验部分的所有数据，包括从 Azureus 39 个版本构建的软件网络数据文件、实验的部分工具软件、实验结果数据等都可以从 [164] 下载。

第四章　软件稳定性分析及结构优选

"*Stability is at the very heart of all software design*[165]."

—— *M. Fowler*

4.1　引言

软件维护是软件生存期中最长的一个阶段，其所花费的人力、物力等资源占了软件总费用的大部分，并且逐渐提高，从 1979 年的 67% 上升到 2000 年的 90% 以上[166]。软件维护费用如此之高，使得如何控制软件维护费用成为一个亟待解决的问题。S. Yau 和 J. S. Collofello 提出了两种用于控制软件维护费用的策略[167]：(1) 提供工具和技术简化工作人员的维护工作；(2) 提供或使用一些有意义的软件度量。本章主要关注第二种方法，即通过构建一些度量来对影响软件维护费用的质量属性进行评价。

IEEE 将软件维护定义为[168]：在软件产品发布之后，对软件系统或其部件进行修改的过程，包括错误的修正、性能或其它属性的改善，或是修改系统以适应新的运行环境。软件维护一般由四个阶段构成[167, 169]：(1) 分析和理解软件系统；(2) 提出一些具体的修改建议以帮助实现预期的维护目标；(3) 分析变更的"涟漪效应"（ripple effect）；(4) 对修改后的软件系统重新进行测试，以检验修改是否保持了软件系统原有的可靠性。从第 3 章的分析我们已经知道，随着与软件交互的环境的变化，软件的变更在所难免。此外，我们常有这样的经验，对软件某部分元素（数据、属性、方法、类等）的变更往往会引起其它相关元素的变更，从而引发一连串的变更，这就是变更的"涟漪效应"。为了研究软件变更的这种"涟漪效应"，人们提出了很多变更影响分析方法（change impact analysis），希望借此为变更的预测和实施提供支持。目前，实施变更和变更影响分析在软件维护中占有重要的地位，占了软件维护总费用的 40% 以上[15]。因此，为了降低软件的维护费用，我们必须实施有效的变更影响分析。

软件稳定性是影响软件变更影响分析的重要属性之一，它用于描述软件对变更放大作用的抵抗能力[167, 169]。如果一个软件的稳定性差，那么对软件元素的变更就会引起其它软件元素大范围的变更，软件维护的费用就会很高。同时，大范围的变更也使得引入错误的可能性增大，软件的可靠性和质量就会下降。软件稳定性如此重要，但是目前这方面的研究工作相对还比较缺乏。

与此同时，相同的功能需求可以通过不同的软件结构来实现，而这些不同的结构在

稳定性上往往有所不同。但是，一直以来软件系统的结构与其稳定性之间的量化关系不清楚，开发人员很难弄明白什么样的软件结构其稳定性才是好的。因此，我们必须采取新适当的方法和工具来研究软件结构和软件稳定性的关系，提出评价软件稳定性的方法，进而对软件的质量进行评估，这可以帮助我们实现结构的优选，即从多个具有相同功能不同结构的软件中选择结构质量高的软件。

复杂网络为我们研究软件结构提供了有力的工具。本章基于软件的网络结构，提出了一种从特征（feature）[1]粒度评价软件结构稳定性的方法 SoS-SN （stability of object-oriented software measurement via simulation in software networks）。SoS-SN 方法：(1) 将面向对象软件系统在特征粒度抽象成加权软件网络——加权特征依赖网络 WFDN（weighted feature dependency networks）；(2) 使用一种仿真的方法分析变更在 WFDN 上的传播过程，并定义软件稳定性度量指标 *SoS*（stability of software）；(3) 从理论分析和实验验证两个角度说明 *SoS* 的合理性和有效性。SoS-SN 方法获得的软件稳定性值 *SoS* 可以用来对软件结构进行优选。

本章后续内容的组织结构如下：4.2 节介绍相关研究内容；4.3 节介绍 SoS-SN 方法，包括：数据的收集方法、软件网络 WFDN 的定义、软件稳定性指标 *SoS* 的定义及计算方法、*SoS* 的收敛性分析等；4.4 节对 *SoS* 的有效性进行理论证明；4.5 节选择了八组 Java 程序，通过数据实验说明 *SoS* 的有效性，同时也说明其在结构优选方面的作用；最后在 4.6 节对本章进行小结。

4.2 相关研究工作

变更影响分析主要用于预测软件某些部分的变更对其它部分（代码、文档等）的影响[170]，它对于提高软件维护的效率至关重要。SoS-SN 方法通过变更影响分析来度量软件的稳定性，并对结构进行优选。因此，本节我们将简单地综述一下变更影响分析方面的研究工作。

S. Yau 和 J. S. Collofello 以软件设计文档为载体研究软件的稳定性[167, 169]。他们从软件设计文档分析出程序模块（module）及模块之间的调用关系、全局数据等，然后提出了一些度量用于估算软件稳定性，并提出了一个算法用于计算这些度量值。这些度量值反应了软件对变更的适应能力，能够在软件生命周期的任何时间点使用，可以为解决

[1]为了描述方便，在本章中我们用特征来指代属性和方法。因为类中的方法描述的是类的动作（动态特征），属性描述的是类的状态（静态特征），都用于描述类的特征。因此，本章同等看待方法和属性，不作区分。

软件维护中碰到的问题提供指导。该方法主要是以软件设计文档为研究载体的，而本章提出的 SoS-SN 方法主要以软件源代码为研究载体，然后以逆向工程的方法提取软件特征及其间的关系，并构建软件网络，计算软件稳定性。

D. Kung 等基于类图（class diagram）进行软件的变更影响分析[171]。他们分析了软件源代码变更的种类，包括数据变更、方法变更、类变更和类库变更，并提出类防火墙（class firewall）用于描述一个类的变更可能影响的类的集合。同时，他们又利用变更影响分析的结果来对回归测试阶段的测试用例进行排序。在这个方法中，D. Kung 等假设类的变更会 100% 影响与这个变更类有直接或间接关系的其它类，而在本章提出的 SoS-SN 方法中，我们分析的粒度在特征粒度，同时分析的时候，我们假设变更是以一定的概率进行传播的，关注的也不仅仅是变更传播问题和测试问题，而是讨论软件整体的稳定性度量问题。

T. Zimmermann 等根据软件的变更历史，研究了软件元素（方法及属性）间的变更共现（变更同时出现）关系，以此研究软件如何演化和为什么演化[172]。该方法可以给出在某个软件元素发生变更时，推荐其它也需要变更的软件元素。该研究主要关注软件元素的变更共现关系，SoS-SN 方法主要研究软件的稳定性度量问题。

N. Tsantalis 等在类粒度提出了一种评价面向对象软件结构灵活性（flexibility）的概率模型[173]。这个模型估算一个特定类受代码修改（新功能的添加、旧功能的修改等）影响的概率。可见，当这个系统对变更很敏感时，其对应的设计显然是有问题的。通过比较两个或多个软件系统设计对变更的敏感程度，可以判断当前的开发是否遵循了某些设计的原则。该研究主要在类粒度展开研究，探讨一个类变更时，其它类也要变更的概率，SoS-SN 方法主要在特征粒度研究软件整体结构的稳定性。

Hassan 等提出了若干变更传播启发式算法来研究变更在软件元素间（方法及属性）的传播，为开发人员进行软件维护提供指导。同时，他们提出了一种方法来研究各种变更传播模型的性能[174]。该研究主要研究一个软件元素变更发生时，还需要修改的其它软件元素，而 SoS-SN 方法主要研究软件整体结构的稳定性。

A. MacCormack 等利用方法间的调用关系构建源文件间的设计结构矩阵 DSMs（design structure matrices）来抽象软件源文件及其间的依赖关系，并提出变更费用（change cost）来度量软件元素对整个软件系统的平均影响[175]。一般而言，系统的变更费用越低越好。任何实施的变更其影响范围越小越好。该方法假设文件的变更会 100% 影响与这个变更类有直接或间接关系的其它文件，而在 SoS-SN 方法中，我们分析的粒度在特征粒度，同时分析的时候，我们假设变更是以一定的概率进行传播的。

I. P. Shaik 等在类粒度提出变更传播系数（change propagation coefficient）以评价软件结构的设计质量[176]。该方法主要在类粒度展开研究，而在 SoS-SN 方法中，我们分析的粒度在特征粒度。

Mirarab 等以贝叶斯网（Bayesian Network）为工具以一种概率的方式来研究变更的影响[177]。这种方法主要使用两种信息，即软件元素（包）静态依赖关系以及历史变更信息。该方法主要在包粒度展开研究，而在 SoS-SN 方法中，我们分析的粒度在特征粒度。

J. Liu 等提出平均传播率（average propagation ratio）来衡量软件结构的易变性[178]。所谓平均传播率，就是从任一类出发，顺着依赖关系的可达类集（包括变更类本身）占软件总类的期望值。平均传播率也可以用来度量软件结构的质量。平均传播率越小意味着修改和调试软件越容易。因此，软件的平均传播率应该控制的越小越好，只有这样才能以较小的代价来实现软件变更。该方法主要在类粒度展开研究，而且假设类的变更会 100% 影响与这个变更类有直接或间接关系的其它类，而在 SoS-SN 方法中，我们分析的粒度在特征粒度，我们假设变更是以一定的概率进行传播的。

L. Li 等在类粒度构建了面向对象软件的变更传播模型，提出了若干度量用于刻画变更的传播过程，并借此研究软件系统的稳定性[179, 180]。他们发现，降低传播概率、改善软件结构设计可以提高软件的稳定性。该研究和本章的工作非常相似。不同之处在于：在 L. Li 的方法中，他们在类粒度开展研究，假设每一次变更只有一个变更点（初始的变更软件元素）；而在 SoS-SN 方法中，我们分析的粒度在特征粒度，同时我们认为每一次变更，会有多个点（≥ 1）要求同时变更。SoS-SN 方法在模型的构建、稳定性度量方法上与 L. Li 等的工作存在较大的差异。

4.3 SoS-SN 方法

从第 4 章 4.2 节的分析我们可以发现，以前的研究工作有三个特点：（1）很多研究工作都假设，一个软件元素（文件、包、类、方法）的变更会以 100% 的概率影响与它们有直接和间接关系的其它软件元素；（2）在一次软件变更的会话中只有一个变更的软件元素；（3）软件变更分析常常在文件、包或类这三个粒度实施。我们认为，这三个特点在一定程度上与现实不符。以类粒度为例，理由如下：

（1）在软件系统中，一个类包含很多（≥ 1）个特征。如果有 ≥ 1 个属性或方法修改了，我们一般就认为这个类变更了。但是并不是任何与这个变更类有关联的其它类中的方法或属性都与这个类中变更的方法或属性发生直接或间接的关系。因此，并不是所

有与变更类发生关联的其它类都会以 100% 的概率受影响。

如图 4.1 所示，类 X 由三个方法（ $b()$、 $c()$、 $d()$）和一个属性（ a ）构成，类 Y 由两个方法（ $e()$、$f()$ ）构成。因此，类 X 的任何属性或方法变了，类 X 就认为变化了。在以前的研究中，人们往往认为类 X 的变更会 100% 影响类 Y，因为类 Y 依赖于类 X。然而，情况并非全如此。例如如果变更发生在方法 $c()$ 中，那么变更可能会传播至类 Y，因为 Y 中的 $f()$ 依赖于 X 中的 $c()$。然而，如果变更发生在 $c()$ 外的其它属性或方法，变更就不会传播至 Y，因为 Y 中的 $e()$、$f()$ 都没有直接或间接依赖于 X 中变更的属性或方法。因此，在做类粒度的变更影响分析时，我们必须引入概率。这种情况对于包粒度、文件粒度、方法粒度都适宜。

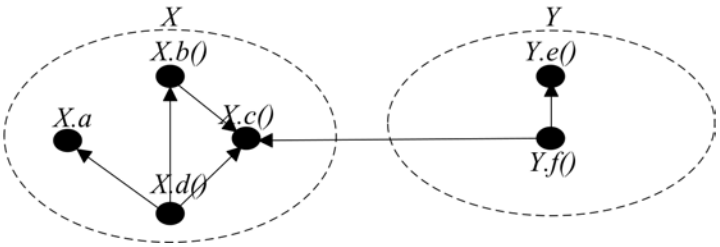


图 4.1　一个简单的例子

（2）软件在其生命周期内会不断的变更。软件的变更需求一般会使多个软件部分同时要求变更，尽管在某一个时间点只可以修改一个变更点，但是如果以一次变更会话来看，引发“涟漪效应”的“罪魁祸首”可能是多个变更点。也就是说，在一个变更会话中，初始的变更点（初始的变更软件元素）可能 ≥ 1。因此，现有的工作假设初始变更点只有一个不是太合理，这往往会高估变更的风险。

如图 4.2 所示，红色节点 $X1$ 和 $X2$ 是一次变更会话中的初始变更节点。在以前的研究中，人们假设每一次变更会话只有一个初始变更点，即在修改了 $X1$ 后再去修改 $X4, X5$ 和 $X6$；修改了 $X2$ 后再去修改 $X4, X5$ 和$X6$。所以若有 2 个初始变更点（红色部分）在一个会话中同时要求变更，那么虚线矩形框内的节点就会被算 2 次。所以，以前的研究会高估变更的风险，而且虚线框内的节点越多，这种高估就越严重。

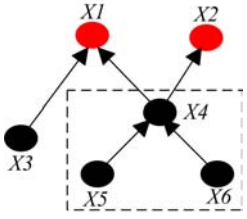


图 4.2　一个简单的例子

（3）复杂的面向对象软件系统往往由很多的包构成，包又包含了类，类又包含方法和属性，形成一种多粒度的层次结构。因此，对包、类的修改最终都可以转化成对方法和属性的修改。因此，为了提高分析的精准度，我们可以考虑将分析的粒度细化至特征粒度。

本章的工作是对现有工作的深入研究，特别可以看成是对 [178, 179, 180] 工作的细化和扩展。在本章中，我们尝试修正现有工作的不足，提出了一种更加精确的方法——SoS-SN 方法，具体的讲：（1）我们将变更影响分析的粒度细化至了特征粒度；（2）分析变更在特征之间传播的时候，引入了概率，也就是说，变更是以一定的概率从变更点不断往外传播的；（3）在将变更需求转化为初始变更点的时候，我们考虑了多点情况，即初始变更点集合元素个数 ≥ 1；（4）基于变更影响分析，我们构建了新的度量以评价软件稳定性，并提出求解该度量的相应算法，并将 *SoS* 应用于软件结构优选。

SoS-SN 方法的框架图如 4.3 所示。下面各小节中，我们将具体解释其中的主要部分。这里我们仍然使用面向对象软件作为我们的研究对象，理由已经在第 3 章 3.3.1 小节阐述，这里不再赘述。

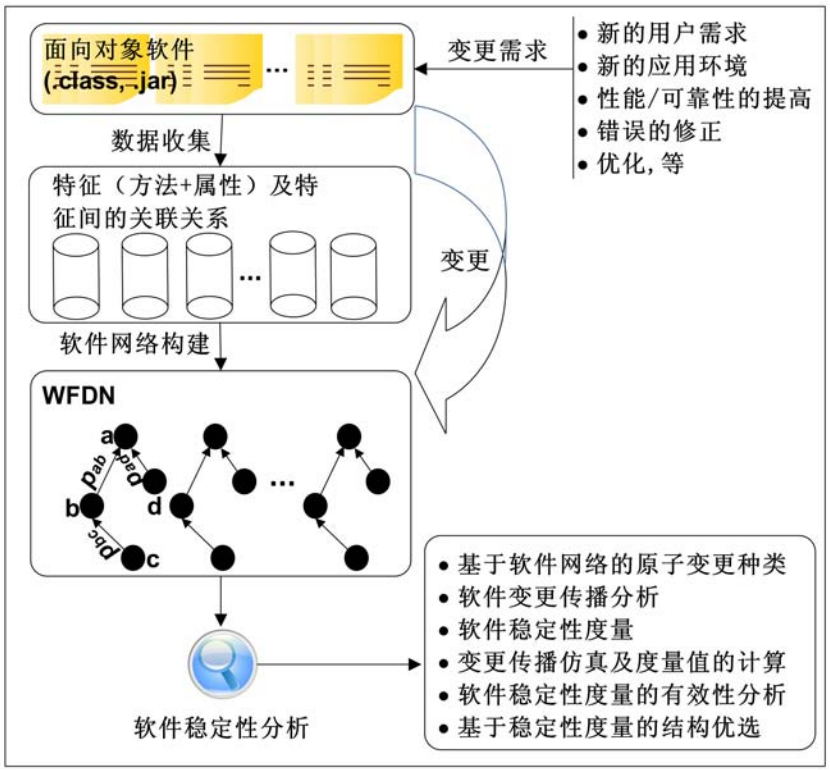


图 4.3　SoS-SN 方法框架

4.3.1 数据收集

SoS-SN 方法的第一步是数据收集，主要是对软件的特征及特征之间关系的收集。这里特征间的关联关系主要是两类，即：方法调用关系和方法使用属性的关系。本文所使用的数据也是由我们自主开发的软件网络分析工具 SNAT 自动提取的。据我们所知，目前还没有开源的工具可以提取方法和属性间的关系。为了提高 SoS-SN 方法的可重复性，我们已将本章中用到的部分工具已公开，可以从 [164] 处下载。

4.3.2 软件稳定性分析

本节将具体地介绍 SoS-SN 方法，主要关注特征粒度软件网络的定义，软件网络变更的种类，软件稳定性度量的定义及计算方法，以及度量的收敛性分析。

4.3.2.1 软件网络

根据 4.3.1 节中收集的数据，我们可以定义特征粒度的软件网络——加权特征依赖网络。在本章中，我们同等看待特征间的关系（方法调用关系和方法访问属性的关系），都看成“使用”关系。下面首先给出加权特征依赖网络的定义：

定义1：加权特征依赖网络（Weighted Feature Dependency Network, WFDN）

WFDN 是一个加权有向图。在 WFDN 中，节点代表一个特定面向对象软件系统中的所有特征，每一个特征用唯一一个节点表示；有向边表示特征之间的使用关系，即：如果特征 A 使用特征 B，那么在 WFDN 中就有一条有向边从代表特征 A 的节点指向代表特征 B 的节点。这里我们只考虑使用关系的存在，忽略使用关系的出现次数，也就是说，如果 A 使用 B 三次，在 WFDN 中代表这两个特征的节点之间也只有一条有向边。有向边上的权值表示变更在这两个节点之间传播的概率，即：如果 WFDN 中存在一条边 $N_A \to N_B$，它边上的权值是 0.6，那么就表示节点 N_B 如果变了，节点 N_A 有 0.6 的概率也有可能要做变更。在 SoS-SN 方法中，我们认为初始的变更节点可能有多个，在这里我们用初始变更点占总节点数的比例作为参数。因此，WFDN 可以定义为：

$$\text{WFDN} = (N, E, \boldsymbol{M_p}, CR), \tag{4.1}$$

其中：N 是 WFDN 的节点集合，代表特定面向对象软件系统所有特征；E 是 WFDN 边的集合，代表特征间的使用关系；$\boldsymbol{M_p}$ 是一个矩阵，存储了节点间变更传播的概率。如果节点 j 连接至节点 i（$j \to i$），那么 $\boldsymbol{M_p}$ 的元素 $\boldsymbol{M_p}(i,j)$ 存储的概率表示如果节点 i 变更了，那么变更将以 $\boldsymbol{M_p}(i,j)$ 的概率传播至节点 j。如果两节点间 k

和 l 间没有边相连，那么 $\boldsymbol{M_p}(k,l)$ 和 $\boldsymbol{M_p}(l,k)$ 就赋值为 0。CR 是一个比例值，表示初始变更节点占节点总数的比例。图 4.4 所示是一个代码片段及其相应的 WFDN。

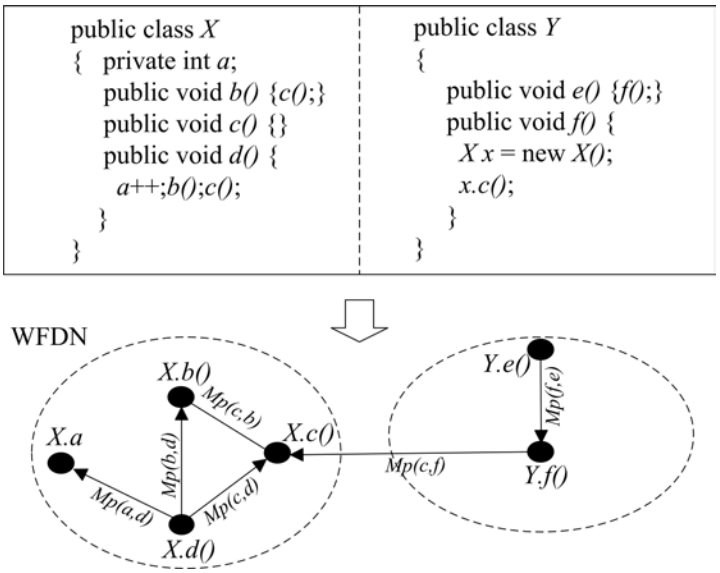


图 4.4　代码片段及其相应的 WFDN

在 WFDN 中，我们假设任意相连两节点间的权值相同，即 $\boldsymbol{M_p}$ 非零元素的值相同。

4.3.2.2　变更类型

使用软件网络进行软件变更影响分析，我们将把对软件代码做的修改转化为一串“原子变更”（即本章中变更的最小粒度），并映射为对网络结构的操作。4.2 节中也不乏一些从结构角度开展软件变更影响分析的工作，但是它们没有指出其方法所能分析的软件变更的种类。也就是说，它们没有说明其提出的方法能够分析哪些变更类型，哪些不能分析。在下文中，我们参考 [181]，针对 WFDN 提出了一串面向对象软件的“原子变更”。在分析的过程中，我们假设最初的和修改后的软件系统都是语法上正确的，并且任何对软件源代码的修改都可以转化为“原子变更”的集合实现。本章针对 WFDN 提出的原子变更类型见表 4.1。

此外，我们主要针对软件系统现有的软件元素及元素间的关系，从结构角度分析软件的稳定性。因此，那些引入新节点和新关联关系的原子变更（如表 4.1 中的 CMD、AF 和 CFD）都被 SoS-SN 方法所忽略。这一不足也是目前软件变更分析和稳定性分析中普遍存在的一个问题。所以从这个意义上来讲，我们的方法并不比其它方法优越。由于 SoS-SN 方法从结构角度研究变更传播并度量软件稳定性，因此我们忽略了变更的语

表 4.1　变更种类

简称	原子变更名	是否实现
DEM	删除一个空的方法	是
DM	删除一个非空的方法	是
CHM	改变一个方法的头	是
CM	改变方法体，但不引入新的关联	是
CMD	改变方法体，并引入新的关联	否
AF	增加新的属性	否
DF	删除一个属性	是
CF	改变一个属性，但不引入新的关联	是
CFD	改变一个属性，并引入新的关联	否

义信息，将所有的变更一视同仁，都看成“变更”，并通过一种仿真的方式来模拟初始变更的发生，即随机选择节点作为初始变更节点集（详见算法 4.1）。

4.3.2.3　变更传播算法

软件在其生命周期内会不断经历变更，以适应新需求、性能提高、错误修正和优化等变更需求。由于软件维护活动的多样性和随机性，预测下一次维护活动在何时实施、实施的内容是什么是毫无意义的[167]。鉴于软件维护活动的随机性，在 SoS-SN 方法中，我们使用随机选择初始变更点的方式来模拟这一过程。在下面的小节中，我们会提出一个算法，通过一种仿真的方式来进行软件变更影响分析。在这之前，我们首先给出这个算法中用到的一些度量的定义。

定义2：节点变更的概率传播域（Change Probability Propagation Field of Node, $CPPFN$）

假设存在一个特定的 WFDN。那么节点 i 在这个 WFDN 中的变更的概率传播域 $CPPFN(i)$ 定义为在一次特定的算法运行中实际受到节点 i 变更影响的节点集合。同时，$sCPPFN = |CPPFN(i)|$ 表示这个集合中元素的个数。其中：$|*|$ 表示集合 $*$ 中元素的个数。

定义3：节点集变更的概率传播域（Change Probability Propagation Field of a Set of Nodes, $CPPFSN$）

假设存在一个特定的 WFDN。那么节点集 $setN$ 在这个 WFDN中的变更的概率传播域 $CPPFSN(setN)$ 定义为在一次特定的算法运行中实际受到 $setN$ 中节点变更影响的节点集合。同时，$sCPPFSN = |CPPFSN(setN)|$ 表示这个集合中元素的个数。$CPPFSN(setN)$ 的计算如下：

$$CPPFSN(setN) = \bigcup_{\forall i \in setN} CPPFN(i) \tag{4.2}$$

在算法 4.1 中：mp 和 cr 是两个 0 和 1 之间的实数；$maxT$ 是一个远大于 $|N|$ 的整数；$simT$ 是仿真算法运行的次数；$bChanged[]$ 是一个布尔型数组，其中每一个元素存储的是 WFDN 中每个节点的状态，“true”表示该节点变更了，“false”表示相应节点未做变更；$bChangedBak[]$ 是 $bChanged[]$ 的备份数组；$bSelected[]$ 也是一个布尔型数组，存储相应节点是否是初始变更节点，即：“true”表示相应节点是初始变更节点，“false”表示相应节点不是初始变更节点；$CPPFSN[]$ 是一个整型数组，存储算法每一次运行后的 $CPPFSN$；$cChgNN$ 表示初始变更节点的个数。算法 4.1 可以在 $O(N^2)$ 时间内计算得到节点集变更的概率传播域。

4.3.2.4 软件稳定性度量

基于上述分析，本小节将定义一些度量来刻画面向对象软件的稳定性。

定义4：变更节点率（Changed Node Ratio, CNR）

在一个具体的 WFDN 中，变更节点率 CNR 定义为算法 $simT$ 次运行中，平均受影响节数占 WFDN 总节点数的比例，并可以通过下式计算：

$$CNR = \frac{\sum_{i=1}^{simT} CPPFSN[i]}{simT \times |N|} \times 100\% \tag{4.3}$$

上式中的符号具有与算法中相同的含义。

定义5：软件稳定性（Stability of Software, SoS）

我们将软件的稳定性 SoS 指标定义为：算法 $simT$ 次运行中，未受影响的节点占 WFDN 节点总数的比例，即：

$$SoS = 1 - CNR \tag{4.4}$$

显然，SoS 是一个在 0 和 1 之间取值的标量。SoS 越大，意味着软件越稳定，变更越不容易在这样的软件结构中进行扩散，软件质量越高。

算法 4.1: 变更传播算法

输入:

WFDN, mp, cr 和 $maxT$;

输出:

$CPPFSN[i]$ (i = 1, 2, ..., $|N|$);

1: 初始化 $\boldsymbol{M_p}$: 如果从节点 i 到节点 j 之间存在一条有向边, 则令 $\boldsymbol{M_p}[i][j] = mp$, 否则为 0. 令 $CR = cr$, $simT = maxT$. 令 $bChanged[i] = false$, $bChangedBak[i] = false$, $bSelected[i] = false$, $CPPFSN[i] = 0$ ($i = 1, 2, ..., |N|$). 令 $t = 1$, $cChgNN = 0$. 准备一个队列 cQueue.

2: 如果 $cChgNN \leq cr \times |N|$, 则从 N 随机选择一个满足 $bChangedBak[i] = false$ 和 $bSelected[i] = false$ 的节点 i。并将该节点压入队列 cQueue, 并设置 $bChanged[i] = true$, $bChangedBak[i] = true$, $bSelected[i] = true$, $cChgNN$++, 然后转第 3 步执行; 否则转第 6 步执行.

3: 如果 cQueue 是空的, 则转第 6 步执行; 否则转第 4 步执行.

4: 从 cQueue 队首取出一个节点, 令其为 N_i. 一个个遍历 N 中的节点(令遍历到的那个节点为 N_j). 如果 $Mp[i][j] = mp$, 则将节点 j 存入临时集合 $tempSet$.

5: 对于 $tempSet$ 中的每个节点 N_k, 随机生成一个大于 0 的实数 dec_k. 如果 $dec_k \geq mp$, 则: (1) 将节点 N_k 存入 cQueue (表示 N_i 的变更会传播至 N_k); (2) 将该节点从 $tempSet$ 删除; (3) 将 N_k 相应的 $bChanged[k]$ 和 $bChangedBak[k]$ 设置为 true; (4) 转第 2 步执行; 否则将它从 $tempSet$ 删除, 并转第 2 步执行.

6: 令 $CPPFSN[t]$ 为 $bChangedBak[l]$ (l = 1, 2, ..., $|N|$)中的非零元素个数, $t = t + 1$. 如果 $t < simT$, 则令 $bChanged[i] = false$, $bChangedBak[i] = false$ (i = 1, 2, ..., $|N|$), 并转第 2 步执行; 否则转第 7 步执行.

7: **返回:** $CPPFSN[i]$ (i = 1, 2, ..., $|N|$).

4.3.2.5 *SoS* 收敛性分析

如前所述，*SoS* 是通过仿真算法来求值的。所以，在两次独立的运行中，即使 mp、cr 等参数设置成相同的值，所得到的 *SoS* 也可能不同。因此如果要使用 *SoS* 来度量软件的稳定性，我们必须验证 *SoS* 是否能够收敛于一个相对的稳定值。

在运行变更传播算法 4.1 前，我们必须设置两个主要的参数 mp 和 cr。所以，下文我们将在 $maxT$ 取相同值的情况下，检验不同 mp、cr 设置下，*SoS* 是否可以收敛。这

里我们使用 JUnit 3.4[182] 作为例子来进行说明。表 4.2 是 JUnit 3.4 的一些统计数据，包括整个系统包的数量、类的数量以及特征的数量。在实验的具体操作中，我们忽略了 JUnit 系统中那些孤立的特征（即那些不与任何其它特征发生交互的特征）。因此，本章讨论的 WFDN 主要由规模（节点数）大于 1 的弱连通图构成。表 4.2 同样显示了 JUnit 3.4 相应 WFDN 的节点数（$|N|$）和边数（$|E|$）。

表 4.2　JUnit 3.4 统计数据

参数	值	参数	值
包数	7	类数	78
特征数	601	$\|N\|$	575
$\|E\|$	889		

根据我们的经验，一般初始变更节点数往往在 6 个以内。因此，这里我们以 1/575 为间隔，将 cr 从 1/575 设置到 6/575，$maxT$ 取值 50,000。对于每一种 cr 的取值，我们以 0.1 为间隔，将 mp 从 0.1 设置到 1，然后分析 SoS 随算法运行步 t 的变化趋势。限于篇幅，这里我们仅显示了 cr 在两种设置（即 $cr = 1/575$ 和 6/575）下的结果（如图 4.5）。从 SoS vs. t 的曲线我们可以发现：(1) 在算法运行的早期（特别是 $t <$ 5,000），SoS 并不稳定，变化比较剧烈。这说明算法运行中选择了不同的初始变更节点集，导致 SoS 波动比较明显。(2) 当 t 远大于 $|N|$（如 $t \geq$ 50,000），SoS 收敛于一个相对稳定的值。这说明当算法运行次数 t 远大于 $|N|$时，SoS 具有比较好的收敛性。对 JUnit 3.4 进行多次试验后我们发现，SoS 在多次实验中基本保持稳定，标准差接近于零。(3) 在相同的 cr 和 t 设置下，cr 越大，SoS 收敛到的值越小，说明软件的稳定性越差。在 mp 和 cr 的其它设置下，我们也获得了类似的结论。

但是，从 JUnit 3.4 获得的结论是否具有普遍性呢？为了回答这个问题，我们随机的选择了不同领域，不同规模的 100 多个软件系统，分析了这些系统 SoS 的收敛性。我们同样也获得了类似的结论。限于篇幅，我们没有说明这 100 个系统的细节和实验结果。基于上述分析，我们可以得出结论：SoS 可以用于度量软件稳定性。

4.4　SoS 的理论评价

软件稳定性度量 SoS 也属于软件复杂性度量的范畴（如图 2.2）。E. J. Weyuker 曾

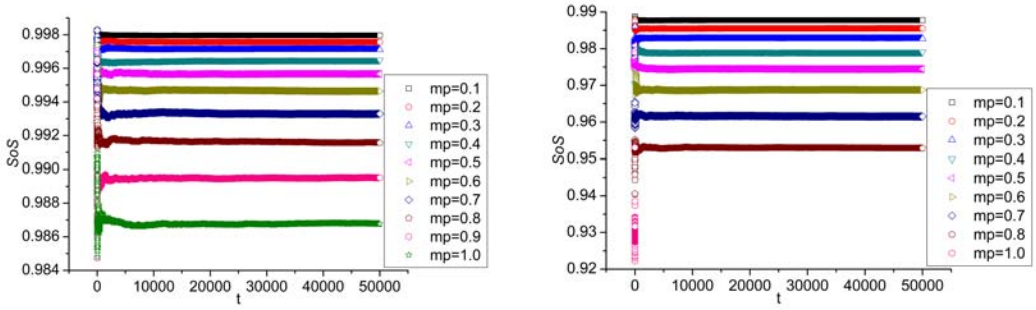


图 4.5　SoS vs. t 的曲线。其中：左图对应 $cr = 1/575$，$maxT = 50{,}000$ 的设置；右图对应 $cr = 6/575$，$maxT = 50{,}000$ 的设置

提出了一组准则用于评价软件复杂性度量的有效性[183]。在这些准则出现之后，很多人对它提出了批评（特别是**准则 9**）。N. E. Fenton 认为 Weyuker 准则不能提供对复杂性一个单一确定的认识[15]。H. Zuse 认为 Weyuker 准则与尺度理论矛盾[9]。S. R. Chidamber 等认为 Weyuker 准则是一个度量成为好的度量的必要条件，但不是充分条件[184]。尽管 Weyuker 准则还存在不足，但是它提供了一种形式化的方法来评价软件度量的有效性，因而被广泛使用[184, 185, 186]。本章中，我们也将使用 Weyuker 准则对提出的稳定性指标 SoS 进行评价。S. R. Chidamber 等指出在用 Weyuker 准则评价复杂性度量时，必须对准则进行筛选[184]。本章中，我们忽略了准则 5、7 和 9。在下面的分析中，为了完整性，我们也给出了准则 5、7 和 9，但并不用这几条准则来评价 SoS，只是简单的给出了不选这几个准则的理由。下文中 M 代表一个软件复杂性度量。

准则 1：一个复杂性度量不能将所有的程序都视为相同复杂性的，即：对于给定的两个程序 P 和 Q，总能找到 $M(P) \neq M(Q)$。表 4.4 是 8 个程序在各种不同 mp 设置下的 SoS 值，从这 80 个不同的 SoS 值我们可以看出，SoS 并没有将所有的程序看成具有相同稳定性的。因此，SoS 满足准则 1。

准则 2：仅仅只有有限个程序，它们具有某个特定的复杂性，即：如果 c 是一个非负的实数，那么仅有有限个程序 P，它们符合 $M(P) = c$。我们讨论问题的论域往往是有限个应用的集合，显然，在这个论域中只有有限个程序具有相同的 WFDN 和 SoS。因此，SoS 满足准则 2。

准则 3：存在两个不同的程序具有相同的复杂性，即：存在两个不同的程序 P 和 Q，符合 $M(P) = M(Q)$。显然我们可以比较容易的构造两个具有不同功能，但是具有相同 WFDN 的程序。因此，SoS 满足准则 3。

准则 4：具有相同功能的两个程序，不一定具有相同的复杂性，即：存在两个具有相同功能的程序 P 和 Q，它们符合 $M(P) \neq M(Q)$。对于 4.5 小节使用的数据，它们

每个都具有相同的功能，但是它们的 WFDN 和 SoS 都不尽相同。表 4.3 的数据说明 SoS 满足准则 4。

准则 5：程序片段的复杂性不应该大于整个程序的复杂性，即：对于任何两个程序 P 和 Q，必须满足 $M(P) \leq M(P+Q)$ 和 $M(Q) \leq M(P+Q)$。因为这条准则只适合那些度量规模的度量指标，而本章提出的 SoS 用于度量软件稳定性（非规模），所以它不适合评价 SoS。

准则 6：由程序 P 和 Q 合并而得到的的程序的复杂性不一定等于 Q 和 R 合并而得到的程序的复杂性，即使 P 和 R 的复杂性相同，即：$\exists P$、$\exists Q$、$\exists R$，若 $M(P) = M(R)$，并不一定满足 $M(P+Q) = M(R+Q)$。这一准则指出 P 与 Q 的交互和 Q 与 R 的交互可能得到不同的结果，从而导致复杂性不一样。如果假设 (N^P, E^P, M_p^P, CR^P)、(N^Q, E^Q, M_p^Q, CR^Q) 和 (N^Q, E^Q, M_p^Q, CR^Q) 分别是程序 P、Q 和 R 的 WFDN，因此，合并 P （Q）和 R 而得到一个单独的程序$(P+R)$ （$(Q+R)$），它的 WFDN 是 $(N^{P+R}, E^{P+R}, M_p^{P+R}, CR_p^{P+R})$ （$(N^{Q+R}, E^{Q+R}, M_p^{Q+R}, CR_p^{Q+R})$）。显然，$N^{P+R}$ 可能不同于 N^{Q+R}。例如：如果 P、Q和 R 的节点集分别是 $N^P = \{a, b, c\}$、$N^Q = \{d, e, f\}$ 和 $N^R = \{b, c, g\}$，那么合并 P 和 R（Q 和 R），我们得到 $N^{P+R} = \{a, b, c, g\}$ （$N^{Q+R} = \{b, c, d, e, f, g\}$）。因为 $|N^{P+R}| \neq |N^{Q+R}|$，所以我们有理由相信程序 $(P+R)$ 和 $(Q+R)$ 的 SoS 不相同。因此， SoS 满足准则 6。

准则 7：如果我们对程序中的语句进行排序，那么排序后的程序的复杂性不一定等于原始程序的复杂性，即：对于两个程序 P 和 Q（Q 由 P 中语句排序后得到），可能满足 $M(P) \neq M(Q)$。这条准则是针对面向过程编程语言提出来的，而在面向对象编程中，类是对问题域的抽象，对类内语句顺序的调整并不会影响该类的使用，所以它不适合评价 SoS。

准则 8：重命名方法和属性对复杂性度量没有影响，即：如果程序 P 由程序 Q 重命名得到，那么 $M(P) = M(Q)$。因为 SoS 的构造与方法和属性等的命名无关。因此，SoS 满足准则 8。

准则 9：两个程序合并后得到的程序的复杂性可能大于两个程序复杂性之和，即：$\exists P$ 和 $\exists Q$，满足 $M(P) + M(Q) < M(P+Q)$。因为 SoS 是一个序级测量（ordinal measure），简单的将两个值相加没什么具体的意义，同时这个准则受到很多学者的质疑，所以它不适合评价 SoS。

综上，我们可以发现，SoS 满足大部分的 Weyuker 准则，仅 5、7 和 9 不满足。准则 5 指出复杂性应该单调增长。SoS 不满足这条准则，因为 SoS 是根据软件拓扑结构

计算的，而不是规模。准则 5 只适合对规模的度量，这一结论也在很多其它的研究中发现了。准则 7 也不适合评价 SoS，因为该准则最初是针对传统编程语言提出来的。准则 9 显示，程序间的交互会增加复杂性。这条准则受到很多学者的质疑。准则 9 不适合评价 SoS，因为 SoS 是一个序级测量，简单的相加没什么含义。

4.5 SoS 的实验评价——结构优选

设计模式（design pattern）是对软件设计中普遍存在、反复出现的设计问题的通用解决方案，是代码设计经验的总结。设计模式被普遍认为是一种可以提高软件质量的方式[187]。这种软件质量的提高应该被 SoS 区分。也就是说，SoS 应该有能力实现结构优选，即从具有相同功能不同结构的软件系统中选择结构较优、质量较高的软件。本节我们通过数据实验的方式检验 SoS 在结构优选方面的能力，并对 SoS 的有效性进行数据实验分析。

4.5.1 数据源

我们将用八组 Java 语言编写的程序来检验 SoS 的有效性。这八组程序，每组都有两个软件版本，这两个软件版本功能相同，唯一的区别是一个采用设计模式，一个不采用设计模式。之所以选择这八组程序作为研究对象，主要因为它们满足：(1) 都用 Java 语言编写，可以被我们的分析工具解析；(2) 两个软件版本具有相同的功能，唯一的区别是一个使用了设计模式，一个没有使用设计模式，在质量上我们可以明显的区分这两个软件，即认为使用设计模式的软件质量好，稳定性高。在本章我们使用的设计模式包括：Adapter、Bridge、Builder、Chain、Composite、Interpreter、Iterator 和 Sate[188]。这八组程序的源代码和字节码都可以从 [164] 下载。表 4.3 显示的是这八组 Java 程序的一些简单统计数据，表中所有标识的含义与前文第一次提到处相同。

4.5.2 实验结果

为了分析面向对象软件的稳定性，我们首先将待分析的系统用 WFDN 表示。限于篇幅，这里仅给出了使用 Adapter 设计模式前和使用 Adapter 设计模式后程序的 WFDN （如图 4.6）。在得到了系统的 WFDN 之后，我们就可以使用算法 4.1 求其 SoS。表 4.4 显示的是在 $maxT = 50,000$，$cr \times |N| = 1$ 和 $cr = 0.2\alpha$ (α= 1, 2, 3, 4, 5) 设置下求得的各程序在使用设计模式前后的 SoS 值。

表 4.3　八组 Java 程序的统计数据

	WFDN		WFDN	
	Before	Before	After	After
Design Pattern	$\|N\|$	$\|E\|$	$\|N\|$	$\|E\|$
Adapter	5	4	12	11
Bridge	26	48	41	65
Builder	11	13	31	34
Chain	7	8	10	13
Composite	11	15	12	15
Interpreter	4	3	20	25
Iterator	5	6	15	21
Sate	5	5	15	16

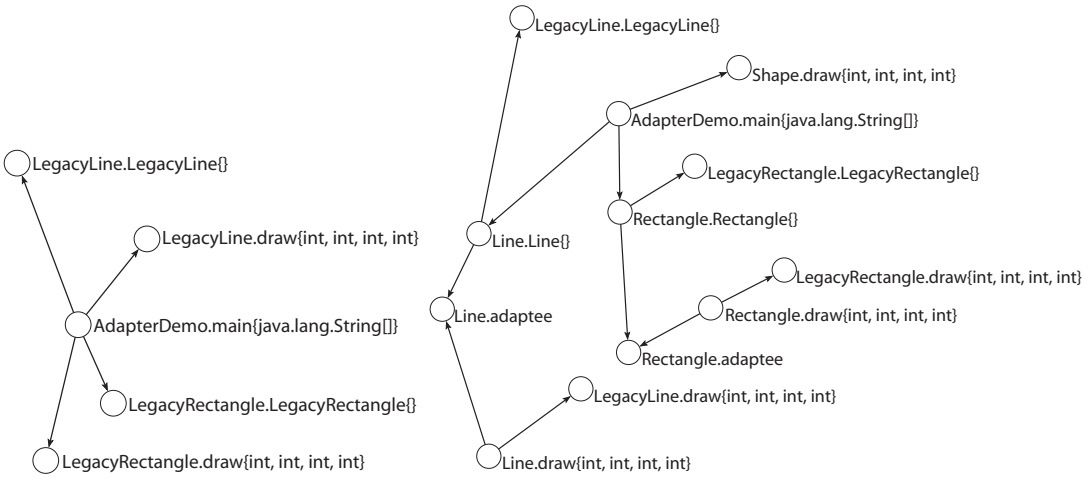


图 4.6　使用 Adapter 设计模式前（左）/后（右）Java 程序的 WFDN

从表 4.4 可见：(1) 使用设计模式后的程序其 *SoS* 都比相应的没有使用设计模式的程序的 *SoS* 要高，这个结果与我们预期的符合，即设计模式可以提高软件系统的质量。 *SoS* 可以实现结构优选。(2) 对于同一个程序的同一个版本，随着 mp 的增大，*SoS* 的值不断下降，这说明随着节点间概率的增大（软件耦合的增强），软件的稳定性不断下降。

在其它 cr 和 mp 设置下，我们得到了类似的结果，限于篇幅，这里将它们省略了，更多的数据和结果请见 [164]。

表 4.4　八组 Java 程序的 SoS（$maxT = 50,000$, $cr \times |N| = 1$, $mp = 0.2\alpha$ $(\alpha = 1, 2, 3, 4, 5)$）

Design Pattern	mp=0.2		mp=0.4		mp=0.6		mp=0.8		mp=1.0	
	Before	After	Before	After	Before	After	Before	After	Before	After
Adapter	0.76726	0.900075	0.73698	0.881	0.7031	0.860717	0.67144	0.83775	0.63974	0.812883
Bridge	0.944435	0.967037	0.924104	0.95579	0.901631	0.943246	0.879369	0.929088	0.858127	0.913768
Builder	0.886064	0.960255	0.860818	0.953065	0.829336	0.944642	0.801873	0.936771	0.776091	0.927984
Chain	0.821386	0.87126	0.778914	0.8382	0.733714	0.80083	0.683729	0.76639	0.631786	0.7293
Composite	0.882482	0.892342	0.847091	0.866942	0.808427	0.837133	0.770873	0.807825	0.734336	0.777725
Interpreter	0.7102	0.936635	0.663675	0.9216	0.613675	0.902025	0.5598	0.8816	0.49895	0.85966
Iterator	0.7471	0.911807	0.68474	0.88482	0.628	0.853267	0.56862	0.822547	0.51544	0.792087
Sate	0.75778	0.91716	0.70876	0.90002	0.65586	0.878807	0.60412	0.8554	0.5623	0.83218

4.6 本章小结

软件稳定性描述软件对变更放大作用的抵抗能力，是软件设计的核心，对软件维护和质量的评估都有重要影响。但是，在软件的整个生命周期中，软件变更的需求存在不确定性（变更的类型、产生时间和粒度不确定），使得如何认识、理解、分析和控制软件稳定性成为软件工程的一个难题。

本章中，我们提出了软件的加权特征依赖网络模型 WFDN，并用于抽象软件在特征粒度的内在拓扑结构，并使用一种仿真的方法分析变更在 WFDN 上的传播过程，最终提出一个新的度量 *SoS* 用于评价软件的稳定性。最后，使用 *SoS* 从功能相同结构不同的软件中选择结构好、质量高的软件系统。我们认为，一个具有高稳定性的软件，应该将变更控制在一个尽可能小的范围内（变更影响的其它软件元素的数量越少越好），也就是说，*SoS* 越大越好。我们使用 Weyuker 准则对 *SoS* 的有效性进行理论分析，发现 *SoS* 基本符合 Weyuker 准则。同时，对八组 Java 程序的仿真实验也表明 *SoS* 可以正确的选择结构质量高的软件。同时，在实验的过程中我们也发现：随着软件元素间的耦合增强，软件的稳定性不断下降。因此，在实际的软件开发中，我们应该尽量降低元素间的耦合，以提高软件质量。

SoS-SN 方法从模型的合理性上改进了现有的工作，但是还存在下列问题：（1）在 WFDN 边权的设置上，我们简单的设成了同值，这显然不太符合实际；（2）我们的工作是针对 Java 程序的，那么对其它面向对象语言（C++等）也适用吗？所以，在未来的工作包括：（1）结合特征变更的可能性大小来合理的设置 WFDN 的边权；（2）在更大范围的软件系统上验证 *SoS* 方法的有效性。

本章实验部分所有数据，包括实验的八组 Java 程序、所有的 WFDN、仿真算法 4.1 程序、实验结果数据等都可以从 [164] 处下载。

第五章　面向对象软件类重构

"*If you get into the hygienic habit of refactoring continuously, you'll find that it is easier to extend and maintain code.*[189]."

—— *J. Kerievsky*

"*Refactoring and adding new functionality are two different but complementary tasks.*"

—— *A. Scott*

5.1　引言

软件质量和软件的寿命在很大程度上受到软件内部结构的影响。然而，软件的最初结构往往不能适应新的需求，错误的修正、性能的提高、软件运行环境的变更等因素都促使软件作出必要的调整。从第 4 章我们已经知道，软件的易变性已成为软件的一个重要特征。同时，激烈的市场竞争，又要求软件开发商快速的响应各种变更，向用户提交高质量的软件产品。因此，在时间、资源等各种约束下，软件维护者对软件的变更往往是在没有充分考虑的情况下作出的，缺少对软件整体结构的合理认识[190]。这种不合理的变更，给软件带来了非常隐蔽的副作用，使软件系统变的越来越复杂，离其最初的设计越来越远。软件维护极大的降低了软件的质量[191]，形成了所谓的"软件腐朽"问题[192]，使得软件维护的费用不断上升[166]。

因此，软件系统在其生命周期内必须时不时的作出调整。软件重构（refactoring）被认为是一种提高软件结构设计和可维护性的有效方式，它可以在软件生命周期的任何一个时段使用[192]。然而，我们必须首先识别出那些对软件质量和软件维护等有负面影响的软件元素，即所谓的 "Bad Smell"，然后才能通过特定的重构策略将这些 "Bad Smell" 移除[190, 193]。但是，选择使用何种重构方式，在哪里实施重构仍然是一个非常困难的问题，因为重构会对软件系统的结构产生不可预知的副作用[190]。

当用属性方法网络和方法耦合网络[1]可视化软件内部结构的时候，我们发现：（1）属于同一个类的方法/属性之间彼此联系比较紧密，属于不同类的方法/属性之间交互比较少。也就是说，网络中存在一些小集团（cliques），集团内的软件元素连接稠密，集团间的软件元素连接稀疏。这种结构与复杂网络研究中的社区结构非常类似。同时，这种

[1]这两个网络的定义请见 5.3.4 小节。

结构上的组织在一定程度上也符合软件工程“高内聚，低耦合”的设计原则。但是，我们也发现存在一些例外的方法/属性，它们不符合这种设计原则。也就是说，它们与同类的方法/属性联系少，与其它类的方法/属性联系紧密。这些方法/属性增加了类之间的耦合，增加了维护的费用，降低了软件的质量。因此，如果可以识别出这些方法/属性，然后把它们移动到更合适的位置，必定可以减少耦合、提高内聚，那么软件的质量就可以在一定程度上获得提高。软件系统由类构成，类又包含了很多的方法和属性，类是方法和属性的自然组织，形成小集团，这种组织结构与复杂网络研究中的社区结构如此相似，这启发我们可以使用复杂网络中的社区发现算法来优化软件的类结构。

鉴于此，本章在第 3 章 3.4.2.5 小节工作的基础上，提出了一种基于软件网络社区发现的面向对象软件类重构的 CR-CDSN 方法（class structure refactoring of object-oriented softwares using community detection in software networks），以帮助软件开发者在编码过程中实施软件重构工作。首先，CR-CDSN 方法将面向对象软件系统在特征（方法和属性）粒度抽象成两类软件网络，即：属性方法网络和方法耦合网络；然后，借鉴复杂网络中的社区发现算法，并针对软件重构的实际需要，提出了一种有导向的社区发现算法 GCDA （guided community detection algorithm）来优化软件的类结构；最终，通过对比优化前后的软件结构，以及对属性方法网络的节点检查，我们为软件开发者提供了一个可能需要移动的方法和属性的列表。

本章后续内容的组织结构如下：5.2 节从软件重构和社区发现算法两个方面介绍相关研究内容；5.3 节介绍 CR-CDSN 方法，包括：数据收集、数据的预处理、相关软件网络定义、有导向的社区发现算法 GCDA 等；5.4 节介绍针对开源 Java 软件的实验设计方法，包括：实验系统介绍、具体的实验过程和结果分析；最后在 5.5 节对本章进行小结。

5.2 相关研究工作

本章的工作主要涉及三大块内容，即软件重构、复杂网络社区发现研究以及软件网络研究。软件网络的研究已在第 2 章 2.3 节作过详细介绍，这里不再赘述。因此，本小节将分两块内容（即软件重构、社区发现算法）来对与本章相关的研究作概述。

5.2.1 软件重构

关于软件重构方面的研究工作很多，T. Mens 和 T. Tourwé 曾对该领域的工作作过

综述，详见 [191]。本节只对那些涉及软件结构的工作作介绍。B. Kang 和 J. M. Bieman 提出了一个软件重构的量化框架[194]。他们的框架主要由 3 个部分构成，即：设计的模型、基于度量的软件重构标准和重构过程。该方法使用图来抽象软件的模块以及模块间的联系，使用内聚和耦合作为度量指标，然后基于这些度量值来决定软件中哪些模块需要重构。T. Dudzikan 和 J. Wlodka 将各种方法整合起来，用于重构软件[195]。用户只要提供一些“Bad Smell”，该方法就可以为用户提供一些解决方案。O. Seng 等提出了一种基于搜索的软件类结构优化方法[190]。该方法仅使用一些简单的度量值（设计准则违反数等），然后利用演化算法，就可以为软件工程实践者识别软件中那些可以提高软件质量的部分。但是该方法的时间耗费严重，即使是一个很小规模的软件，使用该方法都有可能耗费几个小时甚至几天。I. Ivkovic 和 K. Kontogiannis 使用模型转换（model transformations）和质量改善语意标注（quality improvement semantic annotation）技术，提出了一种优化软件结构的新方法[196]。该方法首先用 UML 图表示软件的概念结构；然后用度量、约束等对结构模型进行标注；最后，在具体重构时为用户推荐动作（action）。I. G. Czibula 和 G. Serban 使用聚类技术提出 CARD 方法用于改善软件的结构[197]。CARD 方法使用基于 k-means 聚类算法来获得优化的软件结构，最终为开发者提供一个疑是重构点的列表。

5.2.2 社区发现算法

社区结构是复杂网络中又一个重要的特性，它是一种介于宏观和微观之间的网络特征，是真实世界中许多复杂网络所具有的一种普遍性质。研究表明，社区结构和网络的功能有着紧密的关系，像鲁棒性、高速传播性等。网络社区结构的检测是揭示网络结构和功能之间关系的重要基础，因此成为了最近几年的研究热点。复杂网络采用图作为表示形式，因此很多在计算机领域被广泛应用的图分割算法都可以直接应用于复杂网络的社区检测，如 Kernighan-Lin 算法[198]，谱分割法[199, 200]等。但同时又由于传统图分割算法倾向于把网络分成规模相近的子图或要求分割数目已知，而社区发现中网络的社区数目以及每个社区的节点规模通常是不可预知的，因此在复杂网络领域也出现了很多新的社区发现方法，按照边的连接权值可以分为：无权网络的社区发现方法[201, 202]和加权网络的社区发现方法[203, 204]；按照节点的隶属度可以分为：硬划分的社区发现方法[202, 203, 204]和模糊社区发现方法[205, 206]；按照发现的策略可以分为：层次算法[202, 207]（分裂和凝聚）和最优化算法[208]；或者分为全局社团发现方法[202, 203, 204, 207, 208]和局部社区发现方法[209, 210]等。

5.3 CR-CDSN 方法

尽管本章的工作并不是第一个关于软件重构方面的研究，但是我们会从一个全新的角度来入手。为了识别软件中的重构点，我们首先将面向对象软件系统在特征粒度抽象成两类软件网络——属性方法网络和方法耦合网络。然后提出了一种有导向的社区发现算法 GCDA 来优化软件的结构；最终，通过对比优化前后的软件结构，以及对属性方法网络的节点检查，我们为软件开发者提供了一个疑是需要移动的方法的列表。图 5.1 显示的是 CR-CDSN 方法的框架。下面各小节将具体解释其中的主要部分。这里我们仍然使用面向对象软件作为我们的研究对象，理由已经在第 3 章 3.3.1 小节阐述，这里不再赘述。

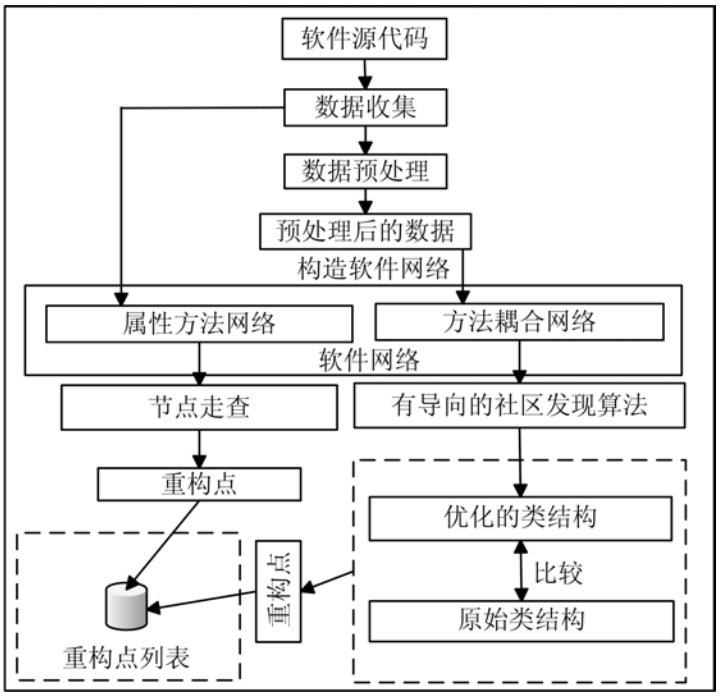


图 5.1 CR-CDSN 方法框架

5.3.1 重构的类型

面向对象软件类重构主要包括以下几种[190, 192]：

（1）提炼类（Extract Class）：若某个类做了应该由 2 个类做的事，则建立一个新类，将相关的属性和方法从旧类搬移到新类。

（2）将类内联化（Inline Class）：若某个类没有做太多事情，则将这个类的所有特征搬移到另一个类中，然后移除原类。

（3）移动属性（Move Attribute）：若某个属性被其所在类之外的另一个类更多的用

到，则在目标类建立一个新的属性，修改原属性所有的用户，令它们改用这个新属性。

（4）下置类属性（Push Down Attribute）：当一个父类中的某个属性字段不是对于每一个子类都必需时，就应该将其下置到具体使用到该属性的子类中去，以避免该属性被所有子类继承下来。

（5）提升类属性（Pull Up Attribute）：提升类属性与提升类方法一样，都是为了让子类能够继承父类中非私有的属性。

（6）方法下移（Push Down Method）：若父类中的某个方法只与部分（而非全部）子类有关，则将这个方法移到相关的那些子类去。

（7）移动方法（Move Method）：若有个方法与其所在类之外的另一个类进行更多的交流（调用后者或被后者调用），则在该方法最常引用的类中建立一个有着类似行为的新方法，将旧方法编为一个单纯的委托方法，或是将旧方法完全移除。

（8）提炼父类（Extract Superclass）：若两个类做类似的事情，则可以为这个两个类建立一个父类，将相同特性移至到父类。

（9）折叠继承体系（Collapse Hierarchy）：若父类和子类之间无太大区别，则可以将它们合为一体。

但是，这些重构如果没有必要的信息是不能够自动实施的。因为 CR-CDSN 方法使用社区发现算法自动识别重构点，因此我们必须像 [190] 那样做必要的限制。在 CR-CDSN 方法中，我们仅仅关注移动方法（Move Method）的重构（暂未考虑其它的重构方式），而且作了类似 [190] 的限制：

（1）被移动方法的目标类和原始类不能在同一个继承体系中；

（2）被移动的方法不是实现接口中方法的方法，也不是对父类方法的重写。

5.3.2 数据收集

CR-CDSN 方法的数据收集与 4 章 SoS-SN 方法数据收集的过程相同，主要是对软件的特征及特征之间关系的收集。同样的，这里特征间的关系也主要是两类，即：方法调用关系和方法使用属性的关系。本章所使用的数据也是由我们自主开发的软件网络分析工具 SNAT 自动提取的。为了提高 CR-CDSN 方法的可重复性，我们也已将本章中用到的部分工具已公开，可以从 [164] 处下载。

5.3.3 数据预处理

在实施软件类重构前，有必要将 5.3.2 节收集的方法根据它们在软件中发挥的作用进行区分，这主要是基于以下几点考虑：

（1）软件开发者在编码中往往会采用各种设计模式以提高软件的质量，但是有些设计模式往往会与现有的设计原则（如“高内聚，低耦合”等）相违背，例如：属于门面模式（facade pattern）的方法内聚性不高，因为它主要是将所有发来的请求委派到真正实现这个请求的类去。

（2）有些软件元素是为其它元素提供一些辅助性功能的，它们不能被孤立地看待。例如：Getter 方法主要用于从类外获取类对象的状态，它们可能被移动，但是仅仅移动它们，而没有将相应的属性或 setter 方法一起移动，那是毫无意义的。本章主要关注方法移动的重构，而不考虑移动属性之类的重构，因此像 Getter 和 Setter 这样的方法不在考虑之内。

因此，像属于设计模式的方法、 Getter 和 Setter 方法等是不能与普通方法同等看待的。CR-CDSN 方法首先将属于设计模式的方法识别出来[2]，并标识为特殊方法，表示这些方法不会被处理。本章将这一过程称之为数据预处理，该过程与 [190] 中的分类（classification）和 [211] 中的结构线索收集（architectural clue gathering）很类似。

为了识别这些特殊的方法，我们必须对所有的方法做一个系统的、彻底的排查。CR-CDSN 方法使用一种类似 [211] 中的方法，充分利用方法的结构和命名规则来识别这些特殊方法。CR-CDSN 方法还将下列几类方法视为特殊方法[190, 211]：

（1）Constant method：返回一个常量的方法。

（2）Empty method：方法体为空的方法。

（3）Getter 和 Setter method：也称为 Accessor 方法，它们的作用是简化对成员属性的访问。

（4）Collection accessor：一些没有太多语句的简单方法，主要用于为所在类增加一些对象。它们与属性紧密耦合，因此在不移动属性的情况下这些方法也是不能移动的。

（5）State method：用于检测对象的状态，它们与类和类内的属性紧密耦合。

（6）Alias method：将对这个方法的调用转移到同类中的其它一个方法。

（7）Factory methods：用于标识它是门面设计模式的实例。

（8）Delegation method：用于将调用转移到其它的对象。它本身不是一个设计模式，但是是很多设计模式（如 Facade 设计模式）的组成部分。

[2]设计模式的识别可以查看相应系统的文档，也可以使用 [187] 中的方法及工具，这里不再赘述。

5.3.4 软件网络

在数据预处理之后，我们可以构建两种特征粒度软件网络：属性方法网络和方法耦合网络，它们是 CR-CDSN 方法的基础。在本节中，我们将首先给出这两个网络的具体定义。

定义1：属性方法网络（Attribute Method Network, AMN）

AMN 是一个无向网络，节点表示一个特定软件系统中的特征（方法和属性）集合。每一个方法或属性仅用唯一的节点表示。两个节点间的边表示方法对属性的访问关系。也就是说，如果方法 A 访问属性 B，那么在表示属性 A 和属性 B 的两节点间就会有一条无向边。这里我们只考虑访问关系的存在，忽略访问关系的出现次数，也就是说，如果 A 访问 B 三次，在 AMN 中两者的相应节点间也只有一条有无向边。因此，AMN 可以表示为：

$$\mathrm{AMN} = (N_{AM}, E_{AM}) \tag{5.1}$$

其中：N_{AM} 是 AMN 的节点集合，表示系统中的特征集合（属性和方法）；E_{AM} 是 AMN 的边集，表示方法和属性间的访问关系。图 5.2 所示的是一个代码片段及其对应的 AMN。

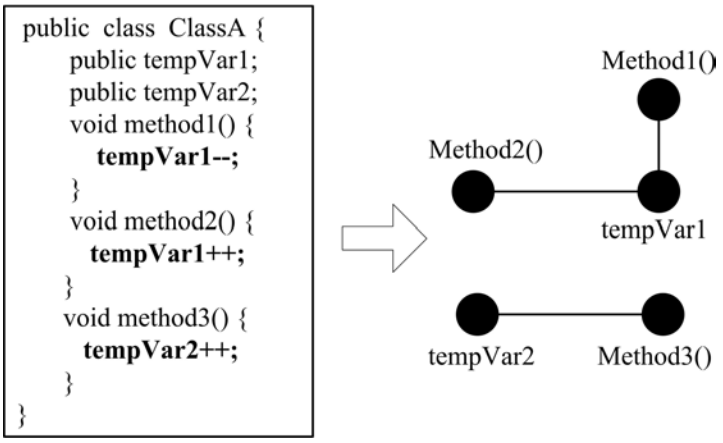


图 5.2　代码片段及其相应的 AMN

定义2：方法耦合网络（Method-Method Network, MMN）

MMN 是一个无向网络，节点表示一个特定软件系统中的方法。每一个方法仅用唯一的节点表示。两个节点间的边表示方法间的耦合关系。本章主要考虑方法间的两类耦合关系：（1）方法间由于直接调用而产生的耦合关系；（2）方法间通过访问共同的属性而产生的耦合关系。也就是说，如果方法 A（B）调用方法 B（A），那么在代表方法 A

和 B 的节点之间就会有一条无向边；如果方法 C 访问属性 E，方法 D 也访问属性 E，那么在代表方法 C 和 D 的节点间也会有一条无向边。同样的，这里我们只也考虑耦合关系的存在，而忽略其出现的次数。也就是说，如果方法 A 调用方法 B 三次， MMN 中两者相应节点间也只有一条有无向边。因此，MMN 可以表示为：

$$\text{MMN} = (N_M, E_M) \tag{5.2}$$

其中：N_M 是 MMN 的节点集合，表示系统中的方法；E_M 是 MMN 中的边集，表示方法间的耦合关系。图 5.3 所示的是一个代码片段及其对应的 MMN。

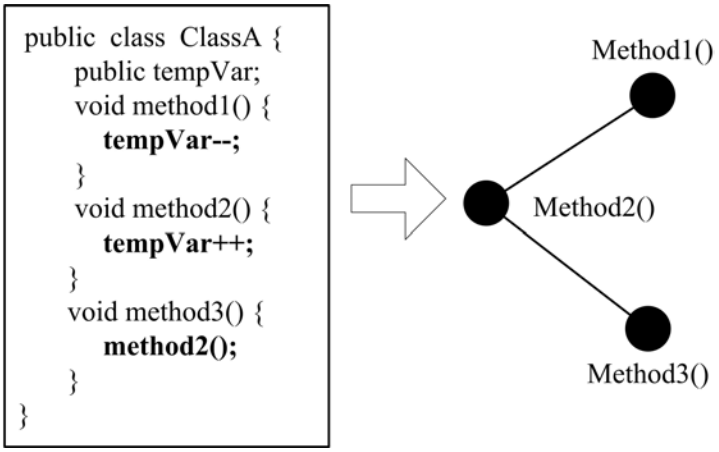


图 5.3　代码片段及其相应的 MMN

5.3.5　模块度评价指标

模块度评价指标用于评价社区划分的质量，它控制着方法移动的过程，因此对 CR-CDSN 方法至关重要。目前，用于评价社区划分质量的指标不少，例如 S. Mancoridis 等提出了模块化质量 MQ（Modularization Quality），A. Tucker 等提出 EVM 函数（EVM Function），M. E. J. Newman 提出了模块度（modularity）Q[162]。本章中我们也将使用模块度 Q 来评价社区划分的质量。M. E. J. Newman 在 [162] 中，将 Q 定义为：

$$Q = \sum_i (e_{ii} - a_i^2) \tag{5.3}$$

其中：Q 是网络的模块度；e_{ii} 表示两个端点都在社区 i 的边占边总数的比例；a_i 表示至少有一个端点在社区 i 的边占边总数的比例。

在执行了一次方法移动操作后，我们将重新计算 Q 值，以决定是否接受这次移动。因此，如何计算 Q 值对算法的性能具有重要的影响，特别是当网络的规模很大的时候。

在本章中，我们将计算 Q 的增加值 ΔQ，而不是 Q，这一思路与 [162] 同。将一个方法从社区 i 移动到社区 j 造成的 Q 值的变化可以计算如下：

$$\Delta Q = \Delta e_{ii} + \Delta e_{jj} - (\Delta a_{ii} + \Delta a_{jj}) \tag{5.4}$$

其中：Δe_{ii}、Δe_{jj}、Δa_{ii} 和 Δa_{jj} 分别是方法从社区 i 移动到社区 j 前后的差值。

在 CR-CDSN 方法中，算法会依次遍历每一个方法节点，若这个节点与其它类的节点有联系，我们就将该节点移动到与其相连的那些节点中产生最大 ΔQ 的那个节点所在的类中。

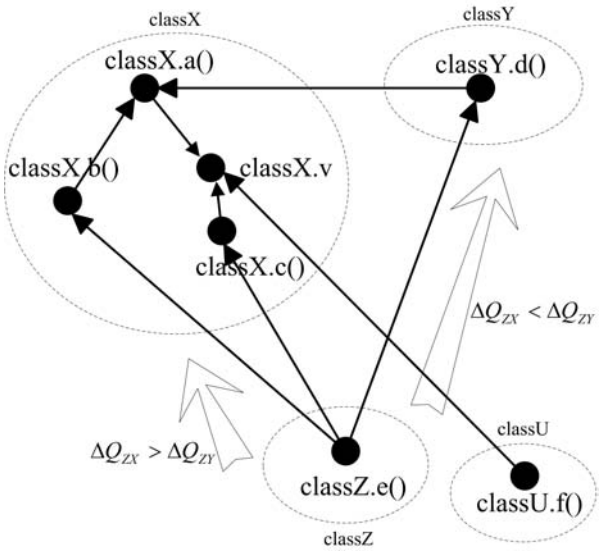


图 5.4　方法移动示意图

如图 5.4 所示，虚线圈起来的部分表示一个类，虚线内的节点表示类内的方法和属性。在 CR-CDSN 方法中，当算法遍历到节点 $e()$ 时，会完成以下步骤：

（1）判断节点 $e()$ 与哪些类有直接的联系：节点 $e()$ 与类 $classX$ 和 类$classY$ 内的方法有直接的调用关系，而与类 $classU$ 中的方法没有直接的关系（肯定不会移动至没有直接关系的类 $classU$）；

（2）CR-CDSN 方法会试探性的将节点 $e()$ 移至有直接关系的其它类，即：从类 $classZ$ 分别移至类 $classX$ 和类 $classY$，然后计算在这两次移动过程中产生的 ΔQ，即：ΔQ_{ZX} 和 ΔQ_{ZY}。如果 $\Delta Q < 0$，则令 $\Delta Q = 0$。

（3）比较 ΔQ_{ZX} 和 ΔQ_{ZY} 的大小：如果 $\Delta Q_{ZX} > \Delta Q_{ZY}$，则将节点 $e()$ 从类 $classZ$ 移至类 $classX$；如果 $\Delta Q_{ZX} < \Delta Q_{ZY}$，则将节点 $e()$ 从类 $classZ$ 移至类 $classY$；如果 $\Delta Q_{ZX} = \Delta Q_{ZY}$，则将节点 $e()$ 随机移至类 $classX$ 或类 $classY$ 之一。

5.3.6 有导向的社区发现算法

正如 5.2.2 节所述，复杂网络中已有很多社区发现算法。其中大多数社区发现算法都是通过最大化模块度（modularity）来得到社区发现结果的[199, 202, 207, 208]。本章也使用模块度来评价 MMN 的社区结构。然而研究表明，使用模块度进行社区发现存在分辨率极限问题[212, 213, 214]，即网络中能发现的社区的大小与网络中边的规模有关，小于某一个尺度的社区不能被发现，而通常被合并为大的社区。

为了解决这一问题，本章提出了一种有导向的社区发现算法——GCDA（guided community detection algorithm）。GCDA 算法运行之初将每个方法分配到了一个特定的类中（这个特定的类就是这个方法定义的类），而不像 [162] 那样分配到一个随机的类。然后通过不断的方法移动操作以提高模块度。GCDA 这样处理的理由是：软件类结构是 MMN 的自然社区结构，大部分的方法都在正确的类中，只有少数的几个方法位置放的不好，降低了内聚，增加了耦合，必须移动。因此，我们可以以一种有导向的方式，即开始时各个方法都属于它们的自然社区中来进行社区探测。必须说明的是，GCDA 实际上做的是虚拟的移动，它是假设某个方法从一个类移动到另外一个类，然后计算网络的模块度，而实际的代码并没有做任何修改。

算法 5.1 显示的是 GCDA 的算法流程，其时间复杂度是 $O(N_m^2)$。其中：$\mathbf{CM}[i][j]$ 是节点连接矩阵，如果 $\mathbf{CM}[i][j] = 1$ 表示节点 i 和节点 j 之间有一条边，否则，它们之间不存在边；$|N_M|$ 表示 MMN 中的节点数；$nodeId[]$ 是一个数组，保存这节点所属社区的标识，所以 $nodeId[i]$ 表示节点 i 属于社区 $nodeId[i]$。$bvisited[]$ 是一个布尔型数组，存储节点是否已被访问，所以 $bvisited[i]$ = 'true' 表示节点已经被访问过了，否则，没有被访问过。$bSpecial[]$ 是一个布尔型数组，存储节点是否是一个特殊节点，所以 $bSpecial[i]$ = 'true' 表示节点 i 是特殊节点，否则不是。

5.3.7 节点走查

节点走查指：依次遍历 AMN 中的每个方法节点，判断该方法是否仅访问了属于另外类的属性，如果存在这样的节点，它们同样也被看成是重构点，必须做适当的移动，即：将这样的方法移至它所访问属性类型所属的类。因为这样的方法也增加了类之间的耦合，降低了软件的质量。如图 5.5 所示，类 $ClassB$ 中的方法 $method1()$ 和 $method2()$ 仅访问了具有 $ClassA$ 类型的属性，所以它们是重构点，因此，它们必须被移动到类 $ClassA$。通过这种移动可以提高内聚性。图 5.5 中，左边部分是原始的代码，右边部分是方法移动后的代码。

算法 5.1: GCDA 算法

输入:

方法耦合网络 MMN;

输出:

优化后的类结构及模块度 Q;

1: 初始化 $\mathbf{CM}[i][j]$, $nodeId[i]$（属于同一个类的方法对应的节点分配相同的社区标识），$bVisited[i]$ = false 和 $bSpeical[i]$（根据数据预处理的结果）

2: 根据公式 5.3 计算 Q 值;

3: **for**$(i = 1; i <= |N_M|; i++)$ {//No.1

4: **for**$(j = 1; j <= |N_M|; j++)$ {//No.2

if$(\mathbf{CM}[i][j]$ && !$bVisited[i]$ && $nodeId[i]$!= $nodeId[j]$ && !$bSpecial[i])$

{//No.3

$bVisited[i]$ = true;

尝试将节点 i 移至社区 $nodeId[j]$，计算 ΔQ，并存至数组 $\Delta Q[]$

}//No.3

}//No.2

5: 从 $\Delta Q[]$ 选择最大的 ΔQ，令其为 ΔQ_{max}，

if($\Delta Q_{max} > 0$) {//No.4

则将节点 i 移至取得最大 ΔQ 的 $nodeId[j]$ 所在社区;

for$(j = 1; j <= i; j++)$ {//No.5

if($\mathbf{CM}[i][j]$ = true) {//No.6

$bVisited[j]$ = false;

}//No.6

}//No.5

$i = 1$;

$Q = \Delta Q + Q$;

}//No.4

}//No.1

6: **返回:** 优化后的类结构及模块度 Q。

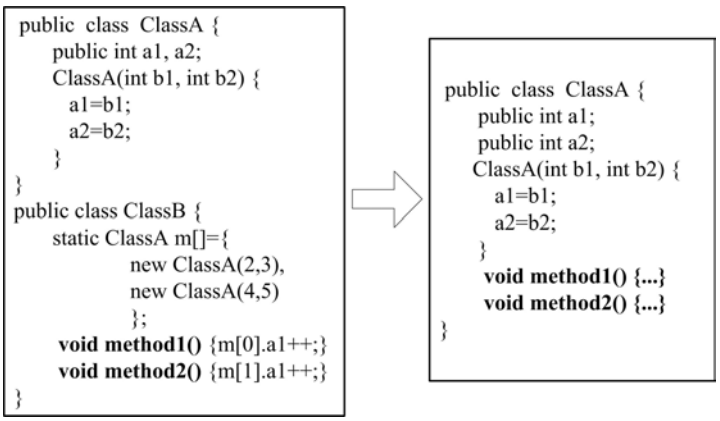


图 5.5　节点走查示例

5.4　实验设计

为了验证 CR-CDSN 方法的有效性，本小节将结合开源 Java 软件 JHotDraw 5.1 进行说明。在 5.4.1 节，我们首先介绍一下 JHotDraw 5.1 系统；在 5.4.2 节，我们将用 CR-CDSN 方法对 JHotDraw 5.1 系统进行重构。

5.4.1　系统简介

JHotDraw 是用 Java 语言开发的一个开源项目，最初由设计模式领军人物 Erich Gamma 等人设计开发，是运用面向对象设计模式的一个典型案例。目前，JHotDraw 已经发展成为一个著名的二维图形编辑器的应用框架，有良好的可复用性和可扩展性。[190, 197] 曾用　JHotDraw 5.1 版本进行案例分析，为了与现有的工作进行对比，我们也使用 JHotDraw 5.1 作为实验对象。表 5.1 显示的是 JHotDraw 5.1 版本的一些统计数据。

表 5.1　JHotDraw 5.1 统计数据

参数名	参数值	参数名	参数值
总文件数	144	空行数	2,448
总大小	144 KB	类数	172
总行数	15,430	方法数	1,277
代码行数	8,419	属性数	443

5.4.2 实验及结果

我们的实验是在一台 PC 机上进行的，这台 PC 机的 CPU 主频是 2.53 GHz，内存为 1 GB。我们使用 SNAT 提取数据，并对数据进行预处理，最后构建了 JHotDraw 5.1 版本的 AMN 和 MMN。

图 5.6 显示的是 JHotDraw 5.1 版本的 AMN。这个 AMN 不是一个全连通图，它由几个小的连通图构成（每一种颜色代表一个连通子图）。表 5.2 是该 AMN 的一些统计数据。

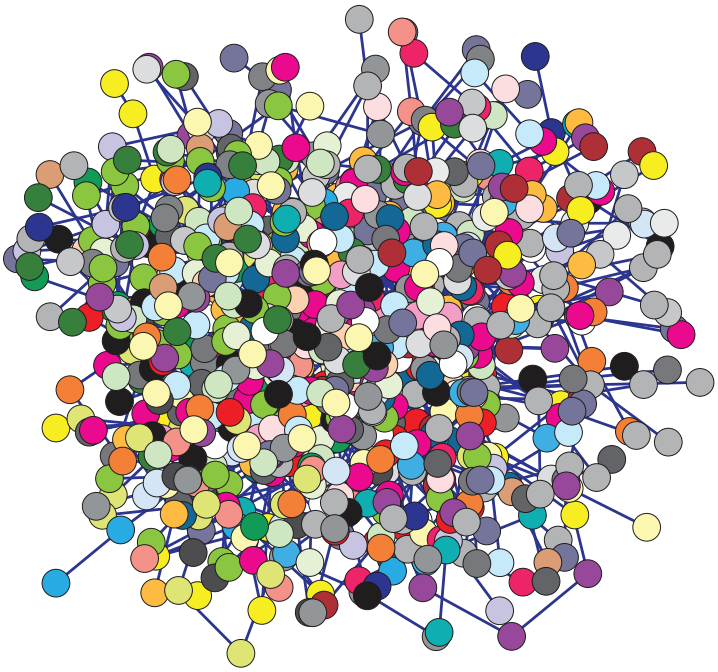


图 5.6　JHotDraw 5.1 版本的 AMN

表 5.2　AMN 的一些统计数据

参数名	参数值
总属性/方法数	957
AMN 数	136
AMN 最小规模	2
AMN 最大规模	49

表 5.3 显示的是通过节点走查得到的重构点。该表第一列对应要移动的方法的名字；第二列对应该方法应该移到的目标类。为了清晰起见，我们省略了方法的参数列表。图 5.7 显示的是这些重构点所在的那个连通图，它是整个 AMN 的一小部分。

表 5.3　AMN 中的重构点

方法名	目标类
CH.ifa.draw.util.ColorMap.Size()	CH.ifa.draw.util.ColorEntry
CH.ifa.draw.util.ColorMap.static()	CH.ifa.draw.util.ColorEntry
CH.ifa.draw.util.ColorMap.name()	CH.ifa.draw.util.ColorEntry
CH.ifa.draw.util.ColorMap.colorIndex()	CH.ifa.draw.util.ColorEntry
CH.ifa.draw.util.ColorMap.color()	CH.ifa.draw.util.ColorEntry

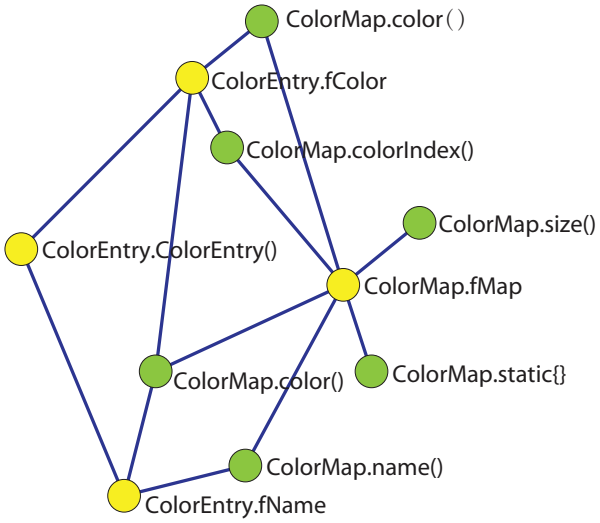


图 5.7　AMN 中包含重构点的小连通图（重构点为绿色节点）

图 5.8 显示的是 JHotDraw 5.1 版本的整个 MMN。这个 MMN 也不是一个全连通图，它由几个小的连通图构成。这里我们仅研究该 MMN 的最大连通图部分（图中绿色部分），因为社区发现算法一般是针对连通图进行的。同时，我们发现不在最大连通图内的节点数量很小，忽略这些节点对我们算法的效果不会产生太大的影响。数据预处理之后，CR-CDSN 方法仍然还有 227 个方法需要处理。

在 JHotDraw 5.1 的 MMN 中，仅有 5 个方法（如表 5.4 和图 5.9）是需要移动的。表 5.4 的第一列对应要移动的方法的名称，第二列对应该方法应该移动到的目标类。为了清晰起见，我们省略了方法的参数列表。

图 5.10 显示的是算法运行中模块度 Q 随时间的变化曲线。我们可以发现 Q 的最大值是 0.5281，最小值是 0.52724。

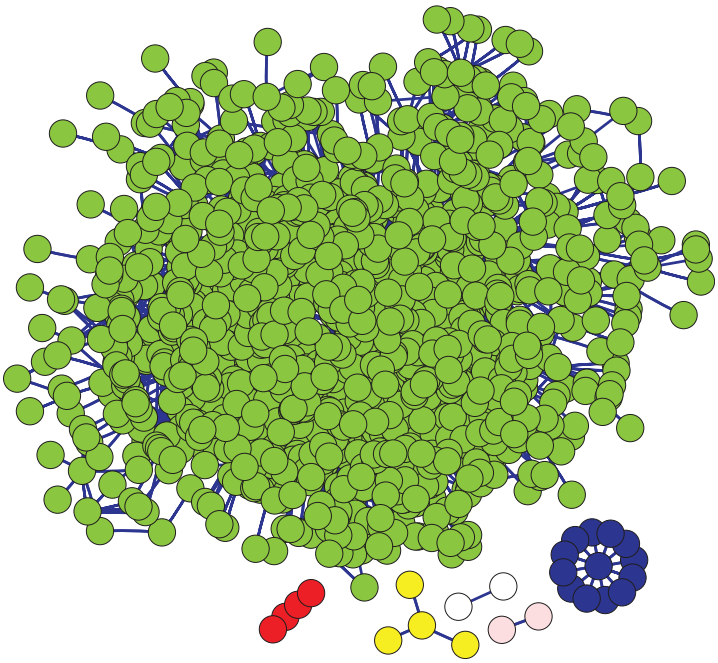


图 5.8　JHotDraw 5.1的 MMN（每一种颜色代表一个连通图）

表 5.4　MMN 中的重构点

方法名	目标类
CH.ifa.draw.util.PolygonFigure.chop()	CH.ifa.draw.util.Geom
CH.ifa.draw.util.PertFigure.writeTasks()	CH.ifa.draw.util.StorableOutput
CH.ifa.draw.util.PertFigure.readTasks()	CH.ifa.draw.util.StorableInput
CH.ifa.draw.util.TextTool.fieldBounds()	CH.ifa.draw.util.standard.TextHolder
CH.ifa.draw.util.TextTool.beginEdit()	CH.ifa.draw.util.standard.TextHolder

5.4.3　实验分析

从前面的分析可以发现，CR-CDSN 方法在 JHotDraw 5.1 中共找到了 10 个重构点（如表 5.3 和表 5.4 所示）。为了检查 CR-CDSN 方法所找到的重构点对开发者来说是否有意义，我们人工对这些重构点进行了分析。通过查看源代码，我们发现 CR-CDSN 方法找到的这些重构点是有意义的。例如 CR-CDSN 方法建议将表 5.3 中的所有方法移动至类 CH.ifa.draw.util.ColorEntry，我们查看了源代码，发现这些方法都仅仅访问了 CH.ifa.draw.util.ColorEntry 类型的数组； CH.ifa.draw.util.PolygonFigure.chop() 被建议移至类 CH.ifa.draw.util.Geom，查看源代码我们发现，它调用了类 CH.ifa.draw.util.Geom 中的方法 length2() 和 intersect()，但是没有使用本身所在类中的任何属性和方法。

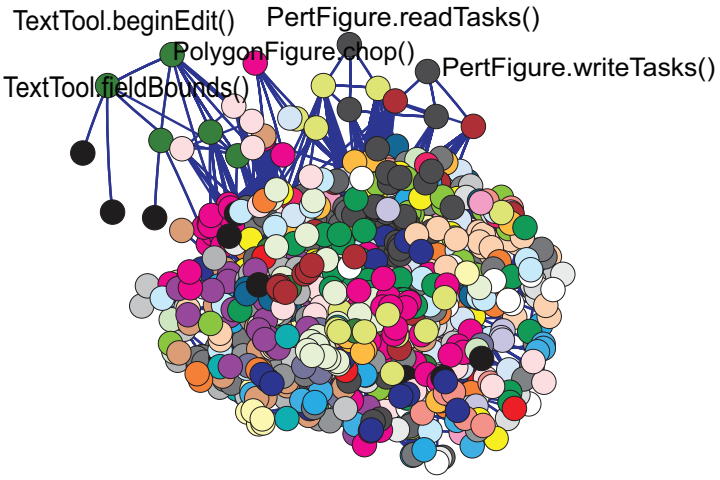


图 5.9　MMN 中的重构点（边上的文字是相应节点对应的方法名）

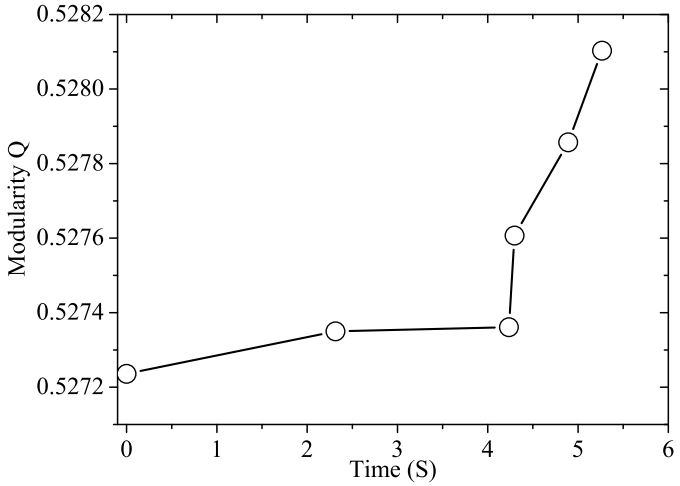


图 5.10　模块度 Q 随时间的变化曲线

将 CH.ifa.draw.util.PolygonFigure.chop() 从类 CH.ifa.draw.util.PolygonFigure 移至类 CH.ifa.draw.util.Geom 有利于减少两类间的耦合。PertFigure.writeTasks、PertFigure.readTasks、TextTool.fieldBounds 和 TextTool.beginEdit 等重构点也可以通过同样的方式来解释。

同时，我们将 CR-CDSN 方法与其它同样使用 JHotDraw 5.1 作为研究对象的工作 [190, 197] 进行对比，我们发现：（1）CR-CDSN 方法在 JHotDraw 5.1 中获得的重构点与 [190, 197] 中的很类似；（2）CR-CDSN 方法的效率比 [190, 197] 的要高：使用 CR-CDSN 方法仅需 6 秒中就可以完成，[190, 197] 中的方法分别需要 30 分钟和 5 分中；（3）CR-CDSN 方法为软件重构提供了新思路，它使用网络模型抽象软件内部结构，借鉴复杂网络社区发现的思想完成软件重构；（4）CR-CDSN 方法仅推荐了一个重构点的列表，但是是否实施重构工作还得由开发者自己最终作出判断。

我们无法与更多的工作进行比较来说明 CR-CDSN 方法的优势性，因为好多已有的方法，包括 [190, 197] 在内，都仅提供了它们发现的重构点，我们无法获取其它更为详细的信息。

5.5 本章小结

软件重构是一种提高软件结构设计和可维护性的有效方式。在本章中，我们将复杂网络社区发现算法应用于面向对象软件的类重构，提出了面向对象软件类重构的 CR-CDSN 方法，用于识别方法移动的重构。该方法的基本思想是软件工程实践中倡导的“高内聚，低耦合”的设计原则，它首先将面向对象软件系统在特征粒度被抽象成两类软件网络，即：AMN 和 MMN；然后借鉴复杂网络社区发现算法，在充分尊重设计模式作用的前提下提出了一种有导向的社区发现算法 GCDA 用于识别 MMN 中的重构点；同时，提出节点走查的方式来识别 AMN 中的重构点。本文的方法取得了与其它方法类似的结果，但是效率比其它的方法要高。

本文提出的 CR-CDSN 方法可以为开发者实施重构工作提供指导。可以将本文提出的方法实现为集成开发环境（如 Eclipse 等）的插件或开发独立的软件工具，用于实际的软件开发中，具有较好的实用前景。

进一步的研究工作包括：(1) 使用其它更多的开源软件来验证本文提出的方法；(2) 实现更多的重构，例如：提炼类、移动属性、提升方法等；(3) 开发相应的插件，并集成到开发环境中。

本章实验部分所有数据，包括 JHotDraw 5.1 系统、软件网络数据文件、实验结果数据、部分工具等都可以从 [164] 处下载。

第六章　面向对象软件回归测试用例排序

“软件测试与维护的费用占了整个软件开发费用的 *50%* 以上[215]。”

—— *G. J. Myers*

“测试是最有效的排除和防止软件缺陷与故障的手段[215]。”

—— *G. J. Myers*

6.1　引言

随着用户需求和系统运行环境的不断变化，软件的变更在所难免。为了检验变更是否正确以及变更是否引入新的错误，有必要对变更后的软件系统重新进行测试，这就是所谓的回归测试[216]。回归测试是软件演化过程中一项频繁进行且开销巨大的维护活动，占了软件维护总费用的三分之一以上[217]。因此，如何提高回归测试的有效性和测试效率，降低回归测试的成本是软件工程界亟待解决的任务之一。

测试用例排序技术是一种致力于提高测试用例使用效率的回归测试技术，它依据测试的历史信息，综合考虑各种因素（代码覆盖率、错误检出率、测试用例执行时间、测试用例对需求的满足情况等），赋予每个测试用例一定优先级，在回归测试过程中按优先级自高到低选取和执行测试用例[218]。研究表明，测试用例排序技术可以帮助测试人员尽早发现软件中的错误，提高测试效率，降低测试的时间开销和人力成本[218, 219, 220, 221, 222]。目前，测试用例排序技术大多基于覆盖率（代码覆盖率等）相关信息对测试用例进行排序，如何将其它非覆盖因素进行综合考虑，仍是测试用例排序技术研究的重点和难点之一[222]。

复杂性是软件的本质属性之一，它使软件系统（结构、行为等）难以理解，进而影响软件过程管理和软件的维护与二次开发。实践证明，软件的复杂性是导致软件错误的主要因素[223]。同时，随着体系结构研究的深入，人们逐渐意识到：软件系统的（拓扑）结构会影响其功能、性能和可靠性等其它指标[169]。近年来，一些复杂系统的研究者将复杂网络的方法引入到软件系统的拓扑结构分析中，用软件的网络模型抽象面向对象（Object-Oriented，OO）软件系统，即节点代表软件的组成实体（属性、方法、类、包等）；边代表这些实体间的依赖关系（方法调用关系、类继承关系等），对大量 OO 软件系统各个粒度的拓扑结构进行研究，发现软件系统的结构并不是随机和无序的，大多数都展现出“小世界”（small-world）和“无标度”（scale-free）等复杂网络特征[63, 64, 65]。

软件系统作为一种无标度网络，具有一个显著的特点，即：网络内的节点在系统中的重要性是不一样的，“hub” 节点的错误在软件网络中通过传播不断放大，进而导致整个软件系统的失效，而其它节点的错误其影响往往是有限的、局部的[224, 225]。因此，测试用例排序技术的研究也应该考虑软件复杂性和软件拓扑结构的这些特征，而这一点往往为人们所忽略。

本章从软件的复杂性和静态结构出发，提出了一种基于加权类依赖网络的面向对象软件回归测试用例排序方法 TCP-WCDN（Test Case Prioritization for regression testing of object-oriented software based on Weighted Class Dependency Networks）。TCP-WCDN 用加权类依赖网络模型（Weighted Class Dependency Network，WCDN）抽象类粒度的 OO 软件系统；通过度量类的复杂性评价修改该类时引入错误的可能性；提出用类节点的错误通过 WCDN 传播影响的其它节点数量的比率评价错误的严重性，并且将仿真方法引入度量指标的计算中。在此基础上，提出度量指标评价类的测试重要性，并结合测试用例的覆盖信息，对测试用例进行排序。本章将 TCP-WCDN 运用于开源 OO 软件系统 Xml-Security[226] 和 JTopas[227] 测试用例的排序中，并通过与其它方法的比较，验证了 TCP-WCDN 的有效性。

6.2 相关研究工作

测试用例排序问题可以形式化的定义如下：

定义1：测试用例排序问题[218]

给定测试用例集 T，T 的全排列集合 PT，以及从 PT 到实数的函数 f，求一个 $T' \in PT$，使得 $(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]$。其中，$PT$ 包含了测试用例所有可能的执行顺序，而函数 f 的值代表了对测试结果的评价。

围绕测试用例排序问题，人们开展了大量的研究工作。S. Elbaum 等结合测试用例历史覆盖信息，提出了 12 种不同的测试用例排序方法[228]。G. Rothermel 等提出了基于测试用例分支覆盖能力的排序方法[218]。S. Elbaum 等将测试用例执行耗费和错误严重度引入测试用例排序中[229]。J. M. Kim 等综合考虑了各种资源约束对测试的影响，并基于这些因素赋予测试用例不同优先级[219]。J. Jones 等研究了基于 MC/DC 覆盖率的测试用例排序技术[230]。H. Srikanth 根据测试用例对需求的满足情况赋予测试用例不同的优先级[231]。Walcott 等基于测试用例的历史执行时间对测试用例进行排序[232]。屈波等提出了基于测试用例设计信息的测试用例排序方法[233]。L. Zhang 等人使用线性规划方法求解测试用例排序问题[234]。L. Zhang 等人根据测试用例的静态调用图，提出了不

需要代码覆盖信息的测试用排序技术[235]。H. Do 等将基于覆盖的测试用例排序技术运用于 JUnit 测试环境[236, 237]。

尽管已有很多测试用例排序方面的研究工作，但是现有的技术都忽略了软件复杂性和软件结构对测试用例优先级带来的影响。研究表明，修改复杂度高的类比修改复杂度低的类更容易错误[36, 37, 238, 239]，错误在不同的类产生对系统的破坏程度存在差异[178]。如图 6.1 所示（红色表示植入的错误，双删除线部分为原本正确的代码），类 A 若引入错误，它将可能影响 A、B、C 和 D 四个类（因为 B 的方法 b、C 的方法 c 和 D 的方法 d 都调用了 A 的方法 a）；类 D 若引入错误只可能影响 D 本身。因此，测试用例排序技术的研究也应该考虑软件复杂性和软件结构的这些特征。

```
public class A
{  ……
   public boolean a(int t1,int t2, int t3) {
     if(t1>t2 && t1>t3) return true;
     if(t1>t2 || t1>t3)
     else return false;
   }
   ……
}
```

```
public class C
{  ……
   public int c(int t1,int t2, int t3) {
     A cA=new A();
     if(cA.a(t1,t2,t3)) return 3;
     else return 4;
   }
   ……
}
```

```
public class B
{  ……
   public int b(int t1,int t2, int t3) {
     A cA=new A();
     if(cA.a(t1,t2,t3)) return 1;
     else return 2;
   }
   ……
}
```

```
public class D
{  ……
   public int d(int t1,int t2, int t3) {
     A cA=new A();
     if(cA.a(t1,t2,t3)) return 5;
     else return 6;     return 6;
   }
   ……
}
```

图 6.1　植入错误的代码片段（红色部分是植入的错误）

6.3　TCP-WCDN 方法

从回归测试用例排序问题的定义可以看出，要对测试用例进行排序，首先必须给出函数 f 的定义。然而，由于难以直接构造 f，大部分研究工作都是基于单个测试用例优先级计算，以及在此之上的最优排列的构造或搜索方法。本章在定义测试用例优先级的时候，综合考虑了软件复杂性和软件结构对测试用例优先级的影响，即：

（1）类复杂度的不同造成修改类时引入错误的概率不同；

（2）错误产生位置的不同（不同类）造成错误对整个系统影响的不同。

本章提出的 TCP-WCDN 的框架如图 6.2 所示。以下分节详细阐述图 6.2 中的关键步骤。

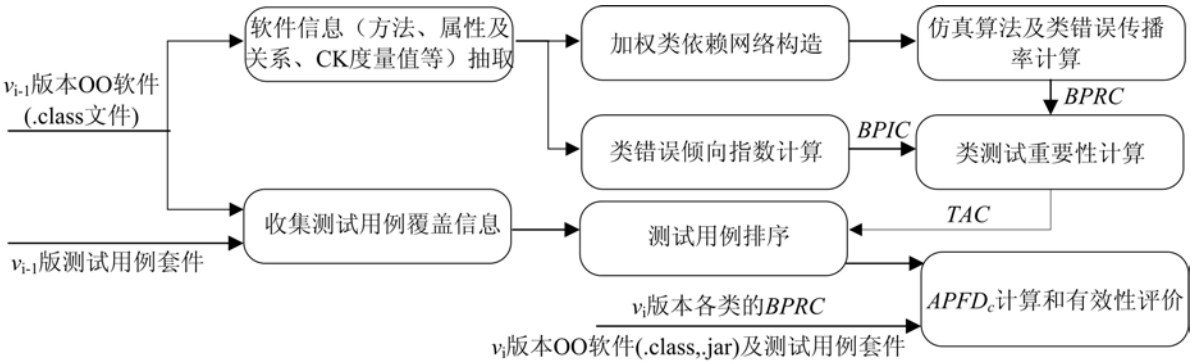


图 6.2　TCP-WCDN 的框架图

6.3.1　数据收集

本章仍使用面向对象软件作为研究对象，理由见第 3 章 3.3.1 小节，这里不再赘述。TCP-WCDN 方法的数据收集过程与第 5 章 CR-CDSN 方法的相同，主要是对软件的特征（方法、属性）及特征之间关系的收集。这里主要考虑特征间的两类关系：方法调用关系和方法使用属性的关系。同时，我们也要计算 $SLOC$、WMC、CBO 等一些常用的复杂性度量指标（具体定义见 6.3.3.1 节）。本章软件网络的获取和相关度量的计算也是由我们自主开发的软件网络分析工具 SNAT 完成的。当然，我们也可以借助一些开源的工具来获取 $SLOC$、WMC、CBO 等数据，如 JMetric[240]、OODMetrics[241]、Code Critickbeta[242] 等，具体的使用方式在这里不再赘述，请查看相关工具的操作手册（manual）。

6.3.2　软件网络

如第 2 章所述，使用网络模型研究 OO 软件系统主要集中于 3 个粒度：方法、类和包。本章将以类粒度的软件网络模型研究 OO 软件回归测试用例排序问题，主要出于以下考虑：

（1）TCP-WCDN 需要获得测试用例在前一个版本的覆盖信息，而目前广泛用于 Java 项目的软件测试工具（如 JUnit[243]、djUnit[244] 等）一般只提供类级的测试覆盖信息；

（2）本章从复杂性的角度评价类的错误倾向性，然而现有的关于复杂性和类错误倾向性方面的研究都是类粒度的；

（3）研究表明，测试的粒度对软件回归测试用例排序技术性能的影响不大[222]。

我们曾在第 4 章 4.3.2.1 节提出了 OO 软件系统的加权特征依赖网络 WFDN 模型，本章将沿用该模型，并对其做必要的修改。同时，在修改后的 WFDN 基础之上，我们将构造 OO 软件系统的加权类依赖网络模型。下面将给出本章加权特征依赖网络和加权类依赖网络的定义。

定义2：加权特征依赖网络（Weighted Feature Dependency Network, WFDN）

OO 软件系统的 WFDN 可以用一个有向图表示，即 WFDN $= (N^f, E^f, \boldsymbol{M_p^f})$。其中：$N^f$ 为节点的集合，表示软件系统中所有特征（方法和属性）的集合；E^f 是有向边的集合，表示特征间的依赖关系；$\boldsymbol{M_p^f}$ 是一个矩阵，如果在 WFDN 中有一条有向边 $< j, i >$（<> 表示一个有序对，下同）从节点 j 指向节点 i，则该边边权 $\boldsymbol{M_p^f}(i,j)$ 等于 1，否则 $\boldsymbol{M_p^f}(i,j)$ 等于 0。$\boldsymbol{M_p^f}(i,j)$ 表示如果节点 i 出错了，错误会以 $\boldsymbol{M_p^f}(i,j)$ 的概率传播至节点 j。

操作符 $gma(y)$ 返回方法 y 引用的所有属性的集合；操作符 $gmm(y)$ 返回方法 y 调用的所有方法的集合。

假设 r 是一个属性，p 和 q 是两个方法，则对应在 WFDN 中有三个节点 $r, p, q \in N^f$。如果 $q \in gmm(p)$，则：$< p, q > \in E^f$；如果 $r \in gma(p)$，则 $< p, r > \in E^f$。

图 6.3 显示的是一个代码片段及其对应的 WFDN。

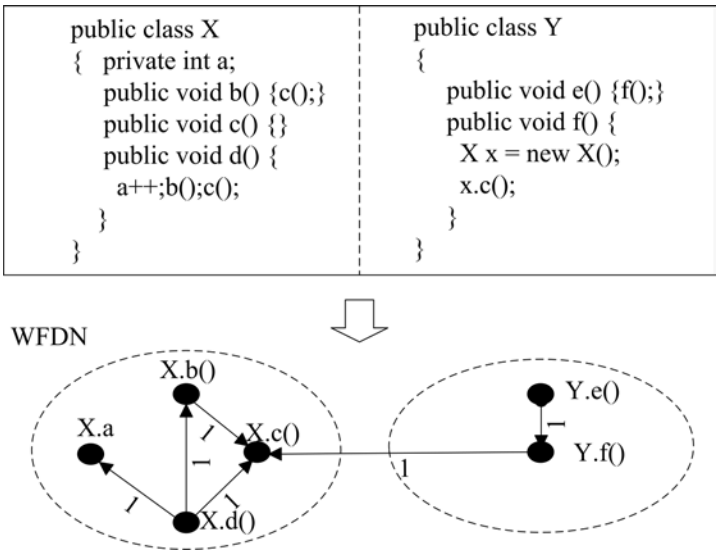


图 6.3　代码片段及其对应的 WFDN

定义3：加权类依赖网络（Weighted Class Dependency Network, WCDN）

OO 软件系统的 WCDN 是一个有向图，用 WCDN $= (N^c, E^c, \boldsymbol{M_p^c})$。其中：$N^c$ 为节点的集合，表示软件系统中的类集合；E^c 是有向边的集合，表示类之间的依赖关

系；$\boldsymbol{M_p^c}$ 是一个矩阵，如果在 WCDN 存在一条有向边 $< j,i >$ 从节点 j 指向节点 i，则其边权 $\boldsymbol{M_p^c}(i,j)$ 等于 p_{ij} $(0 < p_{ij} \leq 1)$，否则 $\boldsymbol{M_p^c}(i,j)$ 等于 0。$\boldsymbol{M_p^c}(i,j)$ 表示如果节点 i 出错了，错误会以 $\boldsymbol{M_p^c}(i,j)$ 的概率传播至节点 j。

本章定义的 WCDN 与 [63, 64, 65] 中的类耦合网络定义有所不同。本章的 WCDN 是定义在 WFDN 之上，类之间的依赖关系是由其包含的特征之间的依赖关系定义的，是一种加权网络，而文献中是无权网络。

操作符 $gfc(x)$ 返回特征 x 所属的类。如果两个特征$j,k \in N^f$, $gfc(j) = m$, $gfc(k) = n$，且满足 $m \neq n, < j,k > \in E^f$，则 $m,n \in N^c, < m,n > \in E^c$。

接下来，将详细叙述 $\boldsymbol{M_p^c}$ 的计算方法，在这之前有两个基本概念须作说明。

定义4：方法的类间可达集（Extra-Class Method Reachability Set, $ECMRS$）

假设有两个不同的类 i 和 j，且 $i,j \in N^c, < i,j > \in E^c$，则类 i 中任一方法 z 的类间可达集 $ECMRS(z,j)$ 是类 j 中可以从方法 z 到达的特征集合。类 j 中特征 w 可以从 z 到达，满足：(1) WFDN 中，z 和 w 间存在一条有向路径 $z \rightarrow ... \rightarrow w$；(2) 该有向路径上的所有特征都属于类 i 或 j，并且每个特征在路径上仅出现一次。所以类中一个特定方法的 $ECMRS$ 是另一个类中可以对该方法产生直接或间接影响的特征集。

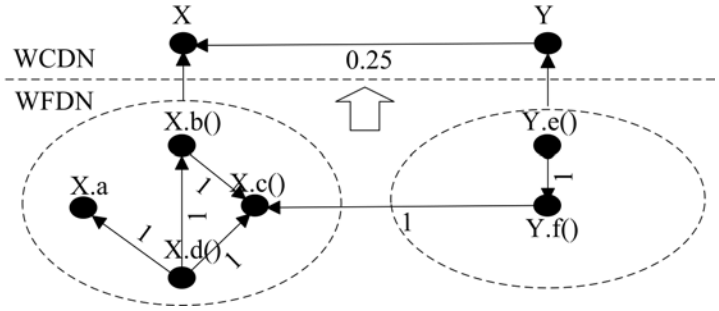


图 6.4　代码片段及其对应的 WFDN

图 6.4 是图 6.3 中 Java 代码片段对应的 WFDN 和 WCDN。如图 6.4 所示，类 Y 依赖于类 X，且 $Y.e() \rightarrow Y.f() \rightarrow X.c()$、$Y.f() \rightarrow X.c()$，所以 $ECMRS$(Y.f(), X) = {X.c()}。

我们提出算法 6.1 计算每个方法的 $ECMRS$：

通过算法 6.1, 我们可以在 $O(|N^f|)$ 时间内得到各个方法的 $ECMRS$。$|\Omega|$ 表示求集合 Ω 中元素的个数。下同。

定义5：类内传播域（Inter-Class Ripple Basin, $ICRB$）

类 i 相对于类 j 的类内传播域 $ICRB(i,j)$ 是类 j 中直接或间接为类 i 中的方法

算法 6.1: $ECMRS$ 求解算法

输入:

软件的 WFDN，软件 WCDN的 E^c；

输出:

每个方法的 $ECMRS$；

注：操作符 $gcm(c)$ 返回类 c 包含的方法的集合，$gcf(c)$ 返回类 c 包含的特征的集合。下同。

1: 从 E^c 选择一条有向边 $< j, i >$，并将其从 E^c 删除；

2: 遍历 N^f 中各节点 x，如果节点 $gfc(x)$ 等于 i，则将节点 x 加入队列 FeatureQueue 中；

3: 弹出 FeatureQueue 中的队首节点 v_s，并将其移入集合 S，将所有 v_s 直接依赖的节点 y（满足 $gfc(y)$ 等于 i 或 j）存入 FeatureQueue 中；

4: 反复执行步骤 3，直到 FeatureQueue 为空；

5: 令 $ECMRS(x, j) = S \cap gcf(j)$，$S = \emptyset$；

6: 反复执行步骤 $1 \sim 5$，直到 E^c为空；

7: **返回：**每个方法的 $ECMRS$。

使用的特征组成的集合，即：

$$ICRB(i, j) = \sum_{m \in gcm(j)} \cup ECMRS(m, j) \tag{6.1}$$

一个类的方法必须通过那些与另一个类有直接依赖关系的方法才能与另一个类中的特征发生依赖关系，我们将类内这样一些方法称为临界方法。如图 6.4 所示，类 Y 中的方法 $e()$ 必须通过方法 $f()$（因为 $f()$ 直接依赖于类 X 中的特征 $X.c()$）才能间接的与类 X中的特征发生联系，我们用操作符 $gccm(c)$ 返回类 c 的临界方法。因此：

$$ICRB(i, j) = \sum_{m \in gcm(j)} \cup ECMRS(m, j) = \sum_{m \in gccm(j)} \cup ECMRS(m, j) \tag{6.2}$$

图 6.4 中 $ICRB(\mathrm{Y}, \mathrm{X}) = \sum_{m \in \{\mathrm{Y.e(),Y.f()}\}} \cup ECMRS(m, \mathrm{X}) = \sum_{m \in \{\mathrm{Y.e()}\}} \cup ECMRS(m, \mathrm{X})$ $= ECMRS(\mathrm{Y.e()}, \mathrm{X}) = \{\mathrm{x.c()}\}$。

基于上述的论述，对于任一类对 i 和 j, $i, j \in N^c$, $< i, j > \in E^c$，我们将错误从类 j

传播到类 i 的概率 $\boldsymbol{M_p^c}(j,i)$ 定义为：

$$M_p^c(j,i) = \frac{|ICRB(i,j)|}{|gcf(j)|} \tag{6.3}$$

因此，图 6.4 中，$M_p^c(\mathrm{X},\mathrm{Y}) = |\{\mathrm{X.c()}\}|/|\{\mathrm{X.a},\mathrm{X.b}(),\mathrm{X.c}(),\mathrm{X.d}()\}| = 1/4 = 0.25$。

6.3.3 类测试关注度计算

一个类如果极易出错，显然应该重点测试；一个类如果出错，这个错误对系统造成的影响比较大（影响的类比较多），我们也应该重点测试。基于这两点思考，本章从两个方面度量类在测试时受关注程度的大小，即：(1) 类错误倾向指数（Bug Proneness Index of Classes, $BPIC$）；(2) 类错误传播率（Bug Propagation Ratio of Classes, $BPRC$），并提出类测试关注度指标（Testing Attention of Classes, TAC）度量类在测试时受关注的程度，其计算公式如下：

$$TAC_i = \alpha \times BPIC_i + \beta \times BPRC_i \tag{6.4}$$

其中：TAC_i 表示第 i 个类的 TAC；$BPIC_i$ 表示第 i 个类的 $BPIC$；$BPRC_i$ 表示第 i 个类的 $BPRC$；α, β 是相应度量的权值，且 $0 \leq \alpha, \beta \leq 1$, $\alpha + \beta = 1$。通过调节 α 和 β 的值，我们可以满足不同的测试需求。如果想重点测试容易出错的类，则可以将 α 设置的相对大点；如果想重点测试那些对系统危害大的节点，则可以将 β 设置的相对大点。本章中我们令 $\alpha = \beta = 0.5$。

下面我们将具体介绍 $BPIC$ 和 $BPRC$ 的计算方法。

6.3.3.1 类错误倾向指数

$BPIC$ 用于描述修改某个类时可能引入错误的概率。现有的研究工作主要从需求的复杂性[221]和系统源代码的复杂性[36, 37, 238, 239]角度来评估类引入错误的可能性大小。本章的研究对象主要是开源的 OO 软件系统，要获得这些软件系统的需求文档存在很大的困难。因此，本章主要从软件源代码的复杂性角度来计算各个类的 $BPIC$ 值。

S. R. Chidamber 和 C. F. Kemerer在 [184] 中提出称为 CK 度量组的 OO 软件度量体系，它从继承（DIT、NOC），类之间的耦合（RFC、CBO）和类自身复杂性（WMC、$LCOM$）等角度, 来度量 OO 软件的复杂性。研究表明，CK 度量组中大部分的度量指标能够很好的预测容易出错的类[36, 37, 238, 239]。J. Xu 等在 [245] 中对现有的研究工作（主要是 IEEE Transaction on Software Engineering 上发表的工作）作了系统

的总结和比较，发现 $SLOC$、WMC、CBO 和 RFC 是广泛认可地能够较好地预测类的错误倾向性的复杂性度量指标。本章将使用这 4 个度量指标来计算类的 $BPIC$。这 4 个度量的定义如下[184]：

（1）$SLOC$（Source Lines Of Code）：$SLOC$ 是源代码行数。空行和注释行都忽略。

（2）WMC（Weighted Methods per Class）：WMC 是一个特定类中每个方法的复杂性的总和，本章我们采用它最简单的形式，即设每个方法的复杂性为一个单位，则 WMC 为该特定类中所包含的方法总个数。

（3）CBO（Coupling Between Object）：CBO 是和该特定类有耦合关系的类的总个数。

（4）RFC（Response For a Class）：RFC 是类中某种特定类型方法的总个数。该种方法的定义是，当这个类的一个对象接收到了外部的一个消息，那些可能被执行的方法。

本章中，$BPIC$ 值可以根据下式计算：

$$BPIC_i = \chi\frac{SLOC_i}{SLOC_{sum}} + \delta\frac{WMC_i}{WMC_{sum}} + \varepsilon\frac{CBO_i}{CBO_{sum}} + \phi\frac{RFC_i}{RFC_{sum}} \tag{6.5}$$

其中: $BPIC_i$、$SLOC_i$、WMC_i、CBO_i 和 RFC_i 分别是类 i 的 $BPIC$、$SLOC$、WMC、CBO 和 RFC。$SLOC_{sum}$、WMC_{sum}、CBO_{sum} 和 RFC_{sum} 分别是系统中所有类的 $SLOC$、WMC、CBO 和 RFC 之和。χ、δ、ε 和 ϕ 是相应度量指标的权值，且满足 $0 \leq \chi, \delta, \varepsilon, \phi \leq 1$, $\chi + \delta + \varepsilon + \phi = 1$。它们的值是根据相应度量指标在类错误倾向性预测中的有效性设定的，即：若某个度量指标越能有效的预测类的错误倾向性，则它相应的权值就越大。度量指标预测类错误倾向性的有效性可以根据前几个版本的历史错误数据和度量指标间的相关性分析得到，方法见 [222]，这里不再赘述。本章中我们令 $\chi = \delta = \varepsilon = \phi = 1/4$。

6.3.3.2 类错误传播率

软件中某一实体出错，与之有（直接或间接）关系的其它实体就可能受到影响（引发新的错误，甚至触发严重的故障），这种现象被称为“涟漪效应”[167, 169]。当软件系统用网络模型表示的时候，这种“涟漪效应”表现为一个节点的错误通过网络上的有向边进行传播和扩散。如前所述，错误产生在不同的类节点，其扩散影响到的其它类节点数是不同的。因此，如何度量不同类节点错误扩散能力的这种差异呢？

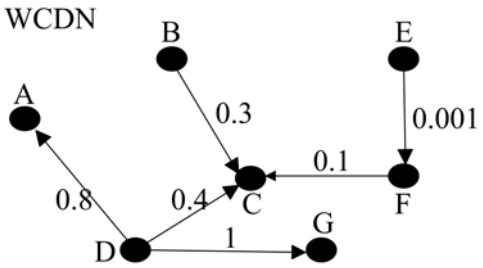


图 6.5　WCDN 例子

如果仅从与错误节点有（直接或间接）关系的节点的数量来度量错误节点的影响力，如图 6.5 所示，我们可以得到受节点 G 影响的节点集合是 $\{G, D\}$，受节点 F 影响的节点集合是 $\{F, E\}$，则节点 G 和 F 的影响力是一样的，但是这种方法忽略了边上权值的差异，即：F 的错误只以 0.001 的概率传给节点 E，而节点 G 的错误却以 100% 传给 D。若我们以节点错误传播概率之和来计算，A 到 D 的错误传播概率是 0.8，C 到 B、D 和 F 的传播概率和也是 0.8，难以区分它们可能影响点的数量，A 影响两个节点($\{A, D\}$)，C 影响四个节点($\{C, B, D, F\}$)。因此，我们必须引入一种方法，既考虑了边上权值的差异，同时又考虑了影响节点数的差异。张莉等在 [180] 提出了一种基于仿真技术的指标计算方法，用于计算软件的稳定性。这种方法正满足以上两点要求。因此，本章将借鉴该方法求解错误类节点的 $BPRC$。

定义6：类的概率传播域（Probability Propagation Field of Classes, $PPFC$）

一个特定 WCDN 中，类节点 i 的 $PPFC$，即 $PPFC(i)$ 定义节点 i 发生错误时可能影响到的节点集合，这个值是在一次特定的仿真中，假设节点 i 包含错误，找出受到该错误影响的节点集合。

定义7：类错误传播率（Bug Propagation Ratio of Classes, $BPRC$）

节点 i 的类错误传播率 $BPRC(i)$ 定义为：

$$BPRC(i) = \frac{\sum_{j=1}^{\mathrm{ST}} |PPFC(i)_j|}{\mathrm{ST} \times |N^c|} \tag{6.6}$$

其中：$PPFC(i)_j$ 表示对节点 i 的第 j 次仿真求得的 $PPFC(i)$，ST 表示仿真的次数。因此，$BPRC(i)$ 表示对节点 i 做 ST 次仿真，求 ST 次仿真 $PPFC(i)$ 的平均值占类节点总数的比率。我们提出算法 6.2 来计算任一节点 i 的 $BPRC(i)$。

通过算法 6.2，我们可以在 $O(|N^c|)$ 时间内得到节点 i 的 $BPRC(i)$。本算法与 [180] 中的算法具有一定的相似性，但又不完全相同，区别如下：

（1）[180] 中仿真算法的目的是求系统的稳定性，本章主要是求单个类的 $BPRC$；

算法 6.2: $BPRC$ 求解的仿真算法

输入:

软件的 WCDN，仿真次数 ST，当前仿真次数 $ct = 1$，欲求 $BPRC$ 的节点 i；

输出:

$BPRC(i)$；

1: 将标志节点是否是错误节点的状态数组 $bBug[]$ 初始化为 false，记录每次仿真错误节点数的数组 $BugNum[]$ 初始化为 0；

2: 设置节点 i 的错误状态 $bBug[i]$ = true，并将节点 i 加入错误队列 BugNodeQueue 中；

3: 选择 BugNodeQueue 中的队首节点 v_s，将所有直接依赖于 v_s（即在 WCDN 中存在指向 v_s 的边的节点）且其 $bBug$ 为 false 的节点加入集合 S；

4: 对集合 S 中的每个节点 v_t，随机产生概率 p_t，如果 p_t 小于边 $< v_t, v_s >$ 上的概率，即 p_t ¡ $M_p^c(v_s, v_t)$，则将其 $bBug$ 设置为 true，并加入到 BugNodeQueue 队列尾；

5: 删除队首节点 v_s，清空集合 S；

6: 反复执行步骤 $3 \sim 5$，直到 BugNodeQueue 为空，计算错误节点总数（即 $bBug[i]$ = true, i = 1, 2, ..., $|N^c|$ 的个数），并存入 $BugNum[ct]$；

7: 设置 $ct = ct + 1$；若 ct ¡ ST，反复执行 $1 \sim 6$，否则仿真停止，输出 $BPRC(i) = \frac{\sum_{j=1}^{ST} |PPFC(i)_j|}{ST \times |N^c|}$

（2）[180] 中仿真算法基于的类网络其边权是假设的，且所有边权相同，而本章 WCDN 的边权是根据 WFDN 构建的，不同边其边权存在差异；

（3）[180] 中仿真算法每一次仿真都随机选择一个节点，求其影响范围，然后求平均值，而本章中这个节点是确定的，是针对一个欲求 $BPRC$ 的节点做仿真的。

由于包含错误的类节点是以一定概率影响直接依赖于该节点的相邻节点的，存在一定的不确定性，仿真的初期 $BPRC$ 的值可能会有所不同，但随着仿真次数 ST的增加，$BPRC$ 会逐步收敛，最终保持稳定。在本章中我们令 ST = 10,000。$BPRC$ 收敛性的分析与 [180] 同，这里不在赘述。

6.3.4 测试用例优先级计算

测试用例的优先级（Priority of Test Cases, PTC）是通过其覆盖的类的 TAC 来计

算的，计算公式如下：

$$PTC_i = \sum_{j=1}^{m} TAC_j \tag{6.7}$$

其中：PTC_i 表示第 i 个测试用例的 PTC 值，m 表示第 i 个测试用例覆盖的类的个数，TAC_i 表示测试用例覆盖的类中第 i 个类的 TAC 值。开源 OO 软件系统在其主页和 CVS 中一般都不提供测试用例的覆盖信息。本章使用 djUnit 来获得测试用例的覆盖信息。djUnit 是 dgic 公司开发的免费 Eclipse 插件，在标准 JUnit 测试框架的基础上实现了对 Virtual Mork 技术的支持，它同样支持 JUnit 的测试用例的运行，同时可以为每个测试用例提供 html 格式的测试覆盖报告。本章通过解析 html 格式的测试覆盖报告来获得每个测试用例覆盖了哪些类的信息。在获得每个测试用例的 PTC 值后，我们按照 PTC 值降序排列测试用例（PTC 相等的测试用例按其编写时的相对顺序排列），从而完成测试用例的排序。

6.4 实验设计

本节将通过实例来验证本章方法的有效性。在研究过程中我们开发了可以自动提取 OO 软件 WCDN、计算测试用例覆盖信息、测试用例优先级，并最终排序测试用例的工具，在此基础上设计和进行对比实验。本节将对实验数据进行描述和分析。

6.4.1 系统简介

实验选取开源 OO 软件 Xml-security 和 JTopas 作为研究对象。其中：Xml-security 项目是为 XML 安全标准（XML 数字签名语法和处理规则、XML 加密语法和处理规则）提供 Java 实现的开源项目。JTopas 是用于解析文本数据的 Java 类库，这些数据可以来自包含一些注释的简单配置文件，如：HTML、XML、RTF stream 和来自程序语言的源代码等。这两个 OO 软件系统常常作为基准软件被广泛运用于各种测试用例排序技术研究中[235, 236, 237]。

我们从 SIR[246]（Software-artifact Infrastructure Repository）上获得了 Xml-security（3 个版本 $v_0 \sim v_2$）和 JTopas（4 个版本 $v_0 \sim v_3$）的源代码（正确版本和预先植入错误的版本）以及整套 JUnit 测试用例。其中：v_0 版本只有正确版本，其它版本都有两个版本，一个正确版本、一个预先植入错误的版本。错误版本（Xml-security 中 $i = 1, 2$；JTopas 中 $i = 1, 2, 3$）的测试用例由两个部分构成：一部分复用了 v_{i-1} 版本的测试

用例，一部分是新增的测试用例。由于新增的测试用例是 v_{i-1} 正确版本没有的，我们无法获得其覆盖路径信息。本章中，新增测试用例我们令其 PTC 为 0。我们首先在 v_i 正确版本构建 WCDN、计算类测试关注度 TAC、收集测试用例覆盖信息、计算测试用例优先级 PTC，并对测试用例进行排序，然后将错误版本中的测试用例（v_i 复用 v_{i-1} 的那些测试用例）按照这个顺序来执行。新增的测试用例在这些复用的测试用例执行完后再执行，并按它们编写时的相对顺序执行。表 6.1 列出的是 v_0 正确版本的和其它错误版本的一些统计信息。

6.4.2 实验结果与分析

图 6.6 显示的是 JTopas-0.5.1 的 WFDN 和 WCDN，其中节点旁的标注是该节点代表的特征（WFDN）或类（WCDN）的名字，边上的权值是错误在特征（WFDN）或类（WCDN）间传播的概率。为了显示上的清晰，WFDN 中我们只显示了部分特征的名字。我们构造了 Xml-security 和 JTopas 所有版本的 WFDN 和 WCDN，由于篇幅关系，我们不再一一列出。

目前使用最广泛的评价测试用例排序技术有效性的指标是由 G. Rothermel 等人提出的 $APFD$（weighted Average of the Percentage of Faults Detected）[218]，但是 $APFD$ 在评价测试用例排序技术有效性时忽略了测试用例的资源耗费和测试用例检测到的错误的严重程度，因此 G. Rothermel 等人对 $APFD$ 进行了改进，提出了 $APFD_c$ 度量指标[229]。本章使用 $BPRC$ 评价类的错误对软件的影响程度，是从软件结构角度对错误严重性的一种度量。因此，本章采用 $APFD_c$ 评价测试用例的有效性。$APFD$ 的相关定义不再赘述。

假设 T 是一个测试用例套件，包含 n 个测试用例，每个测试用例相应的资源耗费是 $t_1, t_2, ..., t_n$；F 是由 T 检测到的 m 个错误的集合，每个错误相应的严重程度用 $f_1, f_2, ..., f_m$ 表示；T' 是 T 的一个排列（代表测试用例套件的一种执行可能）；TF_i 表示 T' 中第一个检测出错误 i 的测试用例在 T' 的位序，则 T' 的 $APFD_c$ 值的计算公式如下：

$$APFD_c = \frac{\sum_{i=1}^{m}(f_i \times (\sum_{j=TF_i}^{n} t_j - \frac{1}{2}t_{TF_i}))}{\sum_{i=1}^{n} t_i \times \sum_{i=1}^{m} f_i} \tag{6.8}$$

如果每个测试用例的资源耗费相等，且测试用例发现的错误的严重性程度也相同，即：$t_1 = t_2 = ... = t_n$，$f_1 = f_2 = ... = f_m$，那么 $APFD_c$ 就转换成了 $APFD$。

表 6.1　实验 OO 软件的统计信息

软件系统	SIR 版本	代码行数	类个数	特征数	植入错误数	测试用例数	v_i 复用 v_{i-1} 测试用例数
Xml-security-1.0.2	v_0	11,691	184	2,140	0	16	0
Xml-security-1.0.4	v_1	12,329	198	2,236	20	15	13
Xml-security-1.0.5D2	v_2	12,562	198	2,245	14	15	15
JTopas-0.3	v_0	1,312	23	389	0	7	0
JTopas-0.4	v_1	1,351	25	410	10	10	7
JTopas-0.5.1	v_2	1,330	25	400	12	11	10
JTopas-0.6	v_3	3,680	65	1,035	16	18	11

注　1)软件名后的数字是其发布的版本号，SIR 版本是 SIR 库中使用的版本号

2)代码行是源代码函数，不包括注释和空行

3)类个数包括两个部分：类数和接口数

4)特征是方法和属性的总称

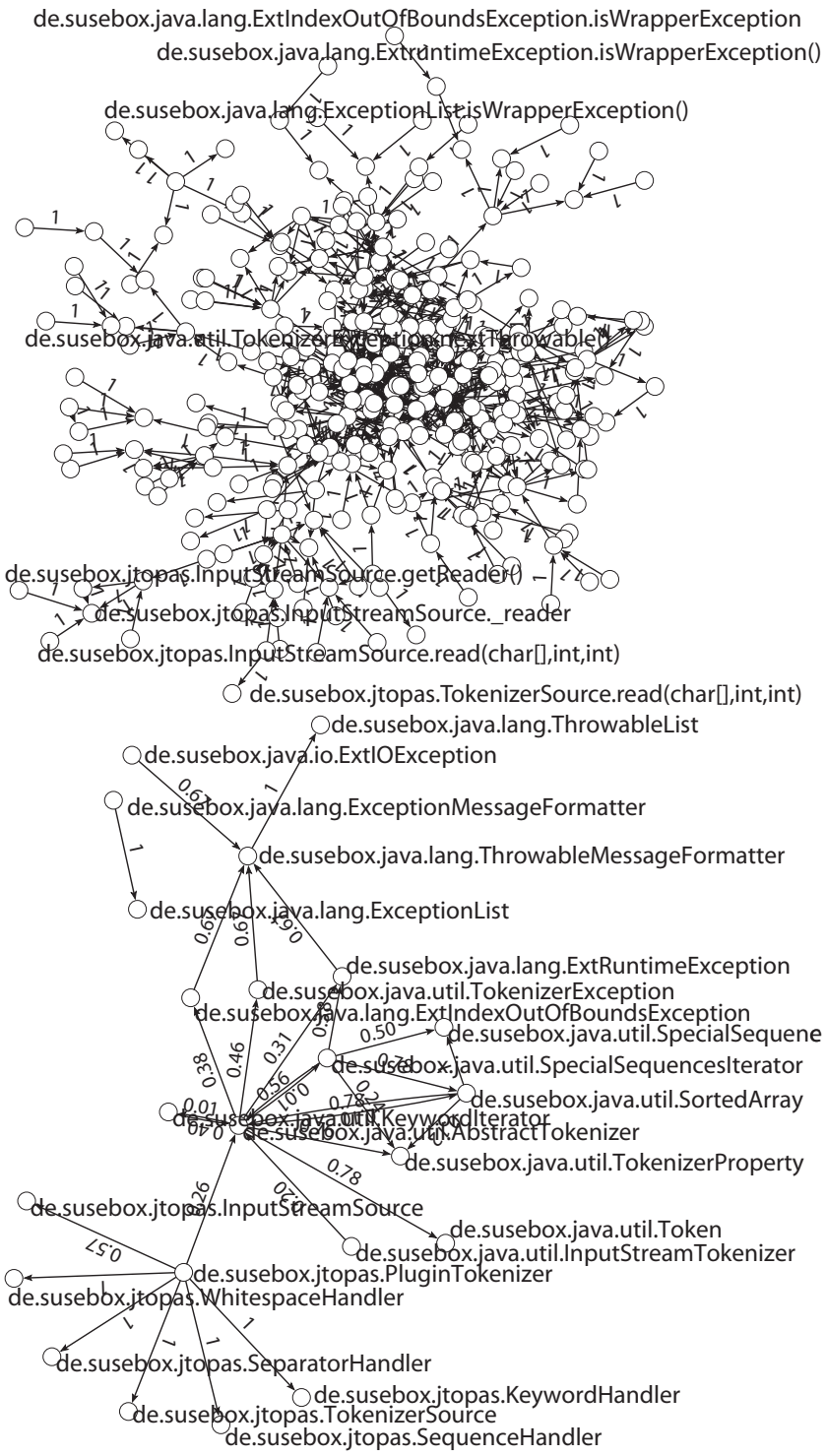


图 6.6　JTopas-0.5.1 的 WFDN（上）和 WCDN（下）

本章不考虑测试用例的资源耗费（开源 OO 软件测试用例的资源耗费难以获得），仅考虑测试用例检测出错误对系统破坏的严重程度。在实验的过程中，我们假设 $t_1 = t_2 = ... = t_n = 1$，$f_k = BPRC_k$, $k = 1, 2, ..., m$。某种排序技术的 $APFD_c$ 值越高，说明这个排序技术在提高严重错误发现速率上的效果越好。

我们将 TCP-WCDN 与已有的七种排序技术进行比较，表 6.2 对这些排序技术做了简要说明。这些排序技术来自文献 [216, 218, 235, 236, 237]。表 6.3 给出了各排序技术排序结果的 $APFD_c$ 值，其中 Random 是 50 次运行结果的平均值。我们可以发现，TCP-WCDN 排序后的测试用例在严重错误发现速率上的效果最接近 Optimal。与其它六种常用方法相比，TCP-WCDN 的 $APFD_c$ 平均（v_0 版本外其余 5 个版本 $APFD_c$ 的平均值）都有不同程度的提高。TCP-WCDN 的 $APFD_c$ 比 Untreated 的 $APFD_c$ 平均提高了 55.1%，比 Random 的 $APFD_c$ 平均提高了 26.1%，比 C_Cov 的 $APFD_c$ 平均提高了 6.3%，比 C_A_Cov 的 $APFD_c$ 平均提高了 10.1%，比 Chg_Cov 的 $APFD_c$ 平均提高了 15.5%，比 Chg_A_Cov 的 $APFD_c$ 平均提高了 10.1%。TCP-WCDN 没有利用反馈信息，也没有利用软件的变更信息，却比使用了反馈信息的排序技术 C_A_Cov 和 Chg_A_Cov，和使用了变更信息的排序技术 Chg_Cov 和 Chg_A_Cov 获得了更好的效果。这说明考虑软件复杂性和结构特征的测试用例排序技术确实能有效的提高发现严重错误的能力。

另外，我们可以发现在 Xml-security-1.0.4 版本中，TCP-WCDN、C_Cov、Chg_Cov 和 Chg_A_Cov 都达到了最优 0.96。这是为什么呢？通过查看源代码和测试用例的覆盖信息，我们发现在 Xml-security-1.0.4 植入的 20 个错误中，只有 3 个可以被测试用例检测出来（即 CHP_AK_1、 CNC_AK_1、 XSI_AK_1），而这 3 个错误都可以由测试用例 org.apache.xml.security.test.interop.IAIKTest 同时检测出来，而这个测试用例在 TCP-WCDN、C_Cov、Chg_Cov 和 Chg_A_Cov 这 4 种排序技术中都排在第一位，所以都达到了最优。

6.5 本章小节

本章从软件复杂性和内部静态结构出发，对 OO 软件的回归测试用例排序技术进行了研究。提出加权类依赖网络模型抽象类粒度 OO 软件系统，通过分析类的复杂性来度量类的错误倾向性，通过仿真方法评价类产生错误后对系统的影响大小，并在此基础上提出类测试关注度（TAC）衡量类的测试重要性。最后根据测试用例覆盖的类的 TAC 总和对测试用例进行排序。开源 OO 软件的实例研究表明，TCP-WCDN 方法在严重错误的检出速率上具有比其它方法更好的性能。软件复杂性和软件结构一直被认为与软件的外部质量特征（可维护性、可靠性等）具有密切联系，本章的主要贡献在于将软件复杂性和软件结构与软件回归测试用例排序结合起来，目标是提高软件回归测试的效率。

表 6.2　测试用例优先级排序方法

助记符	方法名	方法说明
Untreated	No prioritization	按照测试用例编写时的顺序执行
Random	Random prioritization	对测试套件中的测试用例随机进行排序，然后按照排序后的顺序执行测试用例
Optimal	Optimal prioritization	按照最大化严重错误发现速率排列和执行测试用例
C_Cov	Class coverage	按照测试用例覆盖的类的数量降序排列并执行测试用例
C_A_Cov	Additional Class Coverage	按照测试用例对未覆盖（考虑了反馈）类的覆盖量排列和执行测试用例
Chg_Cov	Change Coverage	按照测试用例对变更代码的覆盖量排列和执行测试用例
Chg_A_Cov	Additional Change Coverage	按照测试用例对未覆盖（考虑了反馈）的变更代码的覆盖量排列和执行测试用例

表 6.3　各种排序技术 $APFD_c$ 值表

软件系统	SIR 版本	TCP-WCDN	Optimal	Untreated	Random	C_Cov	C_A_Cov	Chg_Cov	Chg_A_Cov
Xml-security-1.0.2	v_0	0	0	0	0	0	0	0	0
Xml-security-1.0.4	v_1	0.96	0.96	0.40	0.72	0.96	0.96	0.96	0.96
Xml-security-1.0.5D2	v_2	0.95	0.96	0.86	0.79	0.95	0.86	0.94	0.95
JTopas-0.3	v_0	0	0	0	0	0	0	0	0
JTopas-0.4	v_1	0.5	0.9	0.3	0.45	0.3	0.3	0.3	0.3
JTopas-0.5.1	v_2	0.95	0.95	0.41	0.63	0.95	0.95	0.77	0.86
JTopas-0.6	v_3	0.89	0.96	0.77	0.78	0.84	0.79	0.71	0.77

下一步作者将考虑如下研究工作：

（1）Xml-security 和 JTopas 尽管广泛运用于各种研究工作中，但是其规模还不够大，错误数量也不够多。因此下一步拟用更多不同领域的大规模开源 OO 软件验证 TCP-WCDN 方法的有有效性。

（2）研究表明，软件内的错误是关联的[247]。下一步拟从软件结构角度研究错误的关联关系，并将其引入 TCP-WCDN。

（3）TCP-WCDN 仅考虑了类中错误的严重性，忽略了测试用例的资源耗费，下一步拟将测试用例的资源耗费也纳入测试用例优先级排序。

（4）文中求 TAC 和 $BPIC$ 值时用到的权重参数，并没给出其严格的设置方法，下一步拟用机器学习的方法给出它们的设置方法。

（5）拟将 TCP-WCDN 实现为集成开发环境 Eclipse 平台的插件，简化测试用例排序过程，提高回归测试效率。

本章实验部分所有数据，包括实验的软件（正确和错误版本）、所有的 WFDN 及 WCDN、算法 6.1 和 6.2 程序、实验结果数据等都可以从 [164] 处下载。

第七章　总结与展望

7.1　论文总结

应用领域深度和广度上的不断拓展以及人们对软件功能和质量需求的不断提高，使得软件规模激增，软件质量无法得到保障。因此，如何认识、度量、管理、控制乃至降低软件复杂性，是软件工程学亟待解决的一个重要问题。软件结构被认为是影响软件复杂性的重要因素之一。因此，要对软件系统的复杂性进行深入、细致的研究，就必须对软件的结构信息进行合理的描述和有效的量化。

传统的结构度量方法（面向过程/对象度量方法）更多考虑的是软件元素属性的度量，即一个类中有多少方法、一个方法有多少行代码等，尽管也有部分度量（CBO、DIT 等）考虑了元素的交互，但是对于软件结构本身包含的复杂信息还没有充分地挖掘。由于缺少相应的方法和理论，人们很少从整体的角度来研究软件的结构及其演化特性，导致对软件的本质缺乏清晰的认识。

复杂网络可以描述软件系统的骨架，从较高层次描述系统的结构，为我们研究系统结构及其动力学行为提供了有力的工具。近年来，一些研究者将复杂网络相关理论引入软件工程中，构建软件网络模型，用于分析软件结构特征及其动力学行为。它强调从整体上认识、理解和控制系统，而非关注局部，旨在理解软件的结构特性，为软件开发和维护提供支持。

但是，软件网络的研究工作目前仍处于起步阶段。纵观软件网络 8 年多的发展，软件网络的研究尽管已经取得了一些可喜的成果，但是仍存在下面的一些问题：

- 缺乏从不同粒度和演化的角度来分析和理解软件的组织结构、结构的演变和由此产生的行为特征；
- 缺乏软件整体结构特性与软件质量（稳定性、可靠性、可维护性等）之间关系的认识；
- 缺乏软件结构度量如何指导工程实践（软件重构、软件测试等）的研究。

针对上述问题，本文基于软件网络结构，采取软件工程和复杂网络相关理论相结合的研究方法，以分析软件结构特征为着眼点，以理解软件的结构特性，为软件开发和维护提供支持为目标，深入研究了软件不同粒度的结构特征及其演化规律、软件结构稳定

性分析及结构优选、面向对象软件类重构方法、面向对象软件回归测试用例排序方法。本文具体的研究内容和取得的成果包括：

（1）软件多粒度演化分析

针对现有软件演化分析中的不足，提出了软件的多粒度模型，从方法、类和包三个粒度抽象面向对象软件结构，并采用复杂网络的理论与方法，从时间（多版本演化）和空间（方法、类和包三个粒度）两个角度系统的研究软件系统结构特性及其演化规律。真实面向对象软件的实证研究表明：各个粒度软件网络规模（节点数、边数）增长呈“超线性”增长特点，但是增长速度从方法粒度到包粒度不断下降；各个粒度软件网络平均度呈“弱加速”增长特点；各个粒度软件网络都是“小世界”和“无标度”网络，并且度分布的幂指数在一个很小的范围内波动；类和包粒度软件网络是“弱异配”网络，而方法粒度上的软件网络是“弱同配”网络；类和包粒度软件网络表现出明显的“富人俱乐部”现象，而方法粒度的不明显。此外，通过跟踪软件模块度的变化率，我们提出了一种确定软件重构时间点的方法。这些规律的发现可以辅助理解软件系统设计的演化过程及趋势，并指导软件项目管理、成本预测以及软件过程中的软件测试、软件维护等。同时对于构建软件演化模型也具有重要意义。

（2）软件稳定性评估方法及结构优选

软件稳定性是软件设计的核心，对软件维护和质量的评估都有重要影响。但是软件变更的需求存在不确定性，使得如何认识、理解、分析和控制软件稳定性成为软件工程的一个难题。针对以往软件结构稳定性评估方法存在的问题，本文将软件稳定性分析的粒度下降到了特征（方法和属性）粒度，构建了面向对象软件的加权特征依赖网络模型 WFDN 用于抽象软件结构。同时考虑了软件多点变更的问题，提出了一种仿真方法以分析变更在 WFDN 上的传播过程，并定义软件稳定性度量指标 SoS 评价软件的稳定性。我们使用 Weyuker 准则验证了 SoS 的有效性。实例研究也表明：SoS 可以正确的从具有相同功能不同结构的软件中选择结构稳定性好的软件，实现结构优选，这对决策者作出正确的决策具有指导意义；随着软件元素间耦合的增强，软件的稳定性不断下降。因此，在实际的软件开发中，我们应该尽量降低元素间的耦合，以提高软件质量。这一结论为软件结构设计和质量评估提供了理论依据。

（3）面向对象软件类重构

软件重构是一种提高软件结构设计和可维护性的有效方式。在本文中，我们将复杂网络社区发现算法应用于面向对象软件的类重构，提出了面向对象软件类重构的 CR-CDSN 方法，用于识别方法移动的重构。该方法的基本思想是软件工程实践中倡导的

“高内聚，低耦合”的设计原则，它首先将面向对象软件系统在特征粒度抽象成两类软件网络，即：AMN 和 MMN；然后借鉴复杂网络社区发现算法，在充分尊重设计模式的前提下提出了一种有导向的社区发现算法 GCDA 用于识别 MMN 中的重构点；同时，提出节点走查的方式来识别 AMN 中的重构点。本文的方法取得了与其它方法类似的结果，但是效率比其它的方法要高。

本文提出的 CR-CDSN 方法可以为开发者实施重构工作提供指导。可以将本文提出的方法实现为集成开发环境（如 Eclipse 等）的插件或开发独立的软件工具，用于实际的软件开发中，具有较好的实用前景。

（4）面向对象软件回归测试用例排序方法

本文从软件复杂性和内部静态结构出发，对 OO 软件的回归测试用例排序技术进行了研究。提出加权类依赖网络模型抽象类粒度 OO 软件系统，通过分析类的复杂性来度量类的错误倾向性，通过仿真方法评价类产生错误后对系统的影响大小，并在此基础上提出类测试关注度（TAC）衡量类的测试重要性。最后根据测试用例覆盖的类的 TAC 总和对测试用例进行排序。开源 OO 软件的实例研究表明，TCP-WCDN 方法在严重错误的检出速率上具有比其它方法更好的性能。软件复杂性和软件结构一直被认为与软件的外部质量特征（可维护性、可靠性等）具有密切联系，本文的主要贡献在于将软件复杂性和软件结构与软件回归测试用例排序结合起来，目标是提高软件回归测试的效率。

这四个内容围绕软件静态结构分析，从表征、分析、度量和应用四个方面来对面向对象软件静态结构进行分析，彼此之间是紧密关联的。结构的软件网络表征是分析、度量和应用的基础，我们的研究工作都是基于软件的网络模型的。分析、度量和应用是结构分析的三个方面（如上文所述）。分析是度量和应用基础，通过分析可以发现结构中存在的特性和潜在问题，从而为度量和应用研究提供内容。此外，度量是应用的前提，我们是在软件结构度量的基础上来解决软件工程中存在的结构优选、重构和测试问题的。

同时，多粒度演化分析的结果对软件过程的设计、构造、测试和维护阶段的工作以及软件管理都有指导意义。基于稳定性分析的结构优选、软件类重构技术和回归测试用例排序方法分别针对维护阶段、构造阶段和测试阶段的问题提供了相应的解决方案。可以说，基于软件网络的软件结构分析方法对于解决软件工程中，特别是解决软件过程各阶段中的问题都具有重要的意义。

7.2 存在的问题和进一步工作

本文借鉴复杂网络的理论和方法，应用软件网络对软件系统结构进行研究，取得了一些初步的成果。进一步的工作包括：

（1）扩大研究范围，构建各种形式的软件网络

目前的研究主要关注 Java 语言编写的面向对象软件，其它面向对象编程语言，如 C++、C#.NET、Visual Basic.NET 编写的软件，甚至结构化程序的结构特性也是一个值得研究的问题。对这些软件的分析，有助于我们分析不同类型软件结构特性的异同。此外，目前软件网络研究主要关注软件的代码，构建的网络模型中的节点类型比较简单，主要是将方法、类、包抽象为节点，而软件系统中还有其它的概念，如角色、消息、构建、活动、事件等，这些元素也可以考虑在内。因此，我们应该构建包含各种因素的软件网络模型。

本文主要关注软件的静态结构，没有考虑软件动态运行时的动态调用关系，这种动态结构的研究也能给我们提供一些软件设计和开发的意见和建议。

软件开发是一项人工参与的复杂的系统工程，人的可靠性直接影响到软件质量。软件网络研究也应该关注人的因素。因此，我们也可以从软件的开发过程出发构建开发者网络，运用复杂网络理论知识，分析开发者网络，探索开发者在软件开发过程中的协作行为、活跃程度、贡献价值，并给予综合评价，为管理者及其他开发人员的工作提供决策参考。

（2）软件演化建模研究

我们在第 3 章的研究中发现了软件的“超线性”增长、“弱加速”增长等特点。现有的软件演化模型还不能很好的解释这些特点的产生机理。因此我们有必要研究新的软件演化模型，对不断发现的新特性进行解释，并用于指导软件项目管理、成本预测等。

（3）基于软件结构度量的应用研究

本文基于软件结构度量做了基于软件稳定性分析的结构优选、类重构和回归测试用例排序方面的初步研究，在接下来的工作中，我们还可以在以下几个方面进行深入研究：

- 软件需求对软件的变更有很大影响，本文软件稳定性的分析没有考虑需求变更的影响，所以在将来的研究中，我们将分析软件需求的变化与软件元素变化的关系，构建更加合理的软件变更影响分析模型，提高软件稳定性分析的精度，为软件结构优选、软件变更决策制定等提供支持。

- 面向对象软件重构有多种类型，如提炼类、将类内联化、移动/下置/提升类属性

等，这些都还没有在我们目前的方法内实现。因此，我们有必要获得更多的结构信息，来对软件的结构进行重构。

- 在本文软件回归测试用例排序研究中，我们仅考虑了类中错误的严重性，忽略了测试用例的资源耗费。在下一步的研究中，我们拟将测试用例的资源耗费也纳入测试用例优先级排序中。同时，研究基于结构分析的软件集成测试等。

- 基于软件网络的软件缺陷结构挖掘。构建软件的网络模型，并对网络的部分特征进行度量，分析软件度量与软件开发原则之间的关系，分析软件结构上的缺陷，并预测软件错误的分布（错误的倾向性、错误数量等）。

（4）工具开发

进一步完善面向对象软件复杂网络特性分析工具。研究用更加直观、便于用户理解的方式显示大规模软件结构分析和优化的结果。

参 考 文 献

[1] L. Northrop, P. Feiler, R. P. Gabriel, et al., *Ultra-Large-Scale Systems: The Software Challenge of the Future*, `www.sei.cmu.edu/library/assets/ULS_Book20062.pdf`, 2006.

[2] 钱学森, *论系统工程*, 长沙: 湖南科学技术出版社, 1982.

[3] T. DeMarco, *Controlling Software Projects: Management, Measurement, and Estimates*, NJ: Prentice Hall PTR, 1986.

[4] 郑人杰, 殷人昆, 陶永雷, 实用软件工程(第二版), 北京: 清华大学出版社, 2001.

[5] V. Maraia, *The Build Master: Microsoft's Software Configuration Management Best Practices*, NY: Addison-Wesley Professional, 2005.

[6] F. P. Brooks, "No Silver Bullet: Essence and Accidents of SoftwareEngineering", *IEEE Computer*, 1987, 20(4): 10-19.

[7] The Standish Group International Inc., *CHAOS Summary 2009*, `www.standishgroup.com`, Oc. 3, 2009.

[8] W. Davis, *System Analysis and Design: a Structured Approach*, NY: Addison Wesley, 1983.

[9] H. Zuse, *Software Complexity—Measures and Methods*, Berlin: Walter de Gruyter & Co, 1991.

[10] N. E. Fenton, Y. Lizuka, and R. W. Whitty, *Software Quality Assurance and Measurement*, London: Internationla Thomson Computer Press, 1996.

[11] N. E. Fenton, *Software Metrics: a Rigorous Approach*, London: Chapman and Hall, 1991.

[12] R. Shatnawi, W. Li, J. Swain, et al., "Finding Software Metrics Threshold Values using ROC Curves", *Journal of Software Maintenance and Evolution: Research and Practice*, 2010, 22(1): 1-16.

[13] Y. Zhou, B. Xu, and H. Leung, "On the Ability of Complexity Metrics to Predict Fault-Prone Classes in Object-Oriented Systems", *Journal of Systems and Software*, 2010, 83(4): 660-674.

[14] 李兵, 马于涛, 刘婧, 等, "软件系统的复杂网络研究进展", *力学进展*, 2008, 38(6): 805-814.

[15] N. E. Fenton and S. L. Pfleeger, *Software Metrics: a Rigourous and Practical Approach, 2nd Edition*, London: Internationla Thomson Computer Press, 1996.

[16] C. Bill, "Techies as Non-Technological Factors in Software Engineering?", In *Proc. of 13th International Conference on Software Engineering*, Austin, TX, USA, 1991, pp. 147-148.

[17] R. J. Rubey and R. D. Hartwick, "Quantitative Measurement of Program Quality", In *Proc. of the 23rd ACM National Conference*, New York, NY, USA, 1968, pp. 671-677.

[18] A. J. Albrecht, "Measuring Applications Development Productivity", In *Proc. of the IBM Application Development Symposium*, Monterey, CA, 1979, pp. 83-92.

[19] A. J. Albrecht and J. E. Gaffney, "Software Function, Source Line of Code, and Development Effort Prediction: a Software Science Validation", *IEEE Transactions on Software Engineering*, 1983, 9(6): 639-648.

[20] M. Arnold and P. Pedross, "Software Size Measurement and Productivity Rating in a Large-Scale Software Development Department", In *Proc. of 1998 International Conference on Software Engineering*, Kyoto, 1998, pp. 503-506.

[21] L. Briand, S. Morasca, and V. R. Basili, *Defining and Validating High-Level Design Metrics*, Tech. Rep. CS TR-3301, University of Maryland, 1994.

[22] J. Babsiya and C. G. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment", *IEEE Transactions on Software Engineering*, 2002, 28(1): 4-17.

[23] R. Chidamber and F. Kemerer, "A Metrics Suite for Object-Oriented Design", *IEEE Transactions on Software Engineering*, 1994, 20(6): 476-493.

[24] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics: a Practical Guide*, NJ: Prentice Hall PTR, 1994.

[25] F. Brito and E. Abreu, "MOOD-Metrics for Object-Oriented Design", In *Proc. of OOPSLA'94 Workshop on Pragmatic and Theretical Directions in Object-Oriented Software Metrics*, Portland, 1994.

[26] N. E. Fenton and M. Neil, "Software Metrics: Successes, Failures and New Directions", *Journal of Systems and Software*, 2000, 47(2-3): 149-157.

[27] N. E. Fenton and M. Neil, "Software Metrics: Roadmap", In *Proc. of 2000 International Conference on Software Engineering (The track of Future of Software Engineering)*, Ireland, 2000, pp. 357-370.

[28] J. E. Simith, "Characterizing Computer Performance with a Single Number", *Communications of the ACM*, 1988, 31(10): 1202-1206.

[29] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software Engineering: Metrics and Models*, CA: Benjamin-Cummings Publishing Co., Inc., 1986.

[30] N. Gorla and R. Ramakrishnan, "Effect of Software Structure Attributes on Software Development Productivity", *Journal of Systems and Software*, 1997, 36(2): 191-199.

[31] D. P. Darcy, C. F. Kemerer, S. Slaughter, et al., "The Structural Complexity of Software: an Experimental Test", *IEEE Transactions on Software Engineering*, 2005, 31(11): 982-995.

[32] S. S. Yau and J. S. Collofello, "Some Stability Measures for Software Maintenance", *IEEE Transactions on Software Engineering*, 1980, SE-6(6): 545-552.

[33] W. Pan, B. Li, Y. Ma, et al., "Measuring Structural Quality of Object-Oriented Softwares via Bug Propagation Analysis on Weighted Software Networks", *Journal of Computer Science and Technology*, 2010, 25(6): 1202-1213.

[34] W. Pan, B. Li, Y. Ma, et al., "Class Structure Refactoring of Object-Oriented Softwares using Community Detection in Dependency Networks", *Frontier of Computer Sciecne in China*, 2009, 3(3): 396-404.

[35] E. W. Dijkstra, "The Structure of the 'T.H.E.' multiprogramming system", *Communications of the ACM*, 1968, 11(5): 453-457.

[36] R. Subramanyan and M. S. Krisnan, "Empircial Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects", *IEEE Transactions on Software Engineering*, 2003, 29(4): 297-310.

[37] T. Gyimóthy, R. Feren, and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction", *IEEE Transactions on Software Engineering*, 2005, 31(10): 897-910.

[38] Y. Zhou and H. Leung, "Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Serverity Faults", *IEEE Transactions on Software Engineering*, 2006, 32(10): 771-789.

[39] F. P. Brooks Jr, "Three Great Challenges for Half-Century-Old Computer Science", *Journal of the ACM*, 2003, 50(1): 25-26.

[40] 张钹, "网络与复杂系统", 科学中国人, 2004, (10): 37-37.

[41] 陈关荣, "复杂网络及其新近研究进展介绍", 力学进展, 2008, 38(6): 653-662.

[42] D. J. Watts and S. H. Strogatz, "Collective Dynamics of Small World Networks", *Nature*, 1998, 393: 440-442.

[43] A. L. Barabási and R. Albert, "Emergence of Scaling in Random Networks", *Science*, 1999, 286: 509-512.

[44] 何大韧, 刘宗华, 汪秉宏, 复杂系统与复杂网络, 北京: 高等教育出版社, 2009.

[45] Committee on Network Science for Future Army Application, *Network Science*, Washington DC: National Academies Press, 2006.

[46] A. L. Barabási, *Linked: the New Science of Networks*, Cambridge MA: Perseus Publishing, 2002.

[47] D. J. Watts, "The "new" Science of Networks", *Annual Review of Sociology*, 2004, 30: 243-270.

[48] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law Relationships of The Internet Topology", *Computer Communication Review*, 1999, 29(4): 251-262.

[49] G. Siganos, M. Faloutsos, P. Faloutsos, et al., "Power Laws and The AS-Level Internet Topology", *IEEE/ACM Transactions on Networking*, 2003, 11(4): 514-524.

[50] L. A. Adamic and B. A. Huberman, "Power-Law Distribution of The World Wide Web", *Science*, 2000, 287: 2115a.

[51] R. Guimerà, S. Mossa, A. Turtschi, et al., "The Worldwide Air Transportation Network: Anomalous Centrality, Community Structure, and Cities' Global Roles", *Proceedings of the National Academy of Science USA*, 2005, 102: 3394-7799.

[52] W. Li and X. Cai, "Statistical Analysis of Airport Network of China", *Physical Review E*, 2004, 68(4): 46106.

[53] O. Sporns, "Network Analysis, Complexity, and Brain Function", *Complexity*, 2002, 8(1): 56-60.

[54] H. Jeong, B. Tombor, R. Albert, et al., "The Large-Scale Organizationn of Metabolic Networks", *Nature*, 2000, 407: 651-654.

[55] M. E. J. Newman, "Scientific Collaboration Networks. I. Network Construction and Fundamental Results", *Physical Review E*, 2001, 64(1): 16131.

[56] M. A. Serrano and M. Boguñá, "Topology of The World Trade Web", *Physical Review E*, 2003, 68: 015101.

[57] A. E. Motter, A. P. S. Moura, Y. C. Lai, et al., "Topology of The Conceptual Network of Language", *Physical Review E*, 2002, 65: 065102.

[58] G. Corso, "Families and Clustering in a Natural Numbers Network", *Physical Review E*, 2004, 60(3): 036106-036110.

[59] X. Liu, C. K. Tse, and M. Small, "Complex Network Structure of Musical Compositions: Algorithmic Generation of Appealing Music", *Physica A*, 2010, 389(1): 126-132.

[60] S. Abe and N. Suzuki, "Scale-Free Network of Earthquakes", *Europhysics Letters*, 2004, 65(4): 581-586.

[61] M. E. J. Newman, "The Structure and Functionn of Complex Networks", *SIAM Review*, 2003, 45: 167-256.

[62] C. R. Myers, "Software Systems as Complex Networks: Structure, Function, and Evolvability of Software Collaboration Graphs", *Physical Review E*, 2003, 68: 046116.

[63] S. Valverde, R. F. i Cancho, and R. Solé, "Scale Free Networks from Optimal Design", *Europhysics Letters*, 2002, 60: 512-517.

[64] G. Concas, M. Marchesi, P. Pinna, et al., "Power-Law in a Large Object-Oriented Software System", *IEEE Transactions on Software Engineering*, 2007, 33(10): 687-708.

[65] A. Potanin, J. Noble, M. Frean, et al., "Scale-Free Geometry in OO Programs", *Communication of the ACM*, 2002, 48(5): 99-103.

[66] G. Baxter, M. Frean, J. Noble, et al., "Understanding the Shape of Java Software", *ACM SIGPLAN Notices*, 2006, 41(10): 397-412.

[67] S. Jenkins and S. R. Kirk, "Software Architecture Graphs as Complex Networks: a Novel Parttion Scheme to Measure Stability and Evolution", *Information Sciences*, 2007, 177(12): 2587-2601.

[68] J. K. Chhabra and V. Gupta, "A Survey of Dynamic Software Metrics", *Journal of Computer Science and Technology*, 2010, 25(5): 1016-1029.

[69] B. W. Boehm, "Software Engineering", *IEEE Transactions on Computers*, 1976, C-25(12): 1226-1241.

[70] M. H. Halstead, *Elements of Software Science*, Elsevier Science Inc., New York, 1977.

[71] IEEE Standards Association, *IEEE Standard for a Software Quality Metrics Methodology*, IEEE Std. 1061-1992, 1992.

[72] R. R. Dumke and E. Foltin, "Metrics-Based Evaluation of Object-Oriented Software Development Methods", In *Proc. of the 2nd Euromicro Conference on Software Maintenance and Reengineering*, Florence, Italy, 1998, pp. 193-196.

[73] S. Morasca, *Software Measurement: State of the Art and Related Issues*, Slides from the School of the Italian Group of Informatics Engineering, Rovereto, Italy, September, 1995.

[74] R. W. Wolverton, "The Cost of Developing Large-Scale Software", *IEEE Transactions on Computers*, 1974, C-23(6): 615-636.

[75] T. J. McCabe, "A Complexity Measure", *IEEE Transactions on Software Engineering*, 1976, SE-2(4): 308-320.

[76] E. I. Oviedo, "Control Flow, Data Flow and Programmers Complexity", In *Proc. of COMPSAC 80*, Chicago, IL, 1980, pp. 146-152.

[77] W. A. Harrison and K. I. Magel, "A Complexity Measure based on Nesting Level", *ACM SIGPLAN Notices*, 1981, 16(3): 63-74.

[78] K. C. Tai, "A Program Complexity metric based on Data Flow Information in Control Graphs", In *Proc. of the 7th International Conference on Software Engineering*, Orlando, Florida, USA, 1984, pp. 239-248.

[79] M. R. Woodward, M. A. Hennell, and D. Hedley, "A Measure of Control Flow Complexity in Program Text", *IEEE Transactions on Software Engineering*, 1979, SE-5(1): 45-50.

[80] S. M. Henry and D. Kafura, "Software Structure Metrics Based on Information Flow", *IEEE Transactions on Software Engineering*, 1981, SE-7(5): 510-518.

[81] P. N. Robillard and G. Boloix, "The Interconnectivity Metrics: a New Metric Showing How a Program is Organized", *Journal of System and Software*, 1989, 10(1): 29-39.

[82] 邢大红, 曹佳冬, 汪和才, 等, "软件度量学综述", 计算机工程与应用, 2001, 37(1): 17-19.

[83] Morris and L. Kenneth, *Metrics for Object-Oriented Software Development Environments*, Massachusetts Institute of Technology, Master of Science in Management, May 1989.

[84] J. M. Bieman, *Deriving Measures of Software Reuse in Object Oriented Systems*, Technical Report #CS91-112, Colorado State University, Fort Collins/Colorado, USA, July 1991.

[85] A. Lake and C. R. Cook, *A Software Complexity Metric for C++*, Technical Report. Oregon State University, Corvallis, OR, USA, July 1992.

[86] R. Sharble and S. Cohen, "The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods", *ACM SIGSOFT Software Engineering Notes*, 1993, 18(2): 60-73.

[87] F. B. Abreu, "The MOOD Metrics Set", In *Proc. of the ECOOP'95 Workshop on Metrics*, 1995.

[88] F. B. Abreu, "Design Metrics for OO Software System", In *Proc. of the ECOOP'95 Quantitative Methods Workshop*, 1995.

[89] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability", *Journal Of Systems And Software*, 1993, 23(2): 111-122.

[90] Y. Lee, B. Liang, S. Wu, and F. Wang, "Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow", In *Proc. of the International Conference on Software Quality*, Maribor, Slovenia, 1995, pp. 81-90.

[91] M. Sarker, *An overview of Object Oriented Design Metrics*, Sweden: Umea University, 2005.

[92] S. Valverde and R. Solé, *Hierarchical Small Worlds in Software Architecture*, Working Paper of Santa Fe Institue, SFI/03-07-44, 2003.

[93] A. P. S. Moura, Y. C. Lai, and A. E. Motter, "Signatures of Small-World and Scale-Free Properties in Large Computer Programs", *Physical Review E*, 2003, 68: 017102.

[94] N. LaBelle and E. Wallingford, "Inter-Package Dependency Networks in Open-Source Software", ArXiv: Cs.SE/0411096, 2004.

[95] S. Valverde and R. V. Solé, "Universal Properties of Bipartite Software Graphs", In *Proc. of the 9th IEEE International Conference on Engineering of Complex Computer Systems*, Florence, Italy, 2004.

[96] 韩明畅, 李德毅, 刘常昱, 等, "软件中的网络化特征及其对软件质量的贡献", 计算机工程与应用, 2006, 42(20): 29-31.

[97] J. Liu, K. He, Y. Ma, et al., "Scale Free in Software Metrics", In *Proc. of the 30th Annual International Computer Software and Application Conference*, USA, 2004, pp. 229-235.

[98] J. Liu, K. He, R. Peng, et al., "A Study on the Weight and Topology Correlation of Object-Oriented Software Coupling Network", In *Proc. of the 1st International Conference on Complex Systems and Applications*, Inner Mongolia, China, 2006, PP. 955-959.

[99] 闫栋, 祁国宁, "大规模软件系统的无标度特性与演化模型", 物理学报, 2006, 55(8): 3799-3084.

[100] H. Zhang, H. Zhao, W. Cai, et al., "Using the k-core Decomposition to Analyze the Static Structure of Large-Scale Software Systems", *The Journal of Supercomputing*, 2010, 53(2): 352-369.

[101] M. Shi, X. Li, and X. Wang, "Evolving Topology of Java Networks", In *Proc. of the 6th World Congress on Control and Automation*, Dalian, China, 2006, PP. 21-23.

[102] L. Wang, Z. Wang, C. Yang, et al., "Linux Kernels as Complex Networks: a Novel Method to Study Evolution", In *Proc. of the 25th International Conference on Software Maintenance*, Edmonton, Alberta, Canada, 2009, PP. 41-50.

[103] H. Li, B. Huang, and J. Lü, "Dynamical Evolution Analysis of the Object-Oriented Software Systems", In *Proc. of the 2008 IEEE World Congress on Computational Intelligence*, Hong Kong, 2008, PP. 3035-3040.

[104] R. V. Solé and S. Valverde, "Information Theory of Complex Networks: on Evolution and Architectural Constraints", In *Proc. of the International Conference on Complex Networks*, 2004, pp. 189-207.

[105] S. Valverde and R. V. Solé, "Network Motifs in Computational Graphs: a Case Study in Software Architecture", *Physical Review E*, 2005, 72: 026107.

[106] K. He, R. Peng, J. Liu, et al., "Network Motifs in Computational Graphs: a Case Study in Software Architecture", *Journal of Systems Science and Complexity*, 2006, 19(2): 157-181.

[107] 李兵, 王浩, 李增扬, 等, "基于复杂网络的软件复杂性度量研究", 电子学报, 2006, 34(12A): 2371-2375.

[108] H. Li, "Scale-Free Network Models with Accelerating Growth", *Frontier of Computer Science in China*, 2009, 3(3): 373-380.

[109] R. Vasa, J. G. Schneider, C. Woodward, et al., "Detecting Structural Changes in Object Oriented Software Systems", In *Proc. of the International Symposium on Empirical Software Engineering*, Noosa Heads, Australia, 2005, PP. 479-486.

[110] Y. Ma, K. He, and D. Du, "A Qualitative Method for Measuring the Structural Complexity of Software Systems based on Complex Networks", In *Proc. of the 12th Asia-Pacific Software Engineering Conference*, Taipei, Taiwan, 2005, PP. 257-263.

[111] A. Girolamo, L. I. Newman, and R. Rao, "The Structure and Behavior of Class Networks in Object-Oriented Software Design", www.eecs.umich.edu/ leenewm/documents/classnetworks.pdf, 2005.

[112] Y. Ma, K. He, D. Du, et al., "A Complexity Metrics Set for Large-Scale Object-Oriented Software Systems", In *Proc. of the 6th International Conference on Computer and Information Technology*, Seoul, 2006, PP. 257-263.

[113] R. Vasa, J. G. Schneider, and O. Nierstrasz, "The Inevitable Stability of Software Change", In *Proc. of the 23nd IEEE International Conference on Software Maintenance*, Paris France, 2007, PP. 4-13.

[114] H. Melton and E. Tempero, "Static Members and Cycles in Java Software", In *Proc. of the 1st International Symposium on Empirical Software Engineering and Measurement*, Madrid, 2007, PP. 136-145.

[115] Y. Ma, K. He, and J. Liu, "Network Motifs in Object-Oriented Software Systems", *Dynamics of Continuous, Discrete and Impulsive System (Series B: Applications and Algorithms) Special Issue on Software Engineering and Complex Networks*, 2007, 14(S6): 166-172.

[116] C. E. Woodcock and H. S. Alan, "The Factor of Scale in Remote Sensing", *Remote Sensing of Environment*, 1987, 21(3): 311-332.

[117] H. C. Gall and M. Lanza, "Software Evolution: Analysis and Visualization", In *Proc. of the 28th International Conference on Software Engineering*, Shanghai, China, 2006, pp. 1055-1056.

[118] C. Darwin, *On the Origin of Species: By Means of Natural Selection or the Preservation*, London: John Murray, 1859.

[119] M. M. Lehman, J. F. Ramil, P. D. Wernick, et al., "Metrics and Laws of Software Evolution — The Nineties View", In *Proc. of the 4th International Conference on Software Metrics Symposium*, Albuquerque, NM, USA, 1997, pp. 20-32.

[120] M. W. Godfrey and D. M. German, "The Past, Present, and Future of Software Evolution", In *Proc. of the 24th International Conference on Software Maintenance (Speical Track on Frontiers of Software Maintenance)*, Beijing, China, 2008, pp. 129-138.

[121] D. L. Parnas, "Software Aging", In *Proc. of the 16th International Conference on Software Engineering*, Sorrento, Italy, 1994, pp. 279-287.

[122] B. Boehm, "The Changing Nature of Software Evolution", *IEEE Software*, 2010, 27(4): 26-29.

[123] I. Herraiz, *A Statistical Examination of the Evolution and Properties of Libre Software*, Madrid: Universidad Rey Juan Carlos, 2008.

[124] M. D'Ambros, "Supporting Software Evolution Analysis with Historical Dependencies and Defect Information", In *Proc. of the 24th IEEE International Conference on Software Maintenance*, Beijin, China, 2008, pp. 412-415.

[125] T. Mens, Y.-G. Guéhéneuc, J. Fernández-Ramil, et al., "Software Evolution: Maintaining Stakeholders' Satisfaction in a Changing World — Guest Editor's Introduction", *IEEE Software*, 2010, 27(4): 22-25.

[126] N. H. Madhavji, J. C. Fernández-Ramil, and D. E. Perry, *Software Evolution and Feedback: Theory and Practice*, John Wiley & Sons, 2006.

[127] L. Yu, S. Ramaswamhy, and J. Bush, "Symbiosis and Software Evolvability", *IT Professional*, 2008, 10(4): 56-62.

[128] M. Lehman and L. Belady, *Program Evolution: Processes of Software Change*, London: London Academic Press, 1985.

[129] K. Beck, "The Inevitability of Evolution", *IEEE Software*, 2010, 27(4): 28-29.

[130] H. P. Breivold, M. A. Chauhan, and M. A. Babar, "A Systematic Review of Studies of Open Source Software Evolution", In *Proc. of the 17th Asia Pacific Software Engineering Conference*, Sydney, Australia, 2010, pp. 356-365.

[131] M. Rochkind, "The Source code Control System", *IEEE Transactions on Software Egeineering*, 1975, 1(4): 364-370.

[132] J. Hunt and D. Mcllroy, *An Algorithm for Differential File Comparision*, Technical Report CSTR 41, Bell Laboratories, Murray Hill, NJ, 1976.

[133] M. M. Lehman, D. E. Perry, and J. F. Ramil, "Implication of Evolution Metrics on Software Maintenance", In *Proc. of the International Conference on Maintenance*, Bethesda, Maryland, 1998, pp. 208-217.

[134] M. M. Lehman and J. F. Ramil, "Rules and Tools for Software Evolution Planning and Management", *Annals of Software Engineering*, 2001, 11: 153-44.

[135] C. K. S. Chong Hok Yuen, "An Empirical Approach to the Study of Errors in Large Software under Maintenance", In *Proc. of the IEEE Conference on Software Maintenance*, Washington, DC, 1985, pp. 96-105.

[136] C. K. S. Chong Hok Yuen, "A Statistical Rationale for Evolution Dynamics Concepts", In *Proc. of the IEEE Conference on Software Maintenance*, Austin, Tex, 1987, pp. 156-164.

[137] C. K. S. Chong Hok Yuen, "On Analyzing Maintenance Process Data at the Global and Detailed Levels: A Case Study", In *Proc. of the IEEE Conference on Software Maintenance*, Phoenix, Az, 1988, pp. 248-255.

[138] T. Tamai and Y. Torimitsu, "Software Lifetime and Its Evolution Process over Generations", In *Proc. of the IEEE Conference on Software Maintenance*, Orlando, FL, USA, 1992, pp. 63-69.

[139] C. R. Cook and A. Roesch, "Real-Time Software Metrics", *Journal of Systems and Software*, 1994, 24(3): 223-237.

[140] C. F. Kemerer and S. Slaughter, "An Empirical Approach to Studying Software Evolution", *IEEE Transactions on Software Engineering*, 1999, 25(4): 493-509.

[141] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two Case Studies of Open Source Development: Apache and Mozilla", *ACM Transactions on Software Engineering and Methodology*, 2002, 11(3): 309-346.

[142] D. German, "Using Software Trails to Rebuild the Evolution of Software", *ELISA Workshop on Evolution of Large-Scale Industrial Software Applications*, Amsterdam, Netherlands, 2002.

[143] I. Antoniades, I. Samoladas, I. Stamelos, et al., "Dynamical Simulation Models of the Open Source Development Process", *Free/Open Source Software Development*, Idea Group Pulishing, Hershey, PA, 2004, pp. 174-202.

[144] A. Capiluppi, "Models for the Evolution of OS Projects", In *Proc. of the 7th International Conference on Software Maintenance*, Amsterdam, The Netherlands, 2003, pp. 65-74.

[145] M. W. Godfrey and Q. Tu, "Evolution in Open Source Software: a Case Study", In *Proc. of the 16th International Conference on Software Maintenance*, San Jose, California, 2000, pp. 131-142.

[146] F. V. Rysselberghe and S. Demeyer, "Studying Software Evolution Information by Visualizing the Change History", In *Proc. of the 20th International Conference on Software Maintenance*, Chicago, Illinois, 2004, PP. 328-337.

[147] J. Gosling, B. Joy, G. Steele, et al., *The Java Language Specification, Second Edition*, New York: Addison Wesley, 2000.

[148] B. Berliner and J. Polk, *Concurrent Version System (CVS)*, `www.cvshome.org`, 2001.

[149] Sourceforge, `http://sourceforge.net`, visited Feb. 28, 2011.

[150] Freshmeat, `http://freshmeat.net`, visited Feb. 28, 2011.

[151] 何克清, 马于涛, 刘婧, 等, 软件网络, 北京: 科学出版社, 2008.

[152] XDepend, `http://www.xdepend.com`, visited Feb. 28, 2011.

[153] DMS Software Reengineering Toolkit, `http://www.semanticdesigns.com/Products/DMS/DMSToolkit.html`, visited Feb. 28, 2011.

[154] GNU gprof, `http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html`, visited Feb. 28, 2011.

[155] The Web site of Azureus, `http://sourceforge.net/projects/azureus/files/`, May 5, 2011.

[156] L. da F. Costa, F. A. Rodrigues, G. Travieso, et al., "Characterization of Complex Networks: a Survey of Measurements", *Advanced Physics*, 2007, 56: 167-242.

[157] D. R. Cox and P. A. W. Lewis, *The Statistical Analysis of Series of Events*, London: Chapman and Hall, 1966.

[158] A. L. Barabási, H. Jeong, and Z. Néda, "Evolution of the Social Network of Scientific Collaborations", *Physica A*, 2002, 311(3-4): 590-614.

[159] A. P. Masucci and G J Rodgers, "Network Properties of Written Human Language", *Physical Review E*, 2006, 74: 026102.

[160] Log-Log Plot, `http://en.wikipedia.org/wiki/Log-log_plot`, visited Feb. 28, 2011.

[161] X. Zheng, D. Zeng, H. Li, et al., "Analyzing Open-Source Software System as Complex Networks", *Physica A*, 387(24): 6190-6200, 2008.

[162] M. E. J. Newman, "Fast Algorithm for Detecting Community Structure in Networks", *Physical Review*, 2004, 69: 066133.

[163] M. E. J. Newman, "Assortative mixing in networks", *Physical Review Letters*, 2002, 89: 208701.

[164] 潘伟丰, *Xiaokeba's BLOG*, `http://blog.sina.com.cn/breezepan`, 2011.

[165] M. Fowler, *Patterns and Advanced Principles of OOD*, Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2000.

[166] J. Koskinen, *Software Maintenance Costs*, `http://users.jyu.fi/~koskinen/smcosts.htm`, 2011.

[167] S. Yau and J. S. Collofello, "Design Stability Measures for Software Maintenance", *IEEE Transactions on Software Engineering*, 11: 849-856, 1985.

[168] F. Jay and R. Mayer, *IEEE Standard Glossary of Software Engineering Terminology*, Los Alamitos: IEEE Computer Socienty Press, 1990.

[169] S. Yau and J. S. Collofello, "Some Stability Measures for Software Maintenance", *IEEE Transactions on Software Engineering*, 6: 545-552, 1980.

[170] S. Bohner and R. Arnold, *Software Change Impact Analysis*, Los Alamitos: IEEE Computer Socienty Press, 1996.

[171] D. Kung, J. Gao, P. Hsia, et al., "Change Impact Identification in Object-Oriented Software Maintenance", In *Proc. of the IEEE International Conference on Software Maintenance*, Victoria, BC, Canada, 1994, PP. 202-211.

[172] T. Zimmermann, S. Diehl, and A. Zeller, "How History Justifies System Architecture (or not)", In *Proc. of the 6th International Workshop on Principles of Software Evolution*, Washington, DC, USA, 2003, PP. 73-83.

[173] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, et al., "Probabilistic Evaluation of Object-Oriented Systems", In *Proc. of the 10th International Symposium on Software metrics*, Chicago, IL, USA, 2004, PP. 26-33.

[174] A. E. Hassan and R. C. Holt, "Predicting Change Propagation in Software Systems", In *Proc. of the 20th IEEE International Conference on Software Maintenance*, Chicago, IL, USA, 2004, PP. 284-293.

[175] A. MacCormack, J. Rusnak, and C. Y. Bald Win, "Exploring the Structure of Complex Software Designs: an Empirical Study of Open Source and Proprietary Code", *Management Science*, 2006, 52(7): 1015-1030.

[176] I. P. Shaik, W. Abdelmoez, R. Gunnalan, et al., "Using Change Propagation Probabilities to Assess Quality Attributes of Software Architectures 1", In *Proc. of the IEEE International Conference on Computer Systems and Applications*, Dubai/Sharjah, UAE, 2006, PP. 704-711.

[177] S. Mirarab, A. Hassouna, and L. Tahvildari, "Using Bayesian Belief Networks to Predict Change Propagation in Software Systems", In *Proc. of the 15th IEEE International Conference on Program Comprehension*, Banff, AB, Canada, 2007, PP. 177-186.

[178] J. Liu, J. Lü, K. He, et al., "Characterizing the Structural Quality of General Complex Software Networks", *International Journal of Bifurcation and Chaos*, 2008, 18(4): 605-613.

[179] L. Li, G. Qian, and L. Zhang, "Evaluation of Software Change Propagation using Simulation", In *Proc. of the 2009 World Congress on Software Engineering*, Xiamen, China, 2009, PP. 28-33.

[180] L. Zhang, G. Qian, and L. Li, "Software Stability Analysis Based on Change Impact Simulation", *Jisuanji Xuebao/Chinese Journal of Computers*, 2010, 33(3): 440-451.

[181] B. G. Ryder and F. Tip, "Change Impact Analysis for Object-Oriented Programs", In *Proc. of the ACM SIGPLAN-SIGSOFT Workshop on Program analysis for Software Tools and Engineering*, Snowbird, UT, United states, 2001, PP. 45-53.

[182] The Web site of JUnit, `http://junit.sourceforge.net`, May 5, 2011.

[183] E. J. Weyuker, "Evaluating Software Complexity Measures", *IEEE Transactions on Software Engineering*, 1988, 14(9): 1357-1365.

[184] S. R. Chidamber and C. F. Kemerer, "A Metrics Suit for Object-Oriented Design", *IEEE Transactions on Software Engineering*, 1994, 20(6): 476-493.

[185] W. Harrison, "An Entropy-based Measure of Software Complexity", *IEEE Transactions on Software Engineering*, 1992, 18(11): 1025-1029.

[186] H. Zhang, Y. F. Li, and H. B. K. Tan, "Measuring Design Complexity of Semantic Web Ontologies", *Journal of Systems and Software*, 2010, 83(5): 803-814.

[187] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis, "Design Pattern Detection using Similarity Scoring", *IEEE Transactions on Software Engineering*, 2006, 32(11): 896-909.

[188] Data of Design Principles and Design Pattern, `http://www.objectmentor.com`, Dec 20, 2010.

[189] J. Kerievsky, *Refactoring to Patterns*, New York: Addison Wesley, 2004.

[190] O. Seng, J. Stanmmel, and D. Burkhart, "Search-based Determination of Refactorings for Improving the Class Structure of Object-Oriented System", In *Proc. of the Genetic and Evolutionary Computation Conference*, Seattle, WA, United states, 2006, PP. 1909-1916.

[191] T. Mens, T. Tourwé, "A Survey of Software Refactoring", *IEEE Transactions on Software Engineering*, 2004, 30(2): 126-139.

[192] M. Fowler, *Refactoring: Improving the Design of Existing Code*, New York: Addison Wesley, 1990.

[193] W. J. Brown, R. Malveau, H. McCormick, et al., *Antipatterns: Refactoring Software, Architectures, and Projects in Crises*, New York: John Wiley and Sons, 1998.

[194] B. Kang and J. M. Bieman, "A Quantitative Framework for Software Restructuring", *Journal of software maintenance: Research and Practice Software*, 1999, 11(4): 145-284.

[195] T. Dudzikan and J. Wlodka, *Tool-Supported Discovery and Refactoring of Structural Weakness in Code*, TU Berlin: Masters' thesis, 2002.

[196] I. Ivkovic and K. Kontogiannis, "A Framework for Software Architecture Refactoring using Model Transformations and Semantic Annotations", In *Proc. of the Conference on Software Maintenance and Reengineering*, Bari, Italy, 2006, pp. 135-144.

[197] I. G. Czibula and G. Serban, "Improving System Design using a Clustering Approach", *International journal of computer science and network security*, 2006, 6(12): 40-49.

[198] B. W. Kernighan and S. Lin S, "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell System Technical Journal*, 1970, 49: 291-307.

[199] M. E. J. Newman, "Finding Community Structure in Networks using the Eigenvectors of Matrices", *Physical Review E*, 2006, 74: 036104, 2006.

[200] L. Angelini, S. Boccaletti, D. Marinazzo, et al., "Fast Identification of Network Modules by Optimization of Ratio Association", *arXiv:cond-mat/0610182v2*, 2006.

[201] F. Radicchi, C. Castellano, F. Cecconi, et al., "Defining and Identifying Communities in Networks", *Proceedings of the National Academy of Science of the USA*, 2004, 101(9): 2658-2663.

[202] M. E. J. Newman and M. Girvan, "Finding and Evaluating Community Structure in Networks", *Physical Review E*, 2004, 69: 026113.

[203] N. A. Alves, "Unveiling Community Structures in Weighted Networks", *Physical Review E*, 2007, 76: 036101.

[204] I. Farkas, D. Ábel, G. Palla, et al., "Weighted Network Modules", *New Journal of Physics*, 2007, 9(6): 180-199.

[205] S. H. Zhang, R. S. Wang, and X. S. Zhang, "Identification of Overlapping Community Structure in Complex Networks using Fuzzy c-means Clustering", *Physica A*, 2007, 374: 483-490.

[206] S. H. Zhang, R. S. Wang, and X. S. Zhang, "Uncovering Fuzzy Community Structure in Complex Networks", *Physical Review E*, 2007, 76: 046103.

[207] A. Clauset, M. E. J. Newman, and C. Moore, "Finding Community Structure in Very Large Networks", *Physical Review E*, 2004, 70: 066111.

[208] J. Duch and A. Arenas, "Community Detection in Complex Networks using Extremal Optimization", *Physical Review E*, 2005, 72: 027104.

[209] A. Clauset, "Finding Local Community Structure in Networks", *Physical Review E*, 2005, 72: 026132.

[210] J. P. Bagrow, "Evaluating Local Community Methods in Networks", *Journal of Statistical Mechanics: Theory and Experiment*, 2008, P05001.

[211] M. Bauer and M. Trifu, "Architecture-Aware Adaptive Clustering of OO Systems", In *Proc. of the 8th European Conference on Software Maintenance and Reengineering*, Tampere, Finland, 2004, PP. 3-14.

[212] S. Fortunato and M. Barthelemy, "Resolution Limit in Community Detection", *Proceedings of the National Academy of Science of the USA*, 2007, 104: 36-41.

[213] M. Rosvall and C. T. Bergstrom, "An Information-Theoretic Framework for Resolving Community Structure in Complex Networks", *Proceedings of the National Academy of Science of the USA*, 2007, 104(18): 7327-7331.

[214] S. Muff, F. Rao, and A. Caflisch, "Local Modularity Measure for Network Clusterizations", *Physical Review E*, 2005, 72: 056107.

[215] G. J. Myers, *The Art of Software Testing*, New York: Wiley, 1979.

[216] G. Rothermel and M. J. Harrold, "Analyzing Regression Test Selection Techniques", *IEEE Transactions on Software Engineering*, 1996, 22(8): 529-551.

[217] M. J. Harrold, D. Rosenblum, G. Rothermel, et al., "Empirical Studies of a Prediction Model for Regression Test Selection", *IEEE Transactions on Software Engineering*, 2001, 27(3): 248-263.

[218] G. Rothermel, R. J. Untch, and C. Y. Chu, "Prioritizing Test Cases for Regression Testing", *IEEE Transactions on Software Engineering*, 2001, 27(10): 929-948.

[219] J. M. Kim and A. Porter, "A History-Based Test Prioritization Technique for Regression Testing in Resource Constrained Environments", In *Proc. of the 24th International Conference on Software Engineering*, Orlando, FL, United states, 2002, PP. 119-129.

[220] H. Srikanth, L. Williams, and J. Osborne, "System Test Case Prioritization of New and Regression Test Cases", In *Proc. of the International Symposium on Empirical Software Engineering*, Queensland, Australia, 2005, PP. 62-71.

[221] Z. Li, M. Harman, and R. M. Hierons, "Search Algorithm for Regression Test Case Prioritization", *IEEE Transactions on Software Engineering*, 2007, 33(4): 225-237.

[222] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test Case Prioritization: a Family of Empirical Studies", *IEEE Transactions on Software Engineering*, 2002, 28(2): 159-182.

[223] K. S. Lew, T. S. Dillon, and K. E. Forward, "Software Complexity and Its Impact on Software Reliability", *IEEE Transactions on Software Engineering*, 1988, 14(11): 1645-1655.

[224] E. N. Adams, "Optimizing Preventive Service of Software Products", *IBM Journal for Research and Development*, 1984, 28(01): 3-14.

[225] B. Boehm and V. R. Basili, "Software Defect Reduction Top 10 List", *Computer*, 2001, 34(1): 135-137.

[226] Xml-security, `http://santuario.apache.org`, visited Mar. 6, 2011.

[227] JTopas, `http://sourceforge.net/projects/jtopas`, visited Mar. 6, 2011.

[228] S. Elbaum and A. G. Malishevsky, "Prioritizing Test Case for Regression Testing", In *Proc. of the International Symposium on Software Testing and Analysis*, Roma, Italy, 2000, PP. 102-112.

[229] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization", In *Proc. of the International Conference on Software Engineering*, Toronto, Ont, Canada, 2001, PP. 329-338.

[230] J. Jones and M. J. Harrold, "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage", In *Proc. of the International Conference on Software Maintenance*, Florence, Italy, 2001, PP. 92-101.

[231] H. Srikanth, "Requirements-Based Test Case Prioritization", In *Proc. of the 12th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, Newport Beach, California, USA, 2004

[232] K. R. Walcott, M. L. Soffa, and G. M. Kapfhammer, "Time-Aware Test Suit Prioritization", In *Proc. of the International Symposium on Software Testing and Analysis*, Portland, ME, United states, 2006, pp. 1-11.

[233] 屈波, 聂长海, 徐宝文, "基于测试用例设计信息的回归测试优先级算法", 计算机学报, 2008, 31(3): 431-439.

[234] L. Zhang, S. Hou, C. Guo, et al., "Time-Aware Test-Case Prioritization using Integer Linear Programming", In *Proc. of the International Symposium on Software Testing and Analysis*, Chicago, Illinois, USA, 2009, pp. 213-224.

[235] L. Zhang, J. Zhou, D. Hao, et al., "Prioritizing JUnit Test Cases in Absence of Coverage Information", In *Proc. of the 25th International Conference on Software Maintenance*, Edmonton, Alberta, Canada, 2009, pp. 19-28.

[236] H. Do, G. Rothermel, and A. Kinneer, "Empirical Studies of Test Case Prioritization in a JUnit Testing Environment", In *Proc. of the 15th International Symposium on Software Reliability Engineering*, Saint-Malo, Bretagne, France, 2004, pp. 113-124.

[237] H. Do, G. Rothermel, and A. Kinneer, "Prioritizing JUnit Test Cases: an Empirical Assessment and Cost-Benefits Analysis", *Empirical Software Engineering: an International Journal*, 2006, 11: 33-70.

[238] V. R. Basili, L. C. Briand, and W. L. Melo, "Validation of Object-Oriented Design Metrics as Quality Indicators", *IEEE Transactions on Software Engineering*, 1996, 22(10): 751-761.

[239] K. EI Emam, S. Benlarbi, and N. Goel, "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics", *IEEE Transactions on Software Engineering*, 2001, 27(6): 630-650.

[240] JMetric, `http://sourceforge.net/projects/jmetric`, visited Mar. 6, 2011.

[241] OODMetrics, `http://sourceforge.net/projects/oodmetrics/`, visited Mar. 6, 2011.

[242] Code Critickbeta, `http://sourceforge.net/projects/codecritick/`, visited Mar. 6, 2011.

[243] JUnit, `http://junit.sourceforge.net/`, visited Mar. 6, 2011.

[244] djUnit, `http://works.dgic.co.jp/djunit/`, visited Mar. 6, 2011.

[245] J. Xu, D. Ho, and L. F. Capretz, "An Empirical Validation Object-Oriented Design Metrics for Fault Prediction", *Journal of Computer Science*, 2008, 4(7): 571-577.

[246] H. Do, S. Elbaum, and G. Rothermel, "Supporting Controlled Experimentation with Testing Techniques: an Infrastructure and Its Potential Impact", *Empirical Software Engineering: an International Journal*, 2005, 10(4): 405-435.

[247] 景涛, 江昌海, 胡德斌, "软件关联缺陷的一种检测方法", 软件学报, 2005, 16(1): 17-28.

攻博期间发表的文章、参与的研究项目及获得的奖励

一、发表的文章

[1] **Weifeng Pan**, Bing Li, Yutao Ma, et al., "Measuring Structural Quality of Object-Oriented Softwares via Bug Propagation Analysis on Weighted Software Networks", *Journal of Computer Science and Technology*, 2010, 25(6): 1202-1213. (**SCIE & EI Compendex**)

[2] **Weifeng Pan**, Bing Li, Yutao Ma, et al., "Class Structure Refactoring of Object-Oriented Softwares using Community Detection in Dependency Networks", *Frontier of Computer Sciecne in China*, 2009, 3(3): 396-404. (**EI Compendex**)

[3] **Weifeng Pan**, Bing Li, Yutao Ma, et al., "A Novel Software Evolution Model based on Software Networks", In *Proc. of the 1st International Conference of Complex Networks*, Shaihai, China, Feb. 23-25, 2009, pp.1281-1291.

[4] Bing Li, **Weifeng Pan**, and Jinhu Lü, "Multi-Granularity Dynamic Analysis of Complex Software Networks", In *Proc. of 2011 IEEE International Symposium on Circuits and Systems*, Rio De Janeiro, Brazil, May 15-18, 2011. (**to appear**)

[5] **Weifeng Pan**, Bing Li, Yutao Ma, et al., "Multi-Granularity Evolution Analysis of Software using Complex Network Theory", *Journal of System Science and Complexity*. SCI 期刊 (**revise**)

[6] **Weifeng Pan** and Bing Li, "Measuring the Stability of Software Systems via Simulation in Dependency Networks", *Journal of Information Science and Engineering*. SCI 期刊 (**under review for 5 month**)

[7] 潘伟丰, 李兵, 马于涛, 等, "基于加权类依赖网络的面向对象软件测试用例排序", 中国科学*: F* 辑. SCI 期刊 (**已投稿**)

[8] 潘伟丰, 李兵, 邵波, 等, "基于软件网络的服务自动分类和推荐方法研究", *CCF NCSC 2011*, 2011. (**已录用**)

[9] Li Qin, Bing Li, **Weifeng Pan**, et al., "A Novel Method for Mining SaaS Software Tag via Community Detection in Software Services Network", In *Proc. of the 1st International Conference on Cloud Computing*, Beijing, China, Dec. 1-4, 2009, pp. 312-321. (**EI Compendex & ISTP**)

[10] Tao Peng, Bing Li, **Weifeng Pan**, et al., "Requirements Discovery based on RGPS using Evolutionary Algorithmk", In *Proc. of the 21st International Conference on Software Engineering and Knowledge Engineering*, Boston, Massachusetts, USA, July 1-3, 2009, pp. 286-290. (**EI Compendex**)

[11] 李杉, 李兵, **潘伟丰**, 等, "一种mashup服务描述本体的自动构建方法", 小型微型计算机系统, 2011. (**to appear**)

二、参与的科研项目

[1] 国家自然科学基金项目, 2009-2011, 项目批准号: 60873083.

[2] 湖北省青年杰出人才基金, 2009-2010, 项目批准号: 2008CDB351. (第三)

[3] 国家自然科学基金项目, 2011-2012, 项目批准号: 61003073. (第四)

[4] 国家自然科学基金项目, 2010-2012, 项目批准号: 60903034. (第四)

[5] 中央高校基本科研业务费专项资金, 2009-2011, 项目批准号: 6082005. (第六)

[6] 教育部博士点基金, 2010-2012, 项目批准号: 20090141120022. (第八)

三、获得的奖励

[1] 2008-2009 荣获丙等奖学金、武汉大学“优秀研究生”荣誉称号

[2] 2009-2010 荣获乙等奖学金、武汉大学“优秀研究生”荣誉称号

致　　谢

本论文是我的博士毕业论文，也是我对自己博士期间研究成果的一次总结。在武汉大学软件工程国家重点实验室这个学术氛围浓厚的环境里，我度过了人生中一段美好时光，得到了许多老师、同学、朋友和家人的帮助，他们的热情指导和关怀令我终身难忘。我在武大学习三年的时间了，总结到今天就两个字感谢。

我首先要感谢我的导师李兵教授。在攻读博士学位期间，李老师在学习、工作和生活上一直给我以悉心的指导和无微不至的关怀。如果没有他的支持和帮助，我不可能完成我的学业、科研和论文。李老师严谨务实的治学态度、不断进取的奋斗精神和学术上的远见卓识一直是我学习的榜样。曾记得，为了促进课题组同学在学术研究上相互了解和促进，李老师放弃了休息时间，特地组织了每周六的讨论班，每次都亲自出席指导，风雨无阻，使我们获益匪浅；曾记得，为了帮助我尽快进入新的研究方向，李老师曾与我多次交流，为我指明研究方向，并引导我学习写作的基本知识和技巧；曾记得，为了让我们能了解本领域的前沿知识，并扩大与别人的交流，李老师常自费资助我们外出与人交流；曾记得，为了我能找一份好的工作，李老师四方打听，并给我提供了许多宝贵的意见。并且，李老师在百忙之中还非常关心我的生活，经常给我帮助，给予我极大的宽容，让我倍感关爱。此时此刻，对李老师的感激之情，不能言表。

同时，我还要感谢软件工程国家重点实验室的全体老师。在实验室的三年求学生活中，我得到了大家的很多支持与帮助。与他们在一起，使我体会到研究和学习的乐趣。他们是：何克清教授、应时教授、彭蓉教授、梁鹏教授、马于涛老师、刘婧老师、何扬帆老师、王翀老师、王健老师，冯在文老师、陈秀红老师等。特别要感谢吕金虎教授、陆君安教授、吴志健教授、刘进教授给予我的很多帮助和指导。感谢实验室郑英老师、郑艺老师在学习和生活方面给予我的帮助。

我还要感谢我的同门师兄弟、师姐妹，与他们的讨论和交流使我获益匪浅，他们是：曾诚、曹步清、李杉、汪北阳、黄媛、侯婷婷、邵波、沈水晶、何鹏、李定威、秦丽、李其锋、祝建平、肖奎、姚竞、彭涛、余智涛等，特别是感谢覃叶宜、蒲希、周晓燕师妹为本文的部分工作提供了基础。

感谢 2008 级博士班的所有同学，特别是刘坤、文斌、陈华峰、胡博、李蓉、汪靖、董晓健、吴昱、李文凤、赵楷、胡罗凯、王权于、王晖等众多同窗好友，在与他们三年的学习和生活中，我们留下了很多美好的回忆，正是由于他们的陪伴让我度过了愉快而又难忘的两年的美好时光。

在这里，我还要感谢加拿大滑铁卢大学（University of Waterloo）的 Siavash Mirarab，希腊马其顿大学（University of Macedonia）的 Nikolaos Tsantalis，清华大学软件学院的张洪宇教授，南京大学的周毓明教授、陈振宇教授给于我的帮助。

感谢我的女朋友丁丽娜，一年多来，她总是在背后默默地支持我!

最后，我要感谢生育我、养育我、教育我的父母，我将以一生来回报。

潘伟丰

二〇一一年四月于武汉大学信息学部 10 号楼 512 宿舍