

## 基于动态解耦的软件众包任务分解算法

王晨旭<sup>1</sup>, 王晓晨<sup>1</sup>, 余敦辉<sup>1,2</sup>, 吴 珊<sup>1</sup>

(1. 湖北大学 计算机与信息工程学院, 武汉 430062; 2. 湖北省教育信息化工程技术研究中心, 武汉 430062)

**摘 要:** 综合考虑任务粒度与解耦水平, 提出一种改进的软件众包任务分解算法。基于任务网络内的依赖关系计算任务粒度, 根据各子任务在设计结构矩阵中的分布情况衡量解耦水平, 并通过动态解耦进行软件众包任务分解。实验结果表明, 与基于独立水平和传播成本的任务分解算法相比, 该算法风险判定值和缺陷密度分别提升 0.244 0、0.362 6、0.014 6、0.319 4, 可保证软件众包任务完成质量。

**关键词:** 软件众包; 任务分解; 任务粒度; 动态解耦; 设计结构矩阵

开放科学(资源服务)标志码(OSID):



**中文引用格式:** 王晨旭, 王晓晨, 余敦辉, 等. 基于动态解耦的软件众包任务分解算法[J]. 计算机工程, 2019, 45(8): 120-124, 134.

**英文引用格式:** WANG Chenxu, WANG Xiaochen, YU Dunhui, et al. Software crowdsourcing task decomposition algorithm based on dynamic decoupling[J]. Computer Engineering, 2019, 45(8): 120-124, 134.

## Software Crowdsourcing Task Decomposition Algorithm Based on Dynamic Decoupling

WANG Chenxu<sup>1</sup>, WANG Xiaochen<sup>1</sup>, YU Dunhui<sup>1,2</sup>, WU Shan<sup>1</sup>

(1. College of Computer Science and Information Engineering, Hubei University, Wuhan 430062, China;

2. Hubei Province Engineering Technology Research Center for Education Informationization, Wuhan 430062, China)

**[Abstract]** Considering the task granularity and Decoupling Level (DL), an improved software crowdsourcing task decomposition algorithm is proposed. The task granularity is calculated based on the dependency relationship in the task network. The decoupling level is measured according to the distribution of each subtask in the Design Structure Matrix (DSM), and the software crowdsourcing task decomposition is performed by dynamic decoupling. Experimental results show that compared with the task decomposition algorithm based on Independent Level (IL) and Propagation Cost (PC), the risk judgment value and defect density of the algorithm are increased by 0.244 0, 0.362 6, 0.014 6 and 0.319 4 respectively, which can ensure the completion quality of software crowdsourcing tasks.

**[Key words]** software crowdsourcing; task decomposition; task granularity; dynamic decoupling; Design Structure Matrix (DSM)

**DOI:** 10.19678/j.issn.1000-3428.0053485

### 0 概述

随着互联网技术的发展, 软件众包作为一种新型的开发模式, 已经引起各领域的广泛关注。软件众包模式<sup>[1]</sup>的发展使得软件开发不再局限于孤立的开发者团体, 而是由一个组织或社区中的开发者合作完成。高效的开发和低价的成本使得越来越多的用户选择众包模式发布软件任务<sup>[2]</sup>。

复杂的软件众包任务为了吸引更多的工人参与开发, 需要进行合理的任务分解, 而分解的好坏直接影响众包模式的开发效率。如果分解不当, 则会增加通信成本, 增大任务整合难度, 提高工人参加众包任务的门槛。因此, 如何将复杂的软件众包任务分解为若干微任务是亟待解决的问题<sup>[3-5]</sup>。

文献[6]提出基于设计结构矩阵(Design Structure Matrix, DSM)的解耦理论。文献[7]修正了 DSM 并

**基金项目:** 国家自然科学基金(61572371, 61832014); 湖北省技术创新专项(2018ACA13)。

**作者简介:** 王晨旭(1997—), 男, 本科生, 主研方向为软件众包模式、服务计算; 王晓晨, 本科生; 余敦辉(通信作者), 副教授、博士; 吴 珊, 本科生。

**收稿日期:** 2018-12-25      **修回日期:** 2019-02-14      **E-mail:** yumhy@hubu.edu.cn

使用增广约束网络 (Augmented Constrained Network, ACN) 衡量模块间的依赖关系。文献[8]基于 ACN 将 DSM 分解为多层结构。文献[9]提出一种 ArchDRH 设计规则。以上研究均证明了基于 DSM 进行任务解耦的有效性, 但上述方法较复杂。在此基础上, 文献[10]简化了 DSM 的设计方法, 提出一种基于解耦水平 (Decoupling Level, DL) 的度量方法, 但该方法对任务粒度考虑不足。文献[11]综合考虑了任务粒度、任务耦合性以及任务均衡度, 但对任务耦合性的度量说明不明确。文献[12]针对单输入多输出的耦合模型进行研究, 但增加了额外的通信开销。文献[13]提出多阶段混合模型, 但对如何构建解耦集说明不具体。本文在研究众包模式与传统外包模式差异的基础上, 综合考虑任务粒度与解耦性, 提出一种基于动态解耦的软件众包任务分解算法 (CTDL)。

## 1 问题模型

本文研究的软件众包任务基于竞标模式, 相关定义及问题描述具体如下:

**定义 1 (软件任务)** 软件任务是指经过众包平台架构师分解的任务集合。软件任务是一个三元组:  $M = \{DL_m, GR_m, DS_m\}$ , 其中,  $DL_m$  为任务解耦水平,  $GR_m$  为任务粒度,  $DS_m$  表示对任务的描述。

**定义 2 (任务网络)** 本文研究的任务网络是指由类、接口和包构成的依赖关系网络, 任务网络定义为一个有向网络  $G_d = (V_d, E_d)$ , 其中结点  $v_d$  ( $v_d \in V_d$ ) 代表软件的类或者接口, 如果 2 个结点之间存在关系, 则构成有向边  $e_{ij}$  ( $e_{ij} \in E_d$ )。为量化依赖关系, 本文主要考虑以下 4 种情况<sup>[14]</sup>:

1) 实现, 假设  $v_1$  类实现了  $v_2$  接口, 则存在有向边  $e_{12} = \langle v_1, v_2 \rangle$ , 边的权值用  $\alpha$  表示。

2) 泛化, 假设  $v_3$  类继承了  $v_4$  类的功能, 则存在有向边  $e_{34} = \langle v_3, v_4 \rangle$ , 边的权值用  $\beta$  表示。

3) 依赖, 假设  $v_5$  类的变化会引起  $v_6$  类的变化, 则存在有向边  $e_{56} = \langle v_5, v_6 \rangle$ , 边的权值用  $\gamma$  表示。

4) 关联, 假设  $v_7$  类和  $v_8$  类之间有一定的联系, 该关系使得一个类知道另一个类的属性和方法, 则存在有向边  $e_{78} = \langle v_7, v_8 \rangle$ , 边的权值用  $\varphi$  表示。

考虑到上述 4 种关系在实际开发中依赖程度不同, 因此本文约定  $\alpha + \beta + \gamma + \varphi = 1$ 。本文研究的包是具有相关性的一组类或接口的集合, 定义  $G_p = (V_p, E_p)$ , 其中结点  $v_p$  ( $v_p \in V_p$ ) 代表包。如果 2 个包之间存在依赖关系, 则构成有向边  $e_{ab}$  ( $e_{ab} \in E_p$ ), 该边从一个结点内部指向另一个结点内部, 边的权值用  $\delta$  表示, 示例如图 1 所示, 其中,  $Package_1$ 、 $Package_2$  表示包名, 简称为  $P_1$ 、 $P_2$ ,  $class$  表示类名,  $impl$  表示接口, 权值均为关联关系。

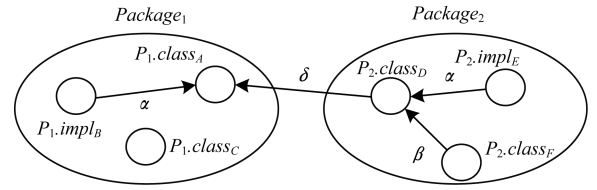


图1 任务网络示例

**定义 3 (设计结构矩阵)** 为能更好地反映各任务之间的依赖关系, 本文定义 DSM, DSM 是一个方阵。该矩阵的行和列使用相同的顺序标记任务名称, 该矩阵分为上下两层, 上层任务对其他任务有依赖关系, 下层任务对其他任务没有依赖关系。单元格  $(x, y)$  表示  $x$  依赖  $y$ 。DSM 示例如图 2 所示, 横纵坐标的含义一一对应,  $\checkmark$  表示 2 个子任务之间有关系, 由于子任务本身不构成任何关系, 因此本文用灰色部分加以区分。

	1	2	3	4	5	6
1. $P_1.class_A$	1	$\checkmark$			上层	
2. $P_1.impl_D$		2				
3. $P_2.class_B$	$\checkmark$		3		下层	
4. $P_1.class_C$				4		
5. $P_2.impl_E$		$\checkmark$			5	
6. $P_2.class_F$		$\checkmark$				6

图2 DSM 示例

文献[8]证明通过 UML 用例图可以快速得到 DSM。本文假设众包平台的软件架构师已根据用例图构建 DSM, 后续研究重点是对 DSM 矩阵的调整。

**定义 4 (包关联矩阵)** 为能更好地反映包之间的联系关系, 便于对包的聚类, 本文定义一个包关联矩阵, 包关联矩阵是一个方阵。该矩阵的单元格表示包之间的关联关系, 具体如图 3 所示。

	$P_1$	$P_2$	$P_3$	$P_4$	$P_5$
$P_1$	0.0	0.7	0.7	0.3	0.5
$P_2$	0.7	0.0	0.4	0.1	0.4
$P_3$	0.7	0.4	0.0	0.6	0.6
$P_4$	0.3	0.1	0.6	0.0	0.5
$P_5$	0.5	0.4	0.6	0.5	0.0

图3 包关联矩阵

## 2 基于任务网络的任务粒度计算

由于众包模式下发包方希望较多的工人参与项目开发, 以便从中选出完成情况最优的任务, 因此降低项目开发的门槛, 让更多工人参与进来显得尤为重要。任务粒度表征任务内部聚合情况, 也可以反映任务组织规模和数量, 每个软件任务中的类和包是影响任务粒度的主要因素, 类和包的数量越多任务粒度越小<sup>[11]</sup>, 本文基于任务网络进行任务

粒度计算。假设某个软件众包项目由  $g$  个子任务构成,  $K_g$  表示第  $g$  个子任务的粒度系数, 该任务的粒度计算如下:

$$GR = \sum_{i=1}^g K_g \times (c_g)^{-1} \quad (1)$$

其中,  $c_g$  表示第  $g$  个子任务内的类和接口数量。

对于一个任务而言, 类、接口、包之间的依赖关系越复杂, 任务内部的关联性就越大<sup>[15]</sup>。本文使用内聚函数  $S(t)$  描述任务  $t$  内部的关联程度, 具体如下:

$$S(t) = \begin{cases} \frac{1}{ph} \sum_{i=1}^p \sum_{j=1}^h (e_{ij\alpha} + e_{ij\beta} + e_{ij\gamma} + e_{ij\varphi}), & e_{ij} \in E_d \\ \frac{1}{q} \sum_{l=1}^q e_{l\delta}, & e_l \in E_p \end{cases} \quad (2)$$

其中,  $p$  表示一个任务内部包的个数,  $h$  表示第  $i$  个包内的边数目,  $q$  表示连接包之间的边个数,  $e_{ij\alpha}$ 、 $e_{ij\beta}$ 、 $e_{ij\gamma}$ 、 $e_{ij\varphi}$ 、 $e_{l\delta}$  表示边的权值。  $e_{ij\alpha} + e_{ij\beta} + e_{ij\gamma} + e_{ij\varphi}$  是出于对 2 个类(接口)之间可能存在多种依赖关系的考虑, 在实际计算中, 若只有一种关系, 则其他项为 0 即可。

任务粒度主要取决于任务的内聚性<sup>[10]</sup>, 因此对于任务中的第  $r$  个任务, 其粒度计算如下:

$$K_r = \exp(-S(r)^{-1}) \quad (3)$$

将式(3)代入式(1)得到任务粒度计算公式如下:

$$GR = \sum_{i=1}^g \exp(-S(g)^{-1}) \times (c_g)^{-1} \quad (4)$$

### 3 基于解耦水平的任务耦合度计算

#### 3.1 计算方法描述

假设软件众包项目由  $n$  个任务组合而成, 整个项目的解耦水平值即为上下层任务的  $DL$  值之和:

$$DL = \sum_{L_i=1}^n DL_{L_i} \quad (5)$$

其中,  $DL$  值越大表示解耦程度越高, 即任务越独立。

由于上下两层依赖情况截然不同, 因此需要采用不同的计算方式, 本文在文献[10]的式(2)~式(5)的基础上做了部分简化。由于下层任务是独立任务, 对于整个项目而言, 下层任务数量越多,  $DL$  值越大, 具体计算如下:

$$DL_{L_i} = \frac{LLM}{AM} \quad (6)$$

其中,  $LLM$  表示下层任务总数,  $AM$  表示任务总数量。

由于每个上层任务依赖下层的任务, 因此依赖越多,  $DL$  值越小, 具体计算如下:

$$DL_{L_i} = \sum_{j=1}^k \left[ \frac{1}{AM} \times \left( 1 - \frac{Dep_j}{LLM} \right) \right] \quad (7)$$

其中,  $Dep_j$  表示依赖于第  $j$  个任务的任务总数。

### 3.2 算例介绍

本文以湖北省教育信息化发展水平监测系统为例进行说明, 如图 4 所示。

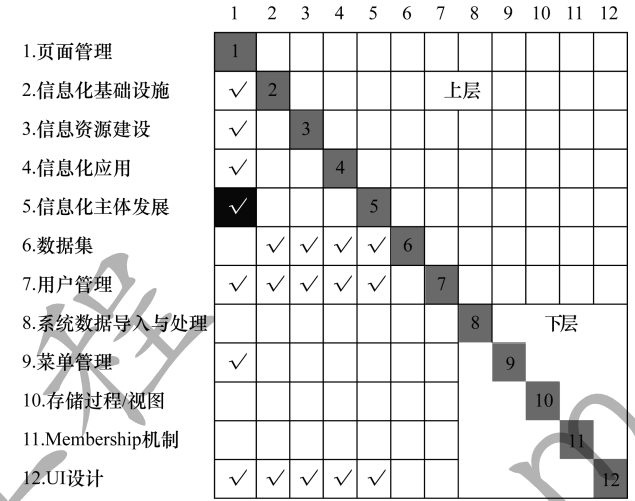


图 4 湖北省教育信息化发展水平监测系统分解

在图 4 中, 横纵坐标的含义一一对应,  $\checkmark$  表示横轴对应的模块依赖纵轴对应的模块, 黑色方框表示页面管理依赖信息化主体发展,  $DL$  具体计算如下:

$$DL_{上层} = \frac{1}{12} \times \left[ \left( 1 - \frac{7}{5} \right) + \left( 1 - \frac{3}{5} \right) \times 4 + \left( 1 - \frac{2}{5} \right) + \left( 1 - \frac{1}{5} \right) \right] = \frac{13}{60}$$

$$DL_{下层} = \frac{5}{12}$$

$$DL = (DL_{下层} + DL_{上层}) \times 100\% = 63.33\%$$

### 4 基于动态解耦的众包任务分解

考虑到软件众包任务分解过程中, 如果任务解耦不足会出现修改一个模块需要同时修改相关联的模块的情况, 则会增加工人之间的通信, 导致任务延期, 甚至增加软件风险; 如果解耦过度, 则会增加任务整合难度; 如果分解后任务粒度太大, 则会提高工人参与任务的门槛。

#### 4.1 算法分解过程

本文分解算法首先给定解耦阈值并根据任务粒度动态调整解耦阈值进行分解, 直到满足任务粒度阈值和解耦阈值, 在分解过程中基于包关联矩阵对包进行聚类形成新任务。

考虑到不同类别的软件分解情况不同, 较难基于经验衡量任务粒度与解耦水平, 因此本文用集合  $GR_r$  表示第  $r$  类软件的粒度。鉴于众包模式的独特性, 在实际开发中众包平台可根据工人数量对任务粒度进行调整, 因此本文引入调节参数  $\eta_0$ , 每类软件的任务粒度均值用  $\lambda_r$  表示, 具体如下:

$$\lambda_r = \left( \frac{1}{b} \sum_{i=1}^b W_b \times GR_b \right) \times \eta_0 \quad (8)$$

其中,  $W_b$  表示权值, 实际开发中由众包平台指定, 调节参数  $\eta_0$  取值为  $(0, 2)$ 。

本文用集合  $DL_r$  表示第  $r$  类软件的解耦水平, 假设第  $r$  类软件包含  $a$  个已成功开发的软件, 则每类软件的解耦水平均值  $\varepsilon_r$  表示如下:

$$\varepsilon_r = \frac{1}{a} \sum_{i=1}^a W_a \times DL_a \quad (9)$$

其中,  $W_a$  表示权值, 实际开发中由众包平台指定。

在对任务进行分解时, 为考虑分解的均衡性, 选取任务内部关联系数排名前  $K$  的包进行聚类, 如图3所示选择关联系数最大的包进行聚类后,  $P_1$ 、 $P_2$ 、 $P_3$  和  $P_4$ 、 $P_5$  可构成2个新任务。在实际开发中  $K$  值可由众包平台指定。

#### 4.2 CTDL 算法描述

CTDL 算法具体描述如下:

输入 待分解的软件项目  $P$  的用例图、解耦阈值、任务粒度阈值  $\lambda_r$

输出 任务分解集合  $T$

1. 将用例图转化为 DSM 矩阵;
2. 构建软件项目  $P$  的任务网络;
3. 计算 DSM 内每个任务的解耦值  $DL$ ;
4. 计算 DSM 的解耦值  $DSM\_DL$ ;
5. 基于任务网络计算任务粒度  $G$ ;
6. 判断  $DSM\_DL$  是否大于解耦阈值  $\varepsilon_r$ , 若大于则转步骤7, 若小于则转步骤9;
7. 判断任务粒度  $G$  是否小于任务粒度阈值  $\lambda_r$ , 若小于则转步骤8, 若大于则转步骤13;
8.  $\varepsilon_r = \varepsilon_r + L$  转步骤6; // 增大阈值,  $L$  表示步长
9. 按照 DSM 内各任务的  $DL$  值进行升序排序, 选择解耦值最大的任务  $Maxtask$ ;
10. 将  $Maxtask$  内关联系数排名前  $K$  的包进行聚合构成新任务  $t_1$ , 剩余任务构成  $t_2$ ;
11. 用  $t_1$  替换任务  $i$ , 将  $t_2$  加入 DSM 矩阵, 计算解耦值  $DSM\_DL$ ;
12. 计算任务粒度  $G$  转步骤6;
13. 输出任务分解集合  $T$

### 5 实验结果与分析

因为软件众包领域缺乏基准实验数据, 本文从 GitHub 社区选取了 108 个已成功开发的软件作为实验数据集, 经分解后累计微任务数达到 1 237 个。本文首先使用 PowerDesigner 对这些软件进行逆向工程, 然后使用 SourceCounter 软件统计其规模及缺陷密度, 最后计算解耦值和任务粒度。实验对比本文解耦方法与其他解耦方法的效果, 并给出任务粒度阈值和解耦阈值的取值范围。

#### 5.1 动态分解算法的性能对比

由于本文提出的基于动态分解的软件众包任务分解算法核心为解耦, 为证明分解算法的有效性, 实验将本文提出分解算法与基于传播成本 (Propagation Cost,

PC) 的分解算法<sup>[16]</sup>、基于独立水平 (Independent Level, IL) 的分解算法<sup>[17]</sup> 进行对比, 主要对比以下2项指标:

1) 风险判定值, 该值参考文献 [18-19] 提出的软件风险评估方法计算得出。

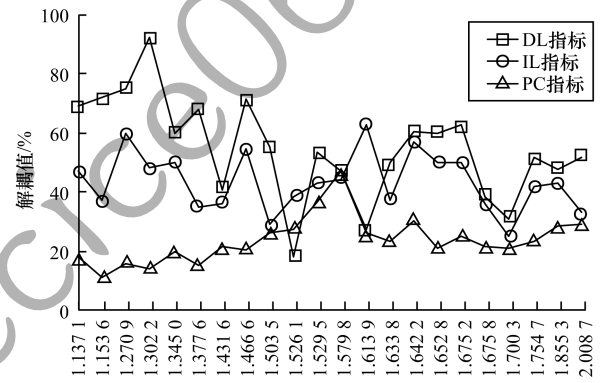
2) 缺陷密度, 该值反映每千行代码可能产生的软件缺陷数量, 对于衡量软件的可维护性、软件漏洞具有一定的参考价值。

基于 PC 与 IL 的分解算法均基于 DSM 矩阵计算, PC 值为非空单元格除以总单元格, IL 值为下层的非空单元格数量除以 DSM 内的任务总数。由于同一类软件的解耦维度具有较高的相似度, 因此本文按照软件类型对其进行聚类, 并以同一类软件解耦值的平均值作为该类软件解耦值的等价值。

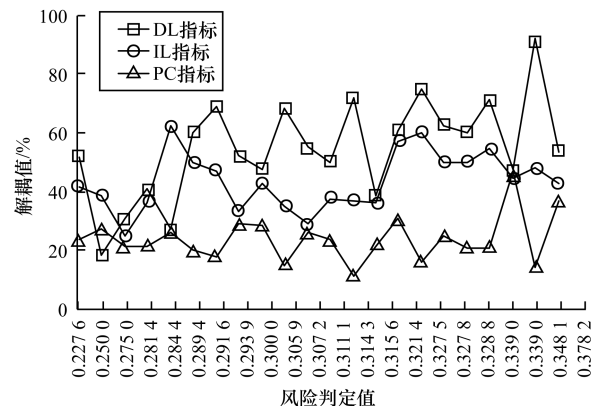
为比较图5中3个指标与风险判定值、缺陷密度的相关性, 本文使用皮尔逊值<sup>[20]</sup> 进行比较。皮尔逊值计算方法如下:

$$r = \frac{\sum ab - \frac{\sum a \sum b}{N}}{\sqrt{\left(\sum a^2 - \frac{(\sum a)^2}{N}\right) \left(\sum b^2 - \frac{(\sum b)^2}{N}\right)}} \quad (10)$$

其中,  $r$  表示相关系数,  $N$  表示变量取值个数,  $a$ 、 $b$  表示变量。



(a) 解耦值与缺陷密度的关系



(b) 解耦值与风险判断值的关系

图5 解耦值与缺陷密度和风险判断值的关系

由表1可以看出, DL 变化更能反映风险判定值和缺陷密度的变化。基于 DL 的分解算法相对基于

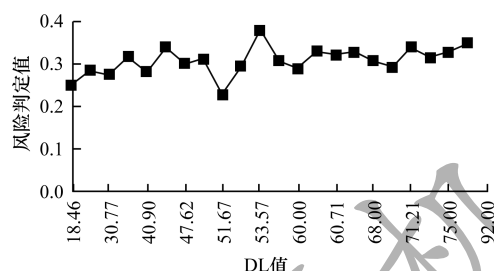
IL 的分解算法风险判定值提升 0.244 0, 缺陷密度提升 0.014 6; 相对于基于 PC 的分解算法风险判定值提升 0.362 6, 缺陷密度提升 0.319 4。

表 1 解耦指标与风险判定值和缺陷密度的关系

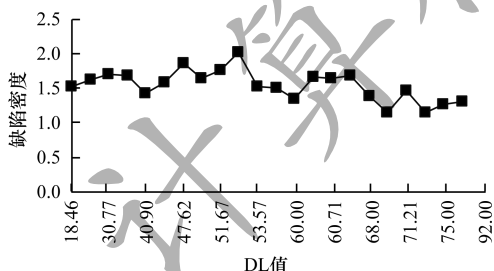
性能指标	基于 DL 的 分解算法	基于 IL 的 分解算法	基于 PC 的 分解算法
风险判定值	0.529 6	0.285 6	0.167 0
缺陷密度	0.525 8	0.511 2	0.206 4

## 5.2 阈值确定

图 6、图 7 分别对比了 DL 值、任务粒度与风险判定值和缺陷密度的关系, 根据实验结果可知, 任务粒度取 0.003 0 ~ 0.003 9, DL 值取 53.51 ~ 71.21, 此时软件风险和缺陷密度较小。

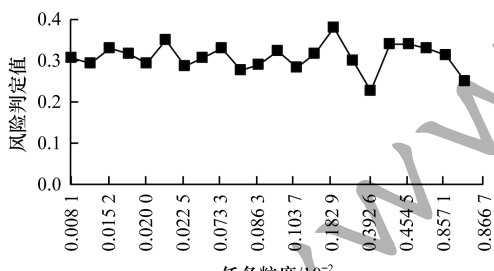


(a) DL 值与风险判定值的关系

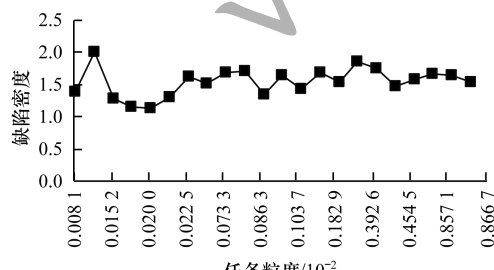


(b) DL 值与缺陷密度的关系

图 6 DL 值与风险判定值和缺陷密度的关系



(a) 任务粒度与风险判定值的关系



(b) 任务粒度与缺陷密度的关系

图 7 任务粒度与风险判定值和缺陷密度的关系

## 6 结束语

本文给出任务粒度和解耦水平的定义, 提出一种基于动态解耦的软件众包任务分解算法。实验结果表明, 与基于 IL 和 PC 的分解算法相比, 本文算法能更好地降低软件风险、缺陷密度以及工人参与任务的门槛, 提高软件众包模式的开发效率。下一步将优化解耦水平与任务粒度的度量方式, 在此基础上综合考虑子任务定价等因素, 以提升任务完成质量, 降低发包方成本。

## 参考文献

- [1] TSAI W T, WU Wenjun, HUHNS M N. Cloud-based software crowdsourcing [J]. IEEE Internet Computing, 2014, 18(3): 78-83.
- [2] 冯剑红, 李国良, 冯建华. 众包技术研究综述 [J]. 计算机学报, 2015, 38(9): 1713-1726.
- [3] NAIK N. Software CROWD-sourcing [C]//Proceedings of the 11th International Conference on Research Challenges in Information Science. Washington D. C., USA: IEEE Press, 2017: 463-464.
- [4] DWARAKANATH A, CHINTALA U, SHRIKANTH N C, et al. Crowd build: a methodology for enterprise software development using crowdsourcing [C]//Proceedings of the 2nd International Workshop on Crowdsourcing in Software Engineering. Washington D. C., USA: IEEE Press, 2015: 8-14.
- [5] KULKARNI A, CAN M. Collaboratively crowdsourcing workflows with turkomatic [C]//Proceedings of 2012 Conference on Computer Supported Cooperative Work. New York, USA: ACM Press, 2012: 1003-1012.
- [6] BALDWIN C Y, CLARK K B. Design rules volume I: the power of modularity [M]. Cambridge, USA: MIT Press, 2000.
- [7] CAI Yuanfang, SULLIVAN K J. Modularity analysis of logical design models [C]//Proceedings of IEEE/ACM International Conference on Automated Software Engineering. Washington D. C., USA: IEEE Press, 2006: 1-8.
- [8] WONG S, CAI Yuanfang, VALETTO G, et al. Design rule hierarchies and parallelism in software development tasks [C]//Proceedings of IEEE/ACM International Conference on Automated Software Engineering. Washington D. C., USA: IEEE Press, 2009: 1-8.
- [9] CAI Yuanfang, WANG H, WONG S, et al. Leveraging design rules to improve software architecture recovery [C]//Proceedings of International ACM SIGSOFT Conference on Quality of Software Architectures. New York, USA: ACM Press, 2013: 113-142.
- [10] MO Ran, CAI Yuanfang, KAZMAN R, et al. Decoupling level: a new metric for architectural maintenance complexity [C]//Proceedings of IEEE/ACM International Conference on Software Engineering. Washington D. C., USA: IEEE Press, 2016: 499-510.
- [11] 包北方, 杨育, 李斐, 等. 产品定制协同开发任务分解模型 [J]. 计算机集成制造系统, 2014, 20(7): 1537-1545.

(下转第 134 页)

(上接第 124 页)

- [12] 田启华,汪涛,杜义贤,等. 单输入多输出耦合设计任务二阶段迭代模型研究[J]. 工程设计学报,2017,24(2):134-140.
- [13] 田启华,文小勇,梅月媛,等. 基于遗传算法的并行设计耦合任务分布规划[J]. 机械设计与研究,2017,33(6):98-103.
- [14] 何鹏,王鹏,李兵,等. 基于多粒度软件网络模型的软件系统演化分析[J]. 电子学报,2018,46(2):257-267.
- [15] 庞辉,方宗德. 网络化协作任务分解策略与粒度设计[J]. 计算机集成制造系统,2008,14(3):425-430.
- [16] MACCORMACK A, RUSNAK J, BALDWIN C Y. Exploring the structure of complex software designs: an empirical study of open source and proprietary code[J]. Management Science,2006,52(7):1015-1030.
- [17] SETHI K,CAI Yuanfang,WONG S,et al. From retrospect to prospect: assessing modularity and stability from software architecture[C]//Proceedings of 2009 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture. Washington D.C.,USA: IEEE Press,2009:1-11.
- [18] 张俊光,吕廷杰,马晓平. 软件项目风险评估方法应用探讨[J]. 计算机应用研究,2006,23(10):76-77.
- [19] 荆文娟. 基于 UML 软件体系结构的软件风险评估[D]. 南京:南京理工大学,2011.
- [20] 申利民,杨益良,陈真. 考虑相似比率的 Web 服务 QoS 协同预测[J]. 计算机集成制造系统,2016,22(1):144-154.

编辑 陆燕菲