基于变更传播仿真的软件稳定性分析

张 莉 钱冠群 李 琳
(北京航空航天大学软件工程研究所 北京 100191)

摘 要 软件自身的复杂性和未来变更需求的不确定性使得软件的稳定性评估十分困难.文中将软件的变更需求看成一系列"原子变更需求"的叠加,把"原子变更需求"的响应过程抽象成初始变更节点的随机选择过程以及由此引起的涟漪效应,提出了基于变更传播仿真的稳定性评估方法SEMCIS,定义了变更的传播模型和评价指标,并且给出了一种基于变更传播仿真的指标计算方法.实验结果表明,降低传播概率、改善软件的体系结构设计可以有效地抵御"涟漪效应"的发生,提高软件的稳定性.入度Hub的相互连接是造成大范围变更的主要原因,而入度Hub中存在的大量后继节点,则是造成大范围变更频繁发生的主要原因.

关键词 软件稳定性;变更分析;涟漪效应;软件维护;复杂网络
中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2010.00440

Software Stability Analysis Based on Change Impact Simulation

ZHANG Li QIAN Guan-Qun LI Lin
(*Software Engineering Institute*, *Beihang University*, *Beijing* 100191)

Abstract It's difficult to evaluate the software stability because of the complexity of software and the uncertainty of future change requirements. In this paper, various change requirements are regarded as the combination of a series of "atomic change requirement". The modification of software, which is used to satisfy the "atomic change requirement", is regarded as: firstly, modify a randomly selected "initial element"; secondly, a ripple effect caused by the change of it. Then we proposed a software stability evaluation method based on change propagation simulation. A change propagation model and a set of change impact metrics are defined. In order to simplify the calculation of the metrics, simulation technology is introduced into software stability evaluation instead of the conditional probability calculation. The experiments result indicates that decreasing the propagation probability or improving software architecture could effectively resist the happening of broad ripple effect and enhance software stability. And the connections between in-degree hubs result in broad change. The existence of a mount of successors of in-degree hubs results in the frequent happening of broad changes.

Keywords software stability; impact analysis; ripple effect; software maintenance; complex network

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.软件的易变性和不可预测性已经成为软件工程研究人员的共识.在软件维护的过程中,我们经常会遇到这样的情况:对软件中一个实体(例如类、方法)的修改,往往会影响到直接或者间

收稿日期:2009-04-20;最终修改稿收到日期:2009-10-14.本课题得到国家自然科学基金(60773155)、国家"九七三"重点基础研究发展规划项目基金(2007CB310803)资助.**张 莉**,女,1968 年生,博士,教授,博士生导师,主要研究兴趣为软件工程、软件体系结构.E-mail: lily@buaa.edu.cn. **钱冠群**(通信作者),男,1978 年生,博士研究生,主要研究兴趣为软件体系结构、逆向工程.E-mail: qianguanqun@cse.buaa.edu.cn. **李 琳**,女,1984 年生,博士研究生,主要研究兴趣为软件体系结构、软件度量.E-mail: lin.li@cse.buaa.edu.cn.

接相关的其它实体，从而引发一系列修改. 这种现象被称为“涟漪效应”(ripple effect)[1]. 为了确定软件的局部变更对系统其它部分的潜在影响，研究人员提出了一系列变更影响分析(change impact analysis)方法，也称为影响分析方法[2]. 希望借此为软件的变更规划、变更决策以及变更预测提供支持.

随着软件体系结构研究的深入，人们逐渐意识到：软件结构是决定软件质量的重要因素. 提出用稳定性(stability)来描述软件对变更过程中潜在“涟漪效应”的抵抗能力[3]. 通过总结以往的软件工程实践经验，研究人员还提出了一系列设计原则和设计方法，例如高内聚低耦合原则、开放封闭原则、软件体系结构模式、设计模式等等. 目的就是希望通过复用专家的设计经验，提高软件对变更的适应能力. 合理运用这些原则和方法设计并开发的软件系统，往往具有良好的可维护性. 但是这些方法主要侧重定性分析，没有给出定量的度量指标和评价方法.

本文基于软件的静态结构，提出了一种基于变更传播仿真的稳定性评价方法 SEMCIS(Stability Evaluation Method Based on Change Impact Simulation). 针对变更需求的多样性和粒度差异，SEMCIS 方法定义了“原子变更需求”，将软件的变更需求看作是一系列“原子变更需求”的叠加，提出用软件在响应“原子变更需求”时修改的实体数量的期望值来评价软件的稳定性. 并且将仿真方法引入到度量指标的计算中，克服了使用概率论方法计算度量指标时，处理循环依赖比较繁琐的缺陷. 在此基础上，本文以开源软件 JEdit① 为例，分析了影响软件稳定性的关键因素. 在验证 SEMCIS 方法有效性的同时，总结出“传播概率”和软件结构对软件稳定性的影响以及造成大规模变更的结构因素.

本文第 2 节介绍相关研究；第 3 节介绍基于变更传播仿真的稳定性评价方法，包括变更传播模型、度量指标和基于仿真的度量方法；第 4 节介绍后续实证分析的数据来源和参数设置；第 5 节结合开源软件实例，分析评价指标在仿真计算过程中的收敛性；第 6、7 节分别讨论传播概率、软件结构对软件稳定性度量指标的影响；最后在第 8 节对全文进行总结.

2　相关研究

Tsantalis 等人使用概率论方法评估面向对象系统的适应性[4-5]. 将软件变更细分成内部变化和外部变化. 内部变化只修改类本身，但不影响其它类；而外部变化表示在修改类的同时，要影响其它类. 在给定内部变化概率和外部变化概率的前提下，Tsantalis 等人提出用联合概率来计算每个类发生变化的概率. 在此基础上，Sharafat 等人[6]根据软件结构和演化历史，进一步提出了内部变化概率和外部变化概率的计算方法，并对预测结果进行了分析. Mirarab 等人[7]则提出，根据软件的静态依赖关系和历史维护信息，用贝叶斯网来构造软件的变化传播模型. 但是，软件的静态结构中普遍存在着双向依赖和循环依赖[8-9]，基于贝叶斯网的变更传播模型无法表达软件实体间的循环依赖关系. 同时，循环依赖也给上述方法的度量指标计算带来了困难. 假设我们要度量软件中每个类发生变更的概率，而类 A 和类 B 之间存在循环依赖，想要知道类 A 的变更概率，必须首先计算类 B 的变更概率，反之亦然. 为了解决这个问题，人们或者采用破环操作，通过删除若干条边来消除模型中的循环依赖；或者采用近似的方法，计算度量指标[5-6]. 这些方法在一定程度上破坏了原有的传播模型，影响了稳定性指标的计算结果. 在本文提出的 SEMCIS 方法中，我们忽略实体内部的实现细节，只考虑实体之间外部变化的传播关系，并且将仿真技术引入到度量指标的计算中，在模拟变更传播过程的基础上，通过统计得到软件的稳定性度量值.

Hassan 等人[10]则提出用启发式算法来预测变更的传播，借此指导开发人员进行软件的变更维护. Zimmermann 等人[11]运用数据挖掘方法研究了演化过程中软件实体之间的协同变更关系. 利用该方法可以给出某个实体发生变更时，需要协同变更的候选实体. 这些研究关注的是软件中实体之间的协同变更关系，本文主要讨论软件整体的稳定性度量问题.

Liu[12]等人则提出用传播代价来衡量软件结构的传播特性. 所谓传播代价，就是从软件的任意实体出发，所有可达实体占软件总体比例的期望值. 这和本文的工作非常相似. 所不同的是，在 Liu 等提出的传播模型中，变更的传播方向与依赖关系的方向一致，并且只考虑传播的步长，没有涉及到传播的不确定性问题. 本文在传播模型、稳定性度量方法上与 Liu 等人的工作存在较大的差异.

① JEdit. http://www.jedit.org/

3 SEMCIS 方法

在软件的整个生命周期中，可能遇到各种各样的变更需求，例如追加功能、修改 Bug、迁移平台、升级第三方运行库等等. 变更需求的不确定性以及类型、粒度和内容的差异，给软件的稳定性度量带来了极大的困难.

对于小粒度的“原子变更需求”，需要修改的软件实体一般彼此关联，而且局限在一个有限的范围之内. 而不同粒度、不同类型、不同内容的“变更需求”可以看作是一系列“原子变更需求”的叠加. 如果我们把软件中的实体作为节点，它们之间的依赖关系看成边，软件就被抽象成一个网络. 软件变更中的“涟漪效应”就可以看成是变更在网络节点间的传播过程. 而“原子变更需求”对应的响应过程可以看作是：“初始变更节点”v_{init}发生改动以及由于修改 v_{init} 而引发的涟漪效应. 这种节点间的涟漪效应存在不确定性，也就是说，节点 v_i 发生变更时，和它相邻的节点 v_j 是否需要修改是不确定的，本文用“传播概率”进行刻画. 这种给定节点间“传播概率”的软件网络模型，我们把它叫做软件的“变更传播模型”. 对于一个具体的软件来说，未来可能发生的“原子变更需求”也是不确定的. 对应到软件网络中，就是“初始变更节点”v_{init}的随机选择过程. 通过度量软件网络在响应不同“原子变更需求”时，需要修改的节点数的期望值，就可以在一定程度上刻画软件网络的变更传播特性，即稳定性. 这里，我们首先给出“原子变更”、“原子变更需求”、“初始变更节点”、“传播概率”的形式化定义.

定义 1. 在软件的一次变更中，所有被修改的软件实体以及它们之间的依赖关系构成一个有向简单图 $G=(V,E)$，其中，V 称为“变更节点集”，V 中的每个节点 v_i 表示一个变更的软件实体，E 是边的集合，E 中的每个元素 $\langle v_i,v_j\rangle$ 是一个有序对，当且仅当 v_i 使用了 v_j 提供的服务时，$\langle v_i,v_j\rangle\in E$. 如果存在一个节点 v_{init}，G 中的其它节点到 v_{init} 可达，我们把这样的变更称为“原子变更”，它所对应的需求叫做“原子变更需求”，节点 v_{init} 叫做“初始变更节点”.

定义 2. 在软件的一次变更中，由于节点 v_j 发生变更，使得直接依赖 v_j 的相邻节点 v_i 发生变更的条件概率，称为节点 v_j 相对 v_i 的“传播概率”，记作 p_{ji}.

从定义 2 中可以看出，传播概率的取值满足 $0\leq p_{ji}\leq 1$. 在真实的软件中，实体之间的传播概率需要分析大量的维护历史数据之后才能给出. 在缺少历史数据的情况下，例如在软件设计的评估阶段，传播概率的取值可以通过专家的经验进行设置.

根据软件的变更传播模型计算稳定性度量指标的另一个问题是，由于软件网络中存在大量的有向环[8-9]，经典的概率论方法无法直接计算出变更节点数的期望值. 为此，以往的研究都是先设法消除网络中的环型结构[5-6]. 但是，这种破环操作在一定程度上破坏了原有的传播模型，影响了稳定性指标的计算结果. 为了解决度量指标的计算问题，本文采用仿真技术来模拟“原子变更”过程. 在此基础上，通过统计多次仿真中变更节点数的平均值给出软件的稳定性度量值.

本文提出的基于变更传播仿真的稳定性评价方法 SEMCIS 如图 1 所示，SEMCIS 的评估流程分为 3 步：

(1) 根据软件的体系结构设计、详细设计或者源代码等，生成软件的变更传播模型；

(2) 仿真软件的变更传播过程，并记录仿真过程中的变更节点数；

(3) 统计仿真数据，给出软件稳定性评价指标的度量值.

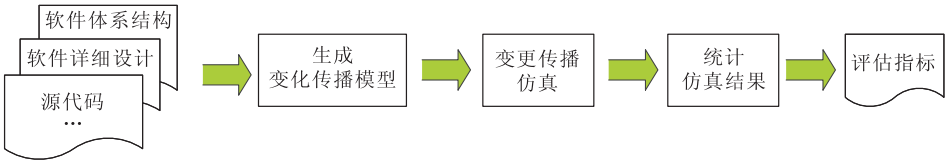


图 1 评估流程示意图

以下分小节详细阐述 SEMCIS 方法中的变更传播模型、评价指标和变更传播的仿真算法.

3.1 变更传播模型

定义 3. 软件的变更传播模型表示为一个带权有向简单图 $G_w=(V,E)$. 其中，V 表示节点的集合，V 中的每个节点 v_i 表示软件中的一个实体；E 是边的集合，E 中的每个元素 $\langle v_i,v_j\rangle$ 是一个有序对，当且仅当 v_i 使用了 v_j 提供的服务时，$\langle v_i,v_j\rangle\in E$. 有

向边〈v_i，v_j〉上的权重表示为节点 v_j 相对 v_i 的传播概率 p_{ji}.

图 2 给出了一段 Java 源代码和它的变更传播模型. 其中，v_1～v_4 表示 Java 代码中的 4 个类，有向边上的权重表示变更传播概率. 例如，有向边〈v_4，v_2〉上的 p_{24} 就表示 v_2 发生变更时，v_4 发生变更的概率.

```
public class v1 {
    int x;
};
public class v2 extends v1 {
    void m2() {
    }
}
public class v3 extends v1 {
    void m3(v1_a, v2_b) {
    }
}
public class v4 {
    void m4(v2_a) {
    }
};
```

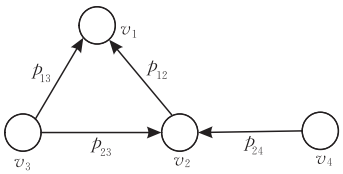


图 2　软件的变更传播模型实例

3.2　评价指标

定义 4.　变更节点数表示为

$$NOCN=|\psi| \tag{1}$$

$NOCN$ 表示为了响应某次变更需求，最终修改的节点总数，ψ 表示在响应变更时实际修改的节点集合. $|\Omega|$ 表示集合 Ω 包含的元素个数，下同.

定义 5.　平均变更节点数表示为

$$AvgNOCN=\frac{\sum_{i=1}^{n}NOCN_i}{n} \tag{2}$$

其中，n 表示变更次数. $AvgNOCN$ 刻画的是在 n 次变更中，需要修改的平均节点数.

定义 6.　变更节点数的标准差表示为

$$StdNOCN=\sqrt{\frac{\sum_{i=1}^{n}(NOCN_i-AvgNOCN)^2}{n-1}} \tag{3}$$

$StdNOCN$ 反映的是，多次变更中修改的节点数之间的差异程度.

定义 7.　最小变更节点数表示为

$$MinNOCN=\min\{NOCN_i\} \tag{4}$$

$MinNOCN$ 表示的是，多次变更中最少需要修改的节点数.

定义 8.　最大变更节点数表示为

$$MaxNOCN=\max\{NOCN_i\} \tag{5}$$

$MaxNOCN$ 表示的是，多次变更中最多需要修改的节点数. 它反映了最差的变更状况.

定义 9.　变更节点比率表示为

$$POCN=\frac{NOCN}{|V|}\times 100\% \tag{6}$$

其中，$|V|$ 表示节点总数，下同. $NOCN$ 反映的是变更节点数的绝对值，它与软件的规模相关. $POCN$ 反映的则是变更节点数相对整体的比例. 在对比不同规模软件的稳定性时，基于 $POCN$ 的比例度量更能反映软件的稳定性质量.

定义 10.　平均变更节点比率表示为

$$AvgPOCN=\frac{AvgNOCN}{|V|}\times 100\% \tag{7}$$

定义 11.　平均变更节点比率的标准差为

$$StdPOCN=\sqrt{\frac{\sum_{i=1}^{n}(POCN_i-AvgPOCN)^2}{n-1}} \tag{8}$$

定义 12.　最小变更节点比率表示为

$$MinPOCN=\min\{POCN_i\} \tag{9}$$

定义 13.　最大变更节点比率表示为

$$MaxPOCN=\max\{POCN_i\} \tag{10}$$

3.3　基于仿真的评价指标计算方法

软件网络的结构研究表明，软件中存在双向边和大量的有向环结构[8-9]，这使得传统的概率论方法无法根据软件的变更传播模型直接进行评价指标的计算. 在以往的研究中，人们一般采用破环性操作删除部分有向边，消除模型中的有向回路，再计算度量指标；或者直接采用近似方法来计算评价指标[5-6]. 这些方法都在一定程度上破坏了原始的变更传播模型，进而影响了评估和预测的精度.

为了避免上述缺陷，简化评价指标的计算过程，本文提出一种基于仿真的变更传播算法. 该算法的核心思想是：首先，随机选择一个初始变更节点 v_{init}；然后，顺着输入边方向，以边上的传播概率逐个影响相邻的未变更的节点；接着，按照广度优先、逐个波及的原则，从新变更的节点出发，依次迭代，向外传播，以此来模拟“原子变更”的执行过程，即对“原子变更需求”的响应过程，直到没有新的节点发生变更. 这时，一次仿真结束. “原子变更”中“变更节点集”到初始变更节点可达的特性，保证了变更传播仿真的可行性. 此外，需要特别指出的是：在“原子变更”中，我们假设每个节点最多只需要修改一次就能满足“原子变更需求”，也就是说，每个节点最多只能变更一次；而每个未变更节点却可以受到相邻多个新变更节点的影响. 每次仿真的结果对应一次“原子变更”，“原子变更”中修改的节点总数记做 $NOCN$. 经过多次仿真后，统计变更节点数的均值、方差、最大值和最小值，就得到该软件的稳定性度量指标.

当变更传播模型中存在循环依赖时，不妨假设类 A、类 B 双向依赖，由于引入了广度优先、逐个波

及的传播准则，在一次仿真中，类 A 和类 B 只能有一个因、一个果，假设类 A 的变更引发类 B 的修改；但在下一次仿真中，由于选择的初始变更节点不同以及变更传播的不确定性，类 B 的变更也可能引发类 A 的修改. 特别的，当我们选择类 A 为初始变更节点时，类 B 的变更必定是由类 A 引起的；同理，当我们选择类 B 为初始变更节点时，类 A 的变更也必然是类 B 引起. 由此可见，一方面，软件结构中存在的循环依赖不会影响变更传播的仿真；另一方面，通过多次仿真，我们可以模拟出循环依赖的两个节点在变更时互为因果的现象. 因此，本文提出的基于仿真的评价指标计算方法，不但保持了传播模型与软件静态结构的一致性，而且简化了度量指标的计算过程.

以下详细介绍基于仿真的评价指标计算方法.

设某个软件的变更传播模型为 $G_w=(V,E)$，节点 v_i 被选为变更初始节点 v_{init} 的概率为 $q(v_i)$. 那么，模型中所有节点被选为 v_{init} 的概率必然满足下面的约束条件：

$$\sum_{v_i\in V} q(v_i)=1,\ 0\leq q(v_i)\leq 1 \tag{11}$$

仿真算法分为以下几个步骤：

1. 输入软件的变更传播模型 $G_w=(V,E)$ 和仿真次数 N，当前仿真次数 n 设为 0；

2. 设置模型中所有节点的变更状态为“未变更”；

3. 以概率 $q(v_i)$ 随机选择一个节点作为初始变更节点 v_{init}，设置节点 v_{init} 的变更状态为“已变更”，并将节点 v_{init} 加入到变更队列 ChangedNodeQueue 中；

4. 选择 ChangedNodeQueue 中的首节点 v_s，获取 v_s 的所有前驱节点（即存在指向节点 v_s 的边的节点）中变更状态为“未变更”的节点集合，设为 R；

5. 对集合 R 中的每个节点 v_t，以传播概率 p_{st} 将其变更状态设置为“已变更”；如果 v_t 的变更状态从“未变更”变成了“已变更”，就将 v_t 添加到变更队列 ChangedNodeQueue 尾部；

6. 从 ChangedNodeQueue 中删除首节点 v_s，并且清空集合 R；

7. 反复执行步 4～6，直到 ChangedNodeQueue 为空，记录变更的节点总数为 $NOCN_n$；

8. 设置当前仿真次数 $n=n+1$；若 $n<N$，反复执行步 2～7，否则仿真停止.

图 3 是变更传播仿真算法的一个流程示意，图中○表示节点状态为“未变更”，●表示“已变更”. 假设在某次仿真中，v_1 被选为初始变更节点 v_{init}，

(a) v_1 首先被标记为“已变更”，并添加到队列 ChangedNodeQueue 中；ChangedNodeQueue 首节点 $v_s=v_1$，v_1 有两条输入边，分别连接 v_2 和 v_3，并且这两个节点都是未变更，因此 $R=\{v_2,v_3\}$；

(b) 分别以概率 p_{12} 和 p_{13} 波及节点 v_2 和 v_3，假设 v_2 被波及、v_3 不受影响，于是 v_2 被添加到 ChangedNodeQueue 队列中，并删除队列的首节点 v_1. 这时，ChangedNodeQueue 队列中只包含新被修改的节点 v_2；

(c) ChangedNodeQueue 队列首节点 $v_s=v_2$，v_2 同样有两条输入边，分别连接未变更节点 v_3 和 v_4，因此 $R=\{v_3,v_4\}$；

(d) 分别以概率 p_{23} 和 p_{24} 波及节点 v_3 和 v_4，假设 v_3 被波及、v_4 不受影响，于是 v_3 被添加到 ChangedNodeQueue 队列中，并删除队列的首节点 v_2. 这时，ChangedNodeQueue 队列中只包含新被修改的节点 v_3；

(e) ChangedNodeQueue 队列首节点 $v_s=v_3$，v_3 没有输入边，因此 R 为空；

(f) 由于 R 为空，因此没有波及操作，直接从 ChangedNodeQueue 中删除首节点 v_3，这时

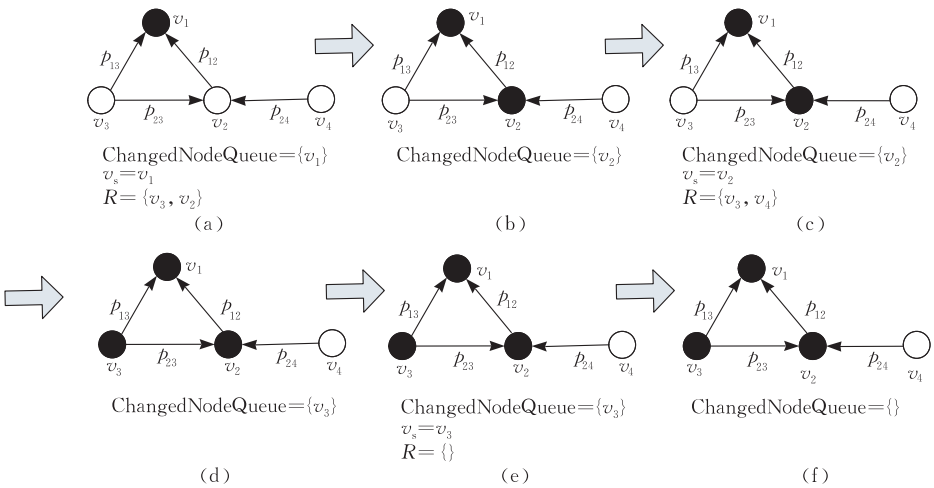


图 3 仿真算法示意图

ChangedNodeQueue也为空，本次仿真结束，变更节点数 $NOCN=3$，$POCN=0.75$.

在我们的传播算法中，不限制网络中节点进入集合 R 的次数. 在图 3 的实例中，v_3 两次进入集合 R，这与软件维护中的实际情况一致. 因为 v_3 是否变更，取决于所有后继节点（v_1 和 v_2）对它的影响.

从前面的仿真实例中可以看出，由于初始变更节点的选择以及变更节点对相邻节点的影响都存在不确定性，因此，多次仿真中 $AvgNOCN$、$StdNOCN$ 等评价指标的度量值也会有所不同. 但是随着仿真次数的增加，上述评价指标都会逐步收敛，最终保持稳定. 在本文的第 5 节，我们将结合软件实例，对仿真过程中度量指标的收敛性进行分析.

4　实验设计

为了验证 SEMCIS 方法的有效性，分析传播概率、软件结构对软件稳定性的影响，本文将结合开源软件 JEdit，进行实证分析. 在第 5 节，我们首先考察 SEMCIS 方法中，评价指标的收敛性问题；在第 6 节、第 7 节，我们分别考察传播概率以及软件结构对稳定性的影响. 通过实验结果分析，在验证 SEMCIS 方法的同时，分析和总结影响软件稳定性的因素，特别是造成大规模变更的原因，为软件设计、维护的决策提供支持.

4.1　数据来源

本文的实验数据来自开源 Java 软件 JEdit 4.2 的核心包 jedit.jar. 软件中的每个类对应变更传播模型中的一个节点，类和类之间的继承、组成、关联和依赖关系对应模型中的一条边. 如果两个类之间存在多种关系，也被看成是一条边. 本文选取其中的最大弱连通子图进行分析. 该子图共包括 796 个节点，3883 条边，节点的最大入度是 223.

4.2　实验设置

在本文提出的软件变更传播模型中，传播概率反映的是软件中相连两个实体之间耦合关系的强弱，而初始变更节点的选择概率反映的是“原子变更需求”的不确定性. 这些概率可以通过对以往的维护历史进行分析或者专家经验得到. 在 JEdit 的实证分析中，本文假设相邻两个类之间的传播概率都等于定值 α，即

$$p_{ji}=\begin{cases}\alpha, & \text{iff } \langle v_i,v_j\rangle\in E\\ 0, & \text{其它}\end{cases}.$$

每个节点被选为变更初始节点的概率服从均匀分布，即

$$p(v_i=v_{\text{init}})=\frac{1}{|V|},\ \text{iff } v_i\in V.$$

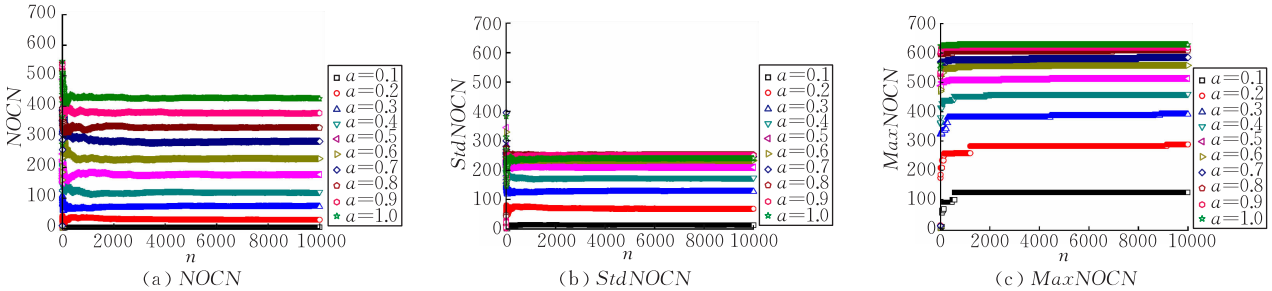
5　评价指标的收敛性分析

在变更传播的仿真算法中，初始变更节点是随机选择的，并且使用传播概率来控制相邻节点间变更传播的可能性. 因此，在两次仿真中，即使 α 相同，$AvgNOCN$、$StdNOCN$ 等指标的仿真结果也会有所不同. 要想运用仿真算法评价软件的变更传播特性，首先需要考察不同传播概率 α 下评价指标的收敛性.

在本文提出的评价指标中，$AvgNOCN$、$StdNOCN$、$MinNOCN$ 和 $MaxNOCN$ 分别与 $AvgPOCN$、$StdPOCN$、$MinPOCN$ 和 $MaxPOCN$ 成正比，而 $MinNOCN$ 一般为 1. 所以，在这里，我们只考察 $AvgNOCN$、$StdNOCN$ 和 $MaxNOCN$ 3 个评价指标的收敛性.

实验中，传播概率 α 分别取 0.1～1（间隔 0.1），仿真总次数 N 设为 10000.

图 4 给出了不同传播概率 α 下，平均变更节点数 $AvgNOCN$、变更节点数的标准差 $StdNOCN$ 和最大变更节点数 $MaxNOCN$ 在仿真中的变化趋势. 其中，X 轴表示仿真次数 n，Y 轴分别表示 n 次仿真后的平均变更节点数 $AvgNOCN$、变更节点数的标准差 $StdNOCN$ 和最大变更节点数 $MaxNOCN$. 图中的每条曲线分别表示不同 α 下的仿真结果.



(a) $NOCN$　(b) $StdNOCN$　(c) $MaxNOCN$

图 4　仿真过程中 JEdit 软件网络的评价指标变化趋势

从变化趋势图上可以看出：当仿真次数 $n<1000$ 时，$AvgNOCN$、$StdNOCN$ 和 $MaxNOCN$ 曲线波动明显，这表明仿真中由于选择了不同的初始变更节点，需要更改的节点数 $NOCN$ 之间存在巨大的差异；当仿真次数 $n\geq 3000$ 时，$AvgNOCN$、$StdNOCN$ 和 $MaxNOCN$ 基本保持稳定. 由此可见，在仿真次数 N 远大于节点数的前提下，本文提出的评价指标在仿真中具有良好的收敛性.

对 JEdit 进行多次实验后我们发现，评价指标在多次实验中的度量值基本稳定，偏差不超过度量值的 1%，说明本文提出的变更仿真算法是有效的，可以用于软件稳定性的分析和度量.

6 传播概率对软件稳定性的影响分析

6.1 对评价指标的影响

以往的研究表明，模块间的耦合程度很大程度上影响到系统的可维护性. 因此，在软件设计时，人们提倡模块化的设计思想，目的是希望软件系统各模块之间的耦合尽可能松散，联系尽可能简单. 这样，在进行软件维护时，不但可以减少变更的扩散范围，而且可以降低理解系统的难度. 在本文提出的 SEMCIS 方法中，变更传播概率在一定程度上反映了模块间的耦合强度. 原则上，传播概率越高，变更的平均影响范围就越大，软件的稳定性越差.

表 1 给出了不同传播概率下的评价指标. 可以看到：随着传播概率的不断增加，最大变更节点数 $MaxNOCN$ 和平均变更节点数 $AvgNOCN$ 不断增加，平均变更节点数的标准差 $StdNOCN$ 也基本呈上升趋势（α 介于 0.8～1.0 时略有不同）. 另外，当 $\alpha<0.8$ 时，平均变更节点数的标准差 $StdNOCN$ 都要大于平均变更节点数. 这说明，尽管变更模型中的传播概率较小，但是在响应不同的“原子变更需求”时，由于选择的初始变更节点不同，需要更改的节点数 $NOCN$ 之间存在巨大的差异.

表 1 不同传播概率下 JEdit 软件网络的变更传播指标

α	$AvgNOCN$	$StdNOCN$	$MaxNOCN$	$AvgPOCN$	$StdPOCN$	$MaxPOCN$
0.1	4.23	12.87	124	0.53	1.62	15.58
0.2	28.91	68.12	288	3.63	8.56	36.18
0.3	71.91	127.31	391	9.03	15.99	49.12
0.4	120.13	173.84	460	15.09	21.84	57.79
0.5	177.05	209.49	513	22.24	26.32	64.45
0.6	228.62	233.25	558	28.72	29.30	70.10
0.7	285.51	246.71	585	35.87	30.99	73.49
0.8	330.00	252.26	608	41.46	31.69	76.38
0.9	377.91	250.10	619	47.48	31.42	77.76
1	424.99	240.08	628	53.39	30.16	78.89

由于传播概率在一定程度上反映了软件元素之间的耦合程度，上述实验结果表明：降低耦合度对软件维护具有重要意义. 过高的耦合性必将导致软件变更时的连锁效应，增加软件的维护成本. 这和软件设计的基本原则是一致的. 也从另一个侧面验证了本文提出的 SEMCIS 方法，可以有效地度量不同耦合强度对软件稳定性的影响.

6.2 对变更节点数分布的影响

平均变更节点数 $AvgNOCN$ 和标准差 $StdNOCN$ 反映了软件在响应“原子变更需求”时的整体统计特性. 那么，多次仿真中变更节点数 $NOCN$ 是否存在某种规律呢？

图 5 给出了不同传播概率下，变更节点数的分布情况. 其中，X 轴表示变更的节点数，Y 轴表示在 N 次仿真中出现 x 个节点发生变更的次数. X 轴和 Y 轴都以 10 为底取对数. 从中我们发现一些有趣的现象：

(1) 当 $\alpha=0.1$ 时，变更节点数呈现出明显的幂率分布. 在尾部，发生大范围变更的次数呈上升趋势；

(2) 当 $\alpha=0.2$～0.9 时，变更节点数的分布明显分裂成两个区域，左侧基本服从幂率或者带指数截断的幂率分布，右侧是一个或两个山峰，两者之间存在明显的断裂带. 随着 α 的增加，右侧的山峰逐步上升并右移，说明软件在响应未知的变更需求时，可能变更的节点数越来越多、发生的几率也越来越高. 从 $\alpha=0.4$ 开始，山峰由一个分裂成两个.

(3) 当 $\alpha=1.0$ 时，变更节点数的分布分裂为两个区域，但是每个区域的概率分布情况不是很明显.

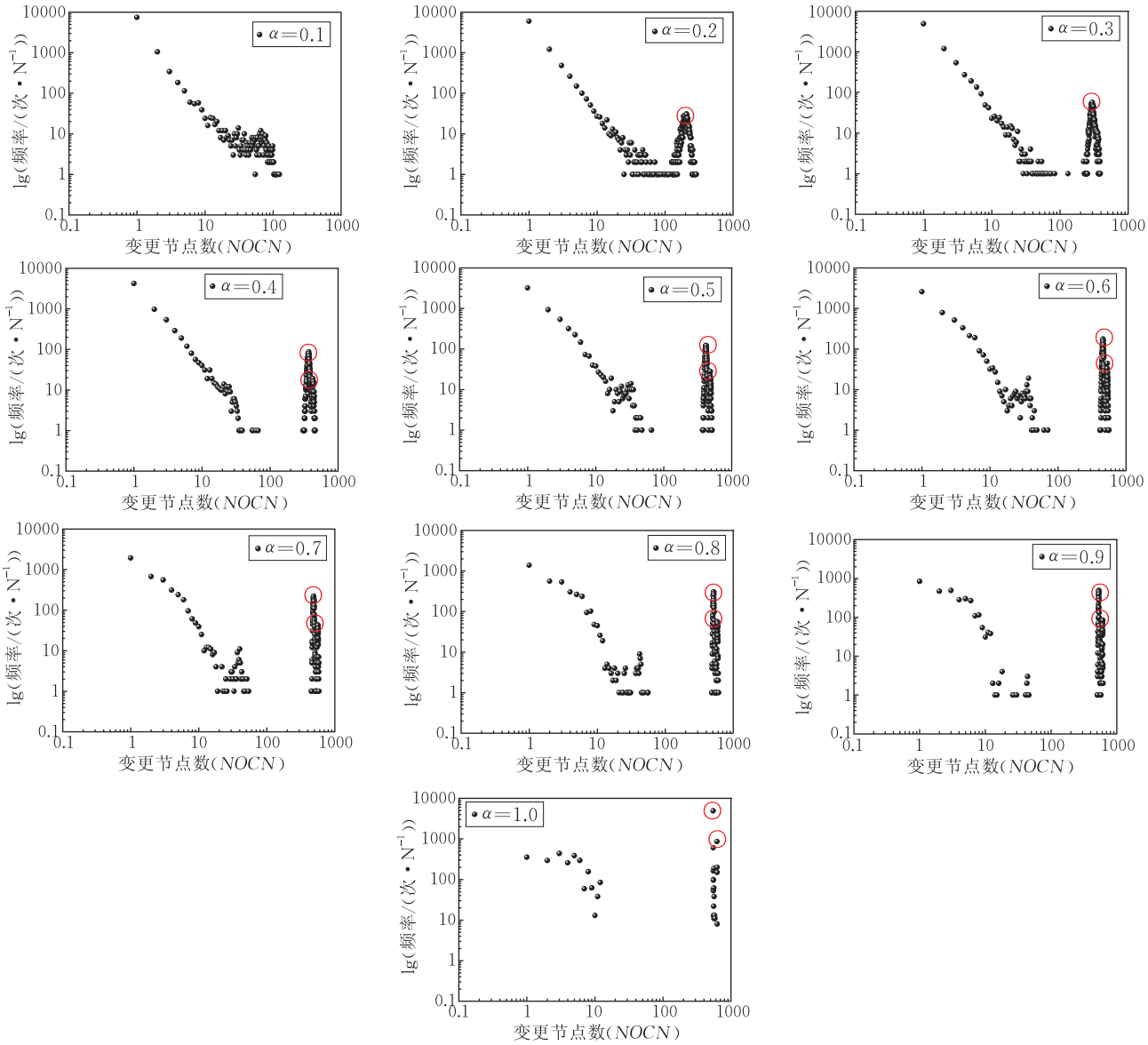


图 5　不同传播概率下 JEdit 软件的变更节点数分布(○表示峰值所在位置)

从这些现象中我们可以看出，每次仿真中对应的变更节点数是极不均匀的，变更节点数的分布图从一定程度上反映了软件响应未知变更的能力. 随着传播概率的增加，发生大规模变更的可能性不断上升. 因此，改善类与类之间的耦合强度，降低传播概率，可以有效地减少原子变更的波及范围，提高软件稳定性.

7　软件结构对软件稳定性的影响分析

以往的研究表明，软件的体系结构在很大程度上决定了软件的全局质量. 其中，软件可维护性与软件的结构有着直接的关系[13]. 由此可见，在传播概率不变的前提下，改善软件的体系结构设计，也就是软件实体之间的连接关系，有可能提高软件的稳定性. 那么，什么样的软件结构，它的稳定性比较好？影响稳定性的结构因素又有哪些呢？

为了研究软件结构对稳定性的影响，我们首先将 JEdit 的变更传播模型(简称真实模型)与相同规模、随机化之后的拓扑结构(简称随机模型)进行对比，分析它们在稳定性上的差异. 在此基础上，对大规模变更产生的原因进行分析.

7.1　软件真实模型与随机模型的稳定性对比

实验分为以下几个步骤：

1. 输入软件的变更传播模型 $G_w=(V,E)$，获取每个节点的出度和入度，组成度序列；

2. 根据步 1 中得到的度序列，运用交换法则生成一系列随机模型[14-15]. 针对真实软件中的有向边 $v_i\rightarrow v_j$ 和 $v_s\rightarrow v_t$，如果 $v_i\rightarrow v_t$ 和 $v_s\rightarrow v_j$ 都不存在，那么删除这两条边，然后

添加 $v_i \rightarrow v_t$ 和 $v_s \rightarrow v_j$ 到网络中. 同时要求,双向边只能和双向边进行交换,随机化后的网络必须保证连通性. 这样,通过多次交换后生成的随机模型,每个节点的度和原始软件保持一致,但是,节点与节点间的度相关性发生了变化.

3. 设置相同的仿真次数 N 和变更传播概率 α,对真实模型和随机模型进行仿真,并统计度量指标.

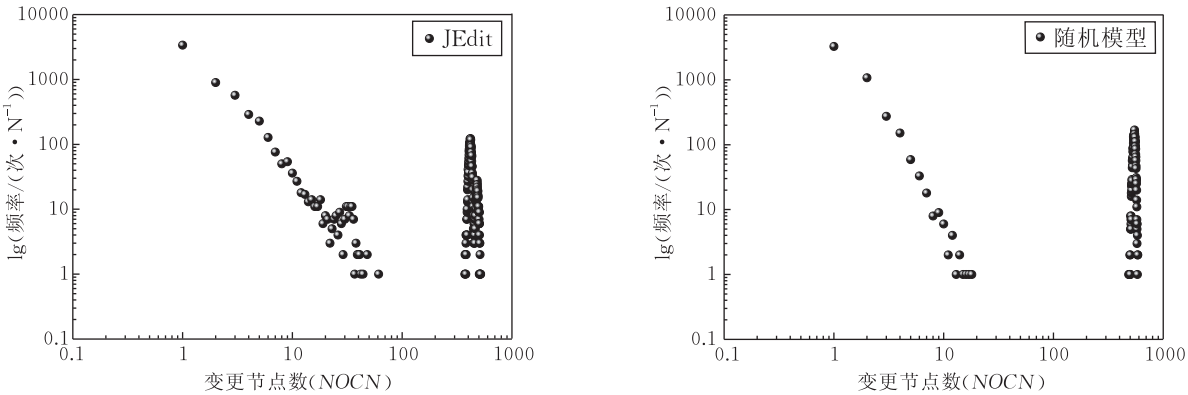
本实验中,传播概率 α 设为 0.5,仿真次数 N 设为 10000. 图 6 给出 JEdit 真实模型和随机模型的变更节点数分布图. 其中 X 轴表示变更的节点数,Y 轴表示在 N 次仿真中,出现 x 个变更节点的次数. X 轴和 Y 轴都以 10 为底取对数. 具体的统计指标参见表 2. 结合图 6 和表 2 的信息,可以发现:

(1) JEdit 真实模型的评价指标,包括 *AvgNOCN*、*StdNOCN*、*MaxNOCN* 在内,都要明显优于随机模型.

(2) 与 JEdit 真实模型类似,随机模型的变更节点数分布同样分裂成两个区域,左侧基本服从幂率或者带指数截断的幂率分布,右侧同样出现山峰,两个区域之间也存在明显的断裂带.

(3) JEdit 网络的左半部分比随机模型的下降趋势要慢一些,这使得左侧部分的最大值要远远高于随机模型. JEdit 网络左侧部分的最大值是 61,而随机模型左侧部分的最大值只有 20 左右. 但是,JEdit 左半部分在总体中所占的比例要明显高于随机模型. 在随机模型中,左侧部分的频数总和为 4904 次,而在 JEdit 软件的变更节点数分布图中,左侧部分的频数总和达到了 5977 次.

(4) JEdit 真实模型的右半部分在 X 轴上的跨度较大,最小值和最大值都要明显小于随机模型. 随机模型右半部分的变更节点数分布在(484,586)之间,而 JEdit 真实模型则在(373,518)之间变化.



(a) JEdit真实模型 (b) 随机模型

图 6 $\alpha=0.5$ 时 JEdit 真实模型和随机模型的变更节点数分布

表 2 $\alpha=0.5$ 时,JEdit 软件真实模型和随机模型的变更传播指标对比

	AvgNOCN	*StdNOCN*	*AvgPOCN*/%	*StdPOCN*/%	*MinNOCN*(左侧)	*MaxNOCN*(左侧)	*MinNOCN*(右侧)	*MaxNOCN*(右侧)
JEdit	174.14	209.37	21.88	26.30	1	61	373	518
随机模型	278.17	271.51	34.95	34.11	1	18	484	586

实验结果表明,本文提出的 SEMCIS 方法和评价指标可以反映出结构对稳定性的影响. 经过人工设计的软件,它的稳定性确实要好于随机模型,可以有效地降低变更的传播范围. 在变更传播概率不变的前提下,改善软件的体系结构设计同样可以提高软件的稳定性.

7.2 造成大规模变更的结构因素

从软件真实的变更传播模型与随机模型的稳定性对比实验中,我们发现:无论是 JEdit 的真实模型,还是随机模型,仿真过程中大规模变更频频发生. 这说明,JEdit 真实模型和随机模型在存在差异的同时,也存在一些共性,而这些共性就是造成大规模变更的主要原因. 要想抑制变更的传播范围,有必要研究造成大规模变更的结构特点,从而为软件的体系结构设计提供指导.

直观上看,修改初始变更节点 v_{init} 导致的变更节点数 $NOCN(V_{\text{init}}) \approx \sum_{i=0}^{\infty} \alpha^i \times n_i(v_{\text{init}})$,其中 $n_i(v_{\text{init}})$ 表示到 v_{init} 的最短距离为 i 的前驱节点个数. 在 α 较小的前提下,随着距离 i 的增加,$\alpha^i \ll \alpha$,因此 $NOCN(v_{\text{init}}) \approx 1+\alpha \times n_1(v_{\text{init}}) = 1+\alpha \times k_{\text{in}}(v_{\text{init}})$,其中 $k_{\text{in}}(v_{\text{init}})$ 表示 v_{init} 的入度. 当传播概率 α 较小时,节点 v_{init} 的修改一般只能影响到与 v_{init} 直接相连的节点. 也就是说,初始变更节点的入度很大程度上决定了最终变更的节点数. 而软件网络的入度基本服从幂率分布[16],所以变更的波及范围基本上和初始

变更节点的入度成正比，也呈现出类似幂率的分布.

但是，从表 1 的 JEdit 仿真结果来看，不同 α 下最大的变更节点数 $MaxNOCN$ 远远大于 $1+\alpha\times\max(k_{in})$. 而且，随着 α 值的增加，最大变更节点数 $MaxNOCN$ 也随之增加. 以 $\alpha=0.1$ 为例，最大变更节点数 $MaxNOCN$ 达到了 124，远远大于 $1+\alpha\times\max(k_{in})=1+0.1\times223=23.3$（JEdit 网络中节点的最大入度是 223）. 要想造成这么大范围的变更，仅靠 1 个入度较大的集散节点（简称为入度 Hub）是远远不够的，必须有多个入度 Hub 相互协作. 首先，多个入度 Hub 节点受到 v_{init} 的影响发生变更，再通过入度 Hub 引起大规模的涟漪效应. 因此，入度 Hub 之间的彼此相连是造成大规模变更，特别是最大变更节点数 $MaxNOCN$ 偏大的主要原因.

随着传播概率的增加，初始变更节点可以波及的距离逐渐增加. 与此同时，能够影响入度 Hub 的节点个数也不断增加. 如图 7 所示，当 α 值很小时，入度 Hub 节点 v_{Hub} 一般只能受到距离为 1 的 v_1、v_2 等节点的影响，同时只能影响 v_5、v_6 等节点. 随着 α 值的增加，v_{Hub} 一旦发生变更，可能波及的节点越来越多. 于此同时，v_{Hub} 被距离大于 1 的节点（例如，v_3、v_4）影响的几率也不断上升. 这就使得：在变更波及范围不断增加的同时，剧烈变更的发生频率也不断上升. 表现在变更分布图上，就是右侧的山峰不断右移、上升，山峰在总体中所占的比重逐渐增加.

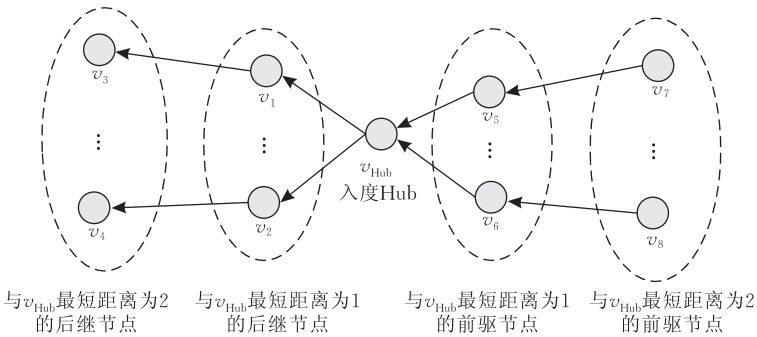


图 7　入度 Hub 节点的影响示意图

通过上面的分析，我们可以得出这样的结论：入度 Hub 的相互连接是造成大范围变更的主要原因；而入度 Hub 中存在的大量后继节点（$v\leftarrow v_{Hub}$），则是造成大范围变更频繁发生的主要原因.

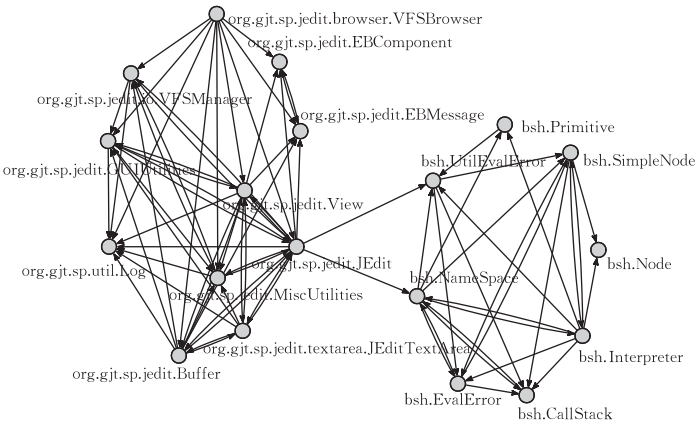
为了验证前面的分析，本文以 JEdit 网络为例，分析入度 Hub 节点的连接方式以及它们的前驱和后继节点对网络传播特性的影响. 首先我们给出入度 Hub 的定义.

定义 14.　入度 Hub 的阈值为

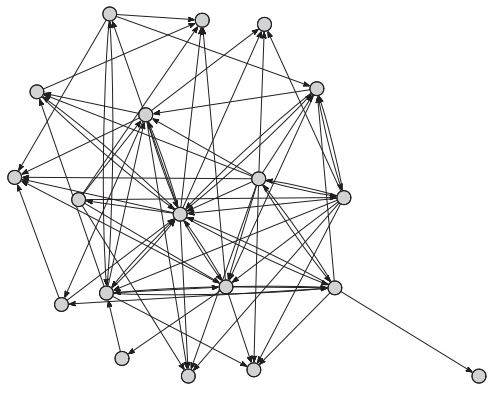
$$\lambda=\langle k_{in}\rangle+2\times std(k_{in}) \tag{12}$$

其中，$\langle k_{in}\rangle$ 和 $std(k_{in})$ 分别表示网络中节点入度的平均值和标准差. 入度大于 λ 的节点都被看作是入度 Hub 节点. 对于 JEdit 网络，节点入度的平均值为 4.9，方差为 13.7，因此入度大于 32.3 的节点都被看作入度 Hub 节点，共计 19 个.

图 8 给出了 JEdit 网络和随机模型中入度 Hub 节点的连接关系图. 从中可以看到，在 JEdit 软件中，入度 Hub 节点形成一个弱连通网络，并且被划



（a）JEdit真实模型



（b）随机模型

图 8　JEdit 真实模型和随机模型的入度 Hub 的连接方式对比

分成两个明显的社区.每个社区中的 Hub 节点连接异常紧密.任何一个节点发生变更,都可能波及到社区内的所有节点,造成大规模的涟漪效应.特别是当右侧社区的节点发生变更时,可能通过有向边影响到左侧社区中的节点,使变更的范围进一步扩大.JEdit 的 Hub 网络中存在的社区结构以及两个社区之间存在的单向连通性,可能是造成变更节点分布中出现两个波峰的原因.相比之下,随机模型中的入度 Hub 节点被紧密连接成一个整体,没有明显的社区结构.一旦某个入度 Hub 节点发生变更,其它节点更容易受到牵连,从而造成更大范围的变更.

表 3 给出了 JEdit 网络和随机模型中前驱和后继节点的个数.从中可以看出,与 JEdit 入度 Hub 直接连接的前驱节点个数达到了 501 个.而 JEdit 入度 Hub 的直接后继节点数达到了 230 个,这意味着网络中 28.9%的节点发生变更时,有可能以概率 α 波及到入度 Hub 节点.而多个入度 Hub 之间的紧密连接,就可能将这个影响扩散开来,造成大范围变更.相比之下,随机模型中的直接前驱和后继节点个数要高出真实模型 10%以上.此外,从 7.1 节随机模型的构造算法中可以看出,真实模型和随机模型具有相同的度序列.但是,真实软件中相邻节点的入度之间呈负相关性,而随机模型中相邻节点的入度之间不存在明显的相关性,所以在初始变更节点和传播概率相同的情况下,随机模型波及到入度 Hub 节点的几率更大.这和仿真中观测到的某些现象是一致的(JEdit 真实模型的右侧波峰相对靠左,而且高度相对较低).

表 3 JEdit 软件网络和随机模型中入度 Hub 的前驱/后继节点个数

	直接后继节点个数(含 Hub)	直接前驱节点个数(含 Hub)
JEdit	230	501
随机模型	254	567

另一方面,由于软件在设计过程中,往往采用逐层分解、高内聚低耦合的模块化设计思想,这使得软件中存在明显的模块结构.模块内实体的联系比较紧密,模块间实体的联系相对稀疏.当模块和模块之间进行交互时,一般要求通过特定的模块接口才能进行.当出现变更需求时,软件的这种模块结构可以有效地防止变更的扩散,使得变更被尽可能地限制在初始变更节点所在的模块内部.只有当公共元素(例如入度 Hub 或模块间接口)发生修改时,变更范围才会进一步扩散.

所以,在改进软件的变更传播特性时,不仅要从微观上控制软件实体和实体之间的耦合程度,提高软件实体特别是公共实体自身的封装性,更重要的是,要从宏观上调整软件的整体结构,特别是控制入度 Hub 之间的连接关系,尽可能避免它们紧密相连,使软件能更好地适应需求的变化,最终降低软件的维护成本.

8 结 语

软件的稳定性,或者说变更传播特性的研究,对软件的可维护性度量和评估具有很强的实用价值.但是,在软件的整个生命周期中,可能遇到的变更需求千变万化,变更需求的类型、粒度和具体内容也各不相同,这给软件的稳定性研究带来了极大的困难.

本文将软件的变更需求看作是一系列"原子变更需求"的叠加,将"原子变更需求"的响应抽象成:初始变更节点的随机选择过程以及由此引起的涟漪效应.提出通过软件在响应"原子变更需求"时平均需要变更的实体个数来度量软件的稳定性.

根据这个思想,本文提出了一种新的稳定性评价方法 SEMCIS,定义了软件的变更传播模型和稳定性度量指标.同时,为了克服概率论方法在处理循环依赖上的缺陷,提出用仿真方法计算稳定性指标,并且给出了相应的仿真算法.

通过对开源软件的实例分析,我们验证了评价指标在多次仿真中的收敛性,并就变更传播概率、软件结构对稳定性的影响进行了分析.实验结果表明:降低软件实体之间的变更传播概率,可以有效地改善软件稳定性;与此同时,在变更传播概率不变的前提下,改善软件的体系结构设计也可以有效地抵抗"涟漪效应"的发生,从而具有较好的稳定性.

进一步的分析表明,入度 Hub 的相互连接是造成大范围变更的主要原因,而入度 Hub 中存在的大量后继节点($v \leftarrow v_{\text{Hub}}$),则是造成大范围变更频繁发生的主要原因.减少公共实体之间的依赖关系,降低它们之间的耦合强度,有助于改善软件的稳定性,降低维护成本.

参 考 文 献

[1] Yau S S, Collofello J S, Macgregor T. Ripple effect analysis of software maintenance//Proceedings of the IEEE Computer Society's Second International Computer Software and Applications Conference (COMPSAC'78), 1978: 60-65

[2] Arnold R, Bohner S. Software Change Impact Analysis (Practitioners). Los Alamitos, CA, USA: Wiley-IEEE Computer Society Press, 1996

[3] Yau S S, Collofello J S. Some stability measures for software maintenance. IEEE Transactions on Software Engineering, 1980, SE-6(6): 545-552

[4] Tsantalis N, Chatzigeorgiou A, Stephanides G et al. Probabilistic evaluation of object-oriented systems//Proceedings of the 10th International Symposium on Software Metrics (METRICS 2004). Chicago, IL, USA, 2004: 26-33

[5] Tsantalis N, Chatzigeorgiou A, Stephanides G. Predicting the probability of change in object-oriented systems. IEEE Transactions on Software Engineering, 2005, 31(7): 601-614

[6] Sharafat A R, Tahvildari L. A probabilistic approach to predict changes in object-oriented software systems//Proceedings of the 11th European Conference on Software Maintenance and Reengineering (CSMR 2007). Amsterdam, Netherlands, 2007: 27-38

[7] Mirarab S, Hassouna A, Tahvildari L. Using Bayesian belief networks to predict change propagation in software systems//Proceedings of the 15th IEEE International Conference on Program Comprehension (ICPC'07). Banff, AB, Canada, 2007: 177-186

[8] Valverde S, Sole R V. Network motifs in computational graphs: A case study in software architecture. Physical Review E, 2005, 72(2): 26107-26108

[9] Zhang L, Qian G Q, Zhang L. Network motif and triad significance profile analyses on software system. WSEAS Transactions on Computers, 2008, 7(6): 756-765

[10] Hassan A E, Holt R C. Predicting change propagation in software systems//Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM 2004). Chicago, IL, USA, 2004: 284-293

[11] Zimmermann T, Diehl S, Zeller A. How history justifies system architecture (or not)//Proceedings of the 6th International Workshop on Principles of Software Evolution. Helsinki, Finland, 2003: 73-83

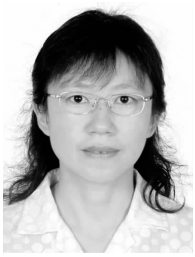
[12] Liu J, Lu J, He K et al. Characterizing the structural quality of general complex software networks. International Journal of Bifurcation and Chaos, 2008, 18(2): 605-613

[13] Henry S, Kafura D. The evaluation of software systems' structure using quantitative software metrics. Software-Practice and Experience, 1984, 14(6): 561-573

[14] Kannan R, Tetali P, Vempala S. Simple Markov-chain algorithms for generating bipartite graphs and tournaments//Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms. New Orleans, LA, USA, 1997: 193-210

[15] Maslov S, Sneppen K. Specificity and stability in topology of protein networks. Science, 2002, 296: 910-913

[16] Qian Guang-Qun, Zhang Li, Zhang Lin et al. Modeling method and characteristics analysis of software dependency networks. Computer Science, 2008, 35(11): 239-243 (in Chinese)
(钱冠群，张莉，张林等. 软件静态结构的依赖网络建模方法与特性分析. 计算机科学，2008，35(11)：239-243)



ZHANG Li, born in 1968, Ph. D., professor, Ph. D. supervisor. Her main research interests include software engineering and software architecture.

QIAN Guan-Qun, born in 1978, Ph. D. candidate. His main research interests include software architecture and reverse engineering.

LI Lin, born in 1984, Ph. D. candidate. Her main research interests include software architecture and software metrics.

Background

As one of the most important attributes of software quality, stability is used to characterize the sensitivity of changing a given system, and the negative impact that may be caused by system changes. Previous research has indicated that 70% of software development budgets is spent on software maintenance. And about 40% of the expense of software maintenance is spent on software changes. Knowing little about the changes or change propagation is the most significant reason which increases the cost and risk of the changes. Thus, effective stability measures can support better decision on software maintenance. But it's difficult to evaluate the software stability because of the complexity of software and the uncertainty of future change requirements.

In this paper, the authors regard various change requirements as the combination of a series of "atomic change requirement". The modification of software, which is used to satisfy the "atomic change requirement", is regarded as: firstly, modify a random selected "initial element"; secondly, a ripple effect will be caused by the change of this "initial element". The expected number of changed elements for various atomic change requirements is used to evaluate the stability of software. In order to simplify the calculation of the metrics, the authors introduce simulation technology into software stability evaluation instead of the conditional probability calculation.

This work has been supported by the National Natural Science Foundation of China under grant No. 60773155 and the Key Basic Research Developing Project (973 project) from Science and Technology Ministry of China under grant No. 2007CB310803.