面向 Code Smells 的“容器—破坏者—发现者”检测策略

林 涛[1],高建华[1],伏 雪[1],林 艳[2]

[1](上海师范大学 计算机科学与工程系,上海 200234)
[2](奥克兰大学 信息系统系,奥克兰 新西兰 92019)
E-mail:l. t@ acm. org

摘 要: 软件重构在软件工程中愈显重要,对需要重构代码 code smells 的检测是基础工作,但 code smells 定义模糊、无量化. 该文将人工免疫的基本概念与信号迁移至软件工程,提出一种基于危险理论中的树突状细胞算法的检测策略. 该策略算法中,包含 code smells 的代码作为抗原,软件度量值转化为危险信号和安全信号等输入信号进行处理,通过权值公式获得成熟信号以及半成熟信号,比较其相对值高低确定代码是否为 code smells,最终根据成熟环境抗原值决定各种 code smells 严重程度的优先次序. 该策略具有较低假阳性率. 实验证明该研究在 F-score(0.784)和 Kappa 分析(0.756)上均有效,高于其他检测方法.

关 键 词: 软件重构;树突状细胞算法;软件缺陷;软件质量;人工免疫理论;危险理论
中图分类号: TP311 **文献标识码:** A **文 章 编 号:** 1000-1220(2015)--

A Container-destructor-explorer Paradigmto Code Smells Detection

LIN Tao[1], GAO Jian-hua[1], FU Xue[1], LIN Yan[2]

[1] (Department of Computer Science and Technology, Shanghai Normal University, Shanghai 200234, China)
[2] (Department of InformationSystemand Operations Management, The University of Auckland, Auckland 92019, New Zealand)

Abstract: Software refactoring is increasingly significant in software engineering, besides it is a fundamental task for the detection of code smells being indefiniteand non-quantitative for the purpose of refactoring. After essential concepts and signals are migrated to software engineering, the paper presents a detection paradigm, whichis based on dendritic cell algorithm in danger theory andregards code smells as the antigen. Software metrics valuesconvert to the danger signal and the safe signal for processing, in which mature signal and semi-mature signal is calculated by weight equation. Code smells can be confirmed in comparison of relative signal values. A variety of code smells′ prioritiesare determined by mature context antigen value. There are lower false positive rates in the paradigm. The experiment proves that this approach is competitive effectiveness in F-score(0.784) as well asKappa analysis(0.756) and outperformance compared to other detection techniques.

Key words: software refactoring; dendritic cell algorithm; software bug; software quality; artificial immune systems; danger theory

1 引 言

软件工程,尤其是是超大规模系统(Ultra-large-scale system,ULSS)面临难以维护、不易升级、频繁崩溃等诸多难题,软件重构的理念应运而生[1]. 一方面,伴随着重构在软件工程中的地位逐步上升,code smells 作为实现重构的一个重要基础也同时受到研究者的更多关注. 另一方面,对 code smells 的检测尚不能自动化.

Fowler 等人提出 code smells 概念[2],并且归纳出 22 种类型. code smells 是程序源代码中可能引起更深层问题的症状. codesmells 的存在使软件项目难以维护,工程不利复用,代码不易理解. 具备 code smells 特征的代码一般需要重构. 其产生原因并非来自需求分析不足或软件首次发行版本的编程失误,而是软件在代码多次修正、升级后逐渐积累的. 针对 code smells 的检测虽然有基于并行搜索[3]和双层最优问题处理[4]等方法,但是均存在比较高的错检率,即假阳性率.

本研究主要贡献为:应用人工免疫理论,提出一种“容器—破坏者—发现者”(Container-Destructor—Explorer,CDE)策略检测软件代码的 code smells.

2 背 景

本节主要介绍 code smells 和“容器—破坏者—发现者”检测策略需要应用的树突状细胞算法的生物学机理.

2.1 Code smells

code smells 是一种“反设计模式”,不会直接产生软件缺陷,但间接影响软件质量[5],诸如维护、复用、拓展等问题[6]. code smells 一般是在软件版本升级时引入的. code smells 研究面临如下两个问题:

1) code smells 的类型难以划分、是值得商榷的. 具体讨论如下:

收稿日期:2014-12-31 收修改稿日期:2015-02-16 基金项目:国家自然科学基金项目(61073163)资助;上海市企业自主创新专项资金项目(沪 CXY-2013-88)资助. 作者简介:**林 涛**,男,1988 年生,硕士研究生,CCF 学生会员,研究方向为软件重构、人工免疫算法;**高建华**,男,1963 年生,博士,教授,博士生导师,CCF 高级会员,研究方向为软件可靠性理论与设计、软件开发环境与开发技术、数据安全与计算机安全、网络测试、LSI/VLSI 测试等领域;**伏 雪**,女,研究方向为可信计算;**林 艳**,女,研究方向为软件复用.

(a) Long parameter list 和 spaghetti code[1](代码包含复杂控制结构)两种类型,一般都是 long method 类型的主要特征.

(b) Blob 类型和 data class 两种类型是同一代码重构问题的体现. Blob 类型是一个类作为行为,其他类封装数据. 因而,这些其他类自然是 data class.

(c) Divergent change 和 shotgun surgery 存在权衡关系. divergent change 是因为各种原因对个别类进行经常性变化; shotgun surgery 则与之对应,是对很多类进行反复变化. 由此,在软件工程中,divergent change 和 shotgun surgery 矛盾对立需要权衡利弊. 另一方面,parallel inheritance hierarchies 类型是 shotgun surgery 的一个子集.

2) 难以量化. 至今,code smells 都缺少精确的数学定义[7],对 code smells 的检测主要依赖程序员从业经验的主观判断.

因而,基于以上分析,无需对所有 code smells 类型进行研究. 本研究选取四种 code smells 类型,即 Long Method、Lazy Class、Speculative Generality 和 Refused Bequest,对其进行量化研究.

2.2 树突状细胞算法生物学机理

树突状细胞算法是基于危险理论的人工免疫算法[8]. 其生物学基础是人类免疫系统中树突状细胞(Dendritic Cell, DC)从淋巴结迁移到机体组织,抓取抗原,并收集抗原所处环境中的多种信号,呈递给 T 细胞以识别抗原.

传统免疫学认为免疫应答是机体判别"自我"与"非我"的物质,危险理论则提出人体免疫是对各种信号做出的反映[9].

3 "容器—破坏者—发现者"检测策略

本研究针对 code smells 检测,提出"容器—破坏者—发现者"检测策略,即 CDE 策略. CDE 策略中的"容器",主要针对重构前代码和重构后代码要分属不同"容器",不可混淆. "破坏者"指抗原 code smells. "发现者"指树突状细胞 DC. 本部分介绍 CDE 检测策略的基本概念、算法流程以及一个实例分析.

3.1 概念抽象

在应用人工免疫算法制定"容器—破坏者—发现者"检测策略前,需将人体免疫理论中的相关信号术语迁移至此研究,如表 1 所示. 病原相关分子模式信号(pathogenic associated molecular patterns, PAMP)是潜在生物入侵的标记. 嵌合分子信号(chimeric signaling molecule, CSM)意味树突状细胞开始准备抗原呈递.

表 1 免疫信号术语在软件工程中的解释

Table 1 Explanation of immune signal terminology in software engineering

人体免疫信号	信号方向	生物学解释	软件工程解释
PAMP 信号	输入	在病原体中大量存在	明显带有 code smells 特征
危险信号		可能伤害机体	模糊带有 code smells 特征
安全信号		对机体无害的正常信号	规范代码特征
炎性因子信号		可以膨胀前三种信号	通过此信号,本策略可以适用于各种规模的软件工程
CSM 信号	输出	CSM 的浓度决定未成熟 DC 是否转换	此信号,决定是否需要对待检代码进一步分析
半成熟信号		免疫系统对安全信号的响应	判定该代码不是 code smells
成熟信号		免疫系统对危险信号的响应	判决该代码是 code smells

3.2 算法

针对 code smells 的 CDE 检测策略基于树突状细胞算法[10],每个 DC 都通过迭代扫描代码输出结果,伪代码如下:

算法 CDE 检测策略

Input: S = collections of code; variety of signals; iDC;

Output: smDC; mDC; MCAV;

```
1  DCs <-Initiate(iDC);
2  For code in a class in S
3    while CSM < Threshold do
4      scan S;
5      get PAMP, danger signal, and safe signal;
6    end
7  end
8  calculate CSM, Semi-Mature signal, and Mature signal;
9  if
10   Semi-Mature > Mature then
11   set as smDC;
12 else
13   set as mDC;
14 end
15 for all DCs do
16   calculate MCAV;
17 end
```

CDE 策略总体流程如图 1 所示.

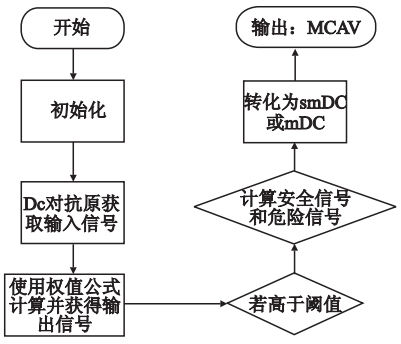

图 1 策略的流程图

Fig. 1 Paradigm's flowchart

CDE 策略分为以下几步:

1) CDE 策略首先初始化软件度量指标与细胞信号之间的相关参数(算法第 1 行).

[1] 对于 code smells 类型名称,中文尚未有统一规范的翻译,例如 spaghetti code 和 speculative generality 两种类型名称,一种中文流行翻译分别是"意大利面条式代码"以及"夸夸其谈未来性",存在争议,故全文 code smells 类型名称全部使用英文原文.

2）扫描源代码,获取 PAMP、危险信号(danger signal)和安全信号(safe signal)等输入信号(算法第 4、5 行).

3）随后根据权值矩阵使用权值公式计算 CSM、半成熟信号(Semi-Mature signal)和成熟信号(Mature signal)等作为输出信号(算法第 8 行).

DC 通过权值公式将各种输入信号转化为输出信号,权值公式如(1)所示,

$$O_{ip} = \sum_{j=0}^{2} W_{jp} S_{ij} * (1 + \mathrm{I}) \tag{1}$$

其中:i 表示代码的位置,j 表示三种输入信号,p 代表三种输出信号,S_{ij}和 O_{ip}分别表示程序代码中第 i 个检测位置的输入信号和输出信号的强度,I 为炎性因子信号,W_{jp}是权值公式中的相应权值,见表 2. 其中,输入信号 PAMP 以权值 W_1 和 W_2 转为 CSM 信号和成熟输出信号,危险信号和安全信号的转化是在其基础上增加系数 λ 和 μ,根据不同规模软件系统可以自定.

表 2　权值矩阵

Table 2　Weight matrix

信号	PAMP 信号	危险信号	安全信号
CSM 信号	W_1	W_1/λ	μW_1
半成熟信号	0	0	1
成熟信号	W_2	W_2/λ	$-\mu W_2$

4）CSM 信号若高于转化阈值,则未成熟 DC 依据半成熟信号和成熟信号的相对高低转化为半成熟 DC 或成熟 DC(算法第 9 行到第 14 行).

未成熟树突状细胞的转化阈值由公式(2)决定:

$$T = ((\max_p * W_{pc}) + (\max_d * W_{dc}) + (\max_s * W_{sc})) * \left(1 + \frac{I}{\eta}\right) \tag{2}$$

其中,T 为转化阈值;$\max_p$、$\max_d$ 和 $\max_s$ 分别是最大 PAMP 信号、最大危险信号以及最大安全信号;W_{pc}、W_{dc} 和 W_{sc}分别是 PAMP 信号、危险信号和安全信号相对 CSM 信号的权值;I 为炎性信号;η 为系数,根据具体情况设置.

5）迭代前四步,最终通过 Mature Context Antigen Value (MCAV)确定该软件工程中的 code smells 是否影响软件总体质量(算法第 16 行).

MCAV 评价整个软件工程中 code smells 的严重程度,由公式(3)决定,

$$\mathrm{MCAV} = \frac{\text{mDC 数量}}{\text{抗原总数}} \tag{3}$$

3.3　实例分析

本节以实例具体解释“容器—破坏者—发现者”检测策略的主要思路.

使用以 JAVA 语言编写的开源软件 PIPE[11] 作为研究对象,检测 TokenEditorPanel 类中的 updateTableAt 方法是否为 long method,该方法的度量指标如表 3 所列.

通过表 4 将各度量指标转化为输入信号,其中 n 为度量值的数量. 对表 4 作如下说明:

1）由于嵌套的使用,将加大程序的时间复杂度和空间复杂度,故存在嵌套的条件和循环语句,按照指数计算该语句的数量.

表 3　PIPE 中 updateTableAt 方法的相关软件度量值

Table 3　Relevant software metrics value of updateTableAt method in PIPE

软件度量指标	updateTableAt 方法
临时变量	2
switch 语句	0
代码行数	18
参数数量	3
条件	5
循环	1
注释语句[2]	2

2）当代编程实践中,例如极限编程,鼓励程序员少写注释,仅通过方法名称,就使代码用意一目了然. 过多的注释蕴含着方法的复杂晦涩. 鉴于此,将注释语句的数量作为 long method 的一个指标.

表 4　输入信号的获得

Table 4　Obtaining of input signals

软件度量指标(数量)	PAMP	危险信号	安全信号*
临时变量	2n	n	当 n<4,4n
switch	2n	n	当 n<3,4n;当 n=0,4
代码行数	n	n/2	当 n<15,5n
参数数量	3n	2n	当 n<4,4n
条件	4n	3n	当 n<4,5n
循环	2n	n	当 n<3,4n
注释语句	5n	2n	当 n=1,9

* 安全信号除所列值之外,在其他情况下均取 0

本实例中,表 2 权值矩阵设置如下:$W_1 = W_2 = 2$,λ = 2,μ = 1.5. 炎性因子信号 I 为 1.8,转换阈值为 498.

计算各输出信号 CSM、半成熟信号以及成熟信号,计算结果分别为 691.6、78.4、221.2. 由此,继续下列两步计算:

1）CSM 超过转换阈值,则 iDC 满足向 smDC 和 mDC 转换的条件.

2）成熟信号高于半成熟信号,则确定 iDC 向 mDC 转换. 判断 updateTableAt 方法为 code smells 中的 long method.

事实上,原作者在注释中所写的替代方法,已消除 updateTableAt 方法的 long method 特征.

同理,PIPE 中共有 189 个方法,其中 long method 有 14 个,则可根据公式(3)计算出 MCAV = 7.4%.

4　实　验

本实验将 CDE 策略应用于 JAVA 开源软件 GanttProject[12]. 其中,GanttProject 是一个跨平台的工程管理软件.

本实验重点关注以下三个问题:

1）本策略能否有效发现代码中的 code smells?

[2] updateTableAt 方法的注释包含一行文字,即 TODO:DO THIS IN A BETTER WAY(做这个可以有更好办法)和十六行代码. 本研究将这十六行代码,作为一句注释.

2）本策略是否优于其他方法？

3）本策略能否比较各种 code smells 对软件工程的影响程度？

4.1 实验设计

实验环境如下：操作系统为 WindowsServer 2012 64bit，处理器为 Intel Core i7 3.13Ghz，内存为 8GB. 开发语言为 JAVA，开发工具为 Eclipse.

对 GanttProject 进行 code smells 测试，由于其是比较大型开源工程，使用其中 1124 个方法，132 个类，共计 21 万行代码. 本研究建立一个手工发现的 code smells 数据库 Gold Standard Database（GSD），该数据库由五名计算机专业研究生（其中三名有至少两年的软件行业从业经验，五人均至少已经使用 JAVA 语言五年）手工分析 GanttProject 中的 codesmells.

选择 long method、lazy class、speculative generality 和 refused bequest 等四种 code smells 类型对 GanttProject 分别使用 CDE 检测策略和 DECOR 方法（对照组）进行检测，实验过程如图 2 所示.

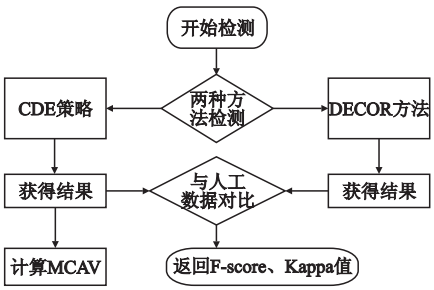

图 2 实验流程图

Fig. 2 Experiment flowchart

DECOR 首先定义领域特定语言（domain-specific language，DSL），然后使用基于 DSL 一致性的概念规约，诸如类的职责和结构等，描述 code smells 症状，最后将其映射成检测算法[13]. 其内核通过 Ptidej 工具集成到 Eclipse 中.

4.2 结果与分析

实验结果如表 5 所示.

对两种检测方式相应于人工检测的结果，求精确值和返回值，分别如公式（4）和（5），

$$\text{精确值} = \frac{\text{自动检测的 code smells 属于 GSD 的数量}}{\text{自动检测的 code smells 总数量}} \quad (4)$$

$$\text{返回值} = \frac{\text{自动检测的 code smells 属于 GSD 的数量}}{\text{GSD 中 code smells 的总数量}} \quad (5)$$

在多数情况下，不能保证精确值与返回值都取得较高值，因而需要使用 F-score，如公式（6）所示，F-score 是精确值与返回值的调和平均值，F-score 值介于 0 到 1 之间，0 为最差，1 为最优.

$$\text{F-score} = 2 * \frac{\text{精确值} * \text{返回值}}{\text{精确值} + \text{返回值}} \quad (6)$$

分析结果如表 6 所示.

DECOR 和 CDE 对四种 code smells 类型检测的结果求平均值，如下页表 7 所列，CDE 策略的精确值、返回值和 F-score 分别比 DECOR 高 0.187、0.522 和 0.440. 此研究中返回值与真阳性率是同一概念，说明 CDE 策略具有良好的真阳性率.

表 5 实验结果

Table 5 Experiment′s results

			手工检测		
			检测出	未检测出	
Long Method	DECOR	检测出	35	28	63
		未检测出	86	975	
			121		
	CDE		手工检测		
			检测出	未检测出	
		检测出	85	18	103
		未检测出	36	985	
			121		
Lazy Class	DECOR		手工检测		
			检测出	未检测出	
		检测出	2	1	3
		未检测出	7	122	
			9		
	CDE		手工检测		
			检测出	未检测出	
		检测出	6	1	7
		未检测出	3	121	
			9		
Speculative Generality	DECOR		手工检测		
			检测出	未检测出	
		检测出	4	5	9
		未检测出	9	114	
			13		
	CDE		手工检测		
			检测出	未检测出	
		检测出	11	4	15
		未检测出	2	115	
			13		
Refused Bequest	DECOR		手工检测		
			检测出	未检测出	
		检测出	6	8	14
		未检测出	20	98	
			26		
	CDE		手工检测		
			检测出	未检测出	
		检测出	24	7	31
		未检测出	2	99	
			26		

表 6 检测的精确值、返回值和 F－score 分析

Table 6 Detection′s precision，recall and F－score analysis

		精确值	返回值	F-score
Long Method	DECOR	0.556	0.289	0.380
	CDE	0.825	0.702	0.759
Lazy Class	DECOR	0.667	0.222	0.333
	CDE	0.857	0.667	0.750
Speculative	DECOR	0.444	0.308	0.364
Generality	CDE	0.733	0.846	0.785
Refused Bequest	DECOR	0.429	0.231	0.300
	CDE	0.774	0.923	0.842

DECOR 和 CDE 两种方法分别与人工检测结果的相符合

程度可以使用 Cohen′s Kappa 分析. Kappa 分析是比较两组数据相符一致性的一种统计方法,其值属于 0(完全不同)到 1(完全相同)之间,值越高,表示自动检测方式的结果与人工结果越吻合,CDE 策略的平均值比 DECOR 高出 0.472,如表 8 所列.

表 7　两种方法对 code smells 检测的均值

Table 7　Averages of code smells′ detection by the two approaches

	精确值	返回值	F-score
DECOR	0.524	0.262	0.344
CDE	0.711	0.784	0.784

表 8　检测的 Kappa 分析

Table 8　Detention′s Kappa analysis

	DECOR	CDE
Long Method	0.331	0.732
Lazy Class	0.310	0.734
Speculative Generality	0.308	0.760
Refused Bequest	0.188	0.799
平均值	0.284	0.756

CDE 检测策略与 DECOR 方法在 Kappa 分析方面的对比,如图 3 所示.

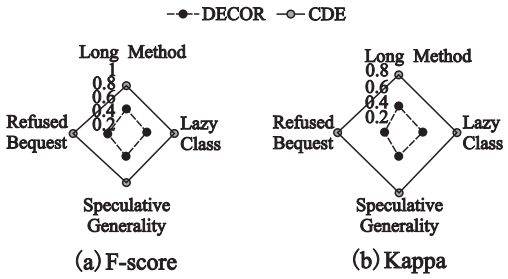

图 3　DECOR 和 CDE 的有效性比较

Fig. 3　Effectiveness comparison between DECOR and CDE

4.3　MCAV 值

通过公式(3),计算 MCAV 值(如表 9 所示)可以得出开源软件中 code smells 的严重程度,回答实验关注的第三个问题.

表 9　GanttProject 中四种 code smells 的 MCAV 值

Table 9　Four code smells′ MCAV value in GanttProject

	Long Method	Lazy Class	Speculative Generality	Refused Bequest
MCAV 值	0.092	0.053	0.114	0.235

Speculative generality 和 Refused Bequest 两种 code smells 类型是 GanttProject 的主要潜在重构问题,在 GanttProject 整个软件工程的环境中,需要重视此两种 code smells.

5　结　论

软件重构的研究与实践在学术界和工业界如火如荼的展开,需要被重构代码的检测作为其基础工作格外重要,本文所提出的“容器—破坏者—发现者”检测策略在多方面表现出优异性能.

至于将来工作,主要有两个方向. 首先,在短期,对更多在代码尺度和编程语言方面各不相同的软件进行 code smells 检测,以完善 CDE 策略.

其次,由于 code smells 定义的争议性,在更长远时间,工作将集中于其形式化定义以及探讨对检测到的 code smells 进行自动重构.

References:

[1] Liu Wei, Hu Zhi-gang, Liu Hong-tao. Singleton pattern directed automatic refactoring for source code [J]. Journal of Chinese Computer Systems, 2014, 35(12): 2664-2669.

[2] Fowler M, Beck K, Brant J, et al. Refactoring: improving the design of existing code [M]. Boston, MA: Addison-Wesley Professional, 1999.

[3] Kessentini W, Kessentini M, Sahraoui H, et al. A cooperative parallel search-based software engineering approach for code-smells detection [J]. IEEE Transactions on Software Engineering, 2014, 40(9): 841-861.

[4] Sahin D, Kessentini M, Bechikh S, et al. Code-smell detection as a bilevel problem [J]. ACM Transactions on Software Engineering and Methodology, 2014, 24(1): 1-44.

[5] Hall T, Zhang M, Bowes D, et al. Some code smells have a significant but small effect on faults [J]. ACM Transactions on Software Engineering and Methodology, 2014, 23(4): 1-39.

[6] Bian Yi-xin, Wang Tian-tian, Su Xiao-hong, et al. A semantics-preserving amorphous procedure extraction method for C clone code [J]. Journal of Computer Research and Development, 2013, 50(7), 1534-1541.

[7] Bowes D, Randall D, Hall T. The inconsistent measurement of message chains [C]. 4th International Workshop on Emerging Trends in Software Metrics (WETSoM), San Francisco, CA, 2013.

[8] Gu F, Greensmith J, Aickelin U. Theoretical formulation and analysis of the deterministic dendritic cell algorithm [J]. Biosystems, 2013, 111(2): 127-135.

[9] Ionita M G, Patriciu V V. Biologically inspired risk assessment in cyber security using neural networks [C]. 10th International Conference on Communications (COMM), Bucharest, 2014.

[10] Mohamad Mohsin M F, Abu Bakar A, Hamdan A R. Outbreak detection model based on danger theory [J]. Applied Soft Computing, 2014, 24: 612-622.

[11] Tattersall S. PIPE-Platform independent petri net editor [EB/OL]. https://github.com/sarahtattersall/PIPE, 2014.

[12] Thomas A, Barashev D. GanttProject [EB/OL]. http://www.ganttproject.biz/, 2014.

[13] Moha N, Gue X, He X, et al. DECOR: a method for the specification and detection of code and design smells [J]. IEEE Transactions on Software Engineering, 2010, 36(1): 20-36.

附中文参考文献:

[1] 刘　伟,胡志刚,刘宏韬. 单例模式导向的源代码自动重构研究 [J]. 小型微型计算机系统,2014,35(12):2664-2669.

[6] 边奕心,王甜甜,苏小红,等. 一种语义保持的克隆代码无定型过程提取方法[J]. 计算机研究与发展,2013(7):1534-1541.