Code Smell 视角下分层 Web 应用失血及充血现象的量化分析

黄子杰,陈军华,高建华
(上海师范大学计算机科学与技术系,上海 200234)

摘 要: 分层 Web 应用的领域层由领域模型构成.仅存储数据且不实现行为的领域模型被称作失血领域模型,其状态和数据由非领域层中的类维护,造成后者的充血现象.失血和充血现象损害了 Web 应用的可维护性,然而,由于缺乏量化分析和研究,其论据多基于主观经验得出.本文度量三种 Code Smell(Feature Envy、Blob 和 Data Class)的强度,将它们作为衡量失血和充血现象的标准,对现象进行量化分析,进而得出它们之间的相关性.本文对一个公开数据集的 91 个 Java Web 项目及 10 个不同领域的开源 Java Web 应用的多个版本进行了实验,实验发现至少有 75% 的项目受领域层失血和服务层充血现象的影响,这些现象极少被解决或减弱,两者的强度在不同类间存在相关性,且两者强度的增量在同一软件项目的不同版本间亦存在相关性.

关键词: 软件可维护性; web 应用; 分层结构; code smell; 领域建模
中图分类号: TP311.5 **文献标识码:** A **文章编号:** 0372-2112 (2020)04-0772-09
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2020.04.019

Quantifying Anemia and Bloodshot of Layers in Web Applications from the Perspective of Code Smell

HUANG Zi-jie, CHEN Jun-hua, GAO Jian-hua
(*Department of Computer Science and Technology, Shanghai Normal University, Shanghai* 200234, *China*)

Abstract: Domain Models shape the Domain Layer of Web Applications. Anemic Domain Models (ADMs) are Domain Models holding only data. States of ADMs are maintained by classes in other layers, causing the latter bloodshot. However, there lacks research revealing the significance of impact that anemia and bloodshot of layers pose on maintainability. To quantify the significance, this paper assesses intensity of 3 Code Smells (Feature Envy, Blob and Data Class) as evaluation standards. Through an experiment conducted on 91 Java projects and multiple releases of 10 Java Web applications, this paper concludes that over 75% of the projects are affected. As the impact persists, correlations of the intensities exist among different classes of a project as well as same classes in different releases of a project.

Key words: software maintainability; web application; layered architecture; code smell; domain modeling

1 引言

分层 Web 应用包含基础设施(Infrastructure)层、领域(Domain)层、应用层(Application Layer,亦称 Service Layer[1],即服务层)和用户界面(User Interface)层[2],如图 1 所示.

Evans[2]认为,Web 应用的核心逻辑应集中于领域层.领域层由领域模型(Domain Model)构成,领域模型是包含领域数据和行为的对象模型[3].失血领域模型(Anemic Domain Model,ADM)是仅保存数据、无任何领域行为的领域模型[3].

Fowler 指出[1,3],在 Web 应用的功能设计中,ADM 是一种常见的反模式(Anti-Pattern),若在分层结构中使用 ADM 建构领域模型,本应属于领域层的核心逻辑会向服务层转移,因此服务层将难以保持精简.据此,本文将领域层的失血现象归纳为:在领域层中使用 ADM,并造成服务层或其它层的充血.

领域层失血和服务层充血的现象(简称失血和充血

收稿日期:2019-02-18;修回日期:2019-08-06;责任编辑:孙瑶
基金项目:国家自然科学基金(No.61672355)

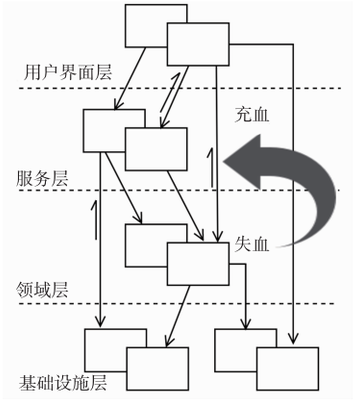

图1　Web应用的四层模型

现象)对代码的可维护性构成长期且渐进的影响. 领域行为被分散到服务层中,但行为的实现仍依赖 ADM 的数据及结构信息,造成服务层对领域层的高耦合. 同时,服务层的体积和复杂性日渐增加,内聚性也随之降低,失去面向对象程序的特性,演变为面向过程风格的代码[1].

然而,对 ADM 的评价基于主观的经验,缺乏相关研究量化 ADM 在开发和维护过程中造成的不良影响. 综上,选取一个可量化的角度分析失血和充血现象的强度及其影响的广泛性是迫在眉睫的.

Code Smell 是软件程序中存在不良设计和不良实现的征兆[4]. 准确地检测和修正 Code Smell 可以指导软件重构、提高软件的可用性和可靠性. 失血和充血现象是领域模型设计不当产生的后果,故 Code Smell 可作为分析 Web 应用失血和充血现象的研究角度. 在 Fowler[5]提出的面向对象程序的 22 种 Code Smell 中,包含 Feature Envy、God Class(亦称 Blob)和 Data Class. Feature Envy 面向类的高耦合问题、Blob 面向类的低内聚问题、Data Class 面向类仅存放数据的问题. 高耦合、低内聚的充血类符合Feature Envy、Blob的特征,仅存储数据的ADM符合 Data Class 的特征. 本文的主要贡献有如下四点.

(1)量化了失血和充血现象. 将 Feature Envy、Blob 和 Data Class 三种 Code Smell 量化为强度指标,据此提出了一种量化分析 Web 应用失血和充血现象的方法,并实现了分析工具.

(2)分析得出失血和充血现象在 Web 应用中普遍存在. 通过分析一个公开数据集中 91 个采用 MVC 设计模式的 Java Web 项目源码和 10 个分布于不同应用领域的 Java Web 应用源码,指出至少有 75% 的应用受到失血和充血现象的影响.

(3)分析得出失血和充血现象存在正相关性且极少减弱. 本文通过对上述 10 个 Java Web 应用的 96 个发布版本(Release)源码分析得出:宏观上,对于项目整体,失血和充血现象随着应用版本的迭代持续加强;微观上,对于项目的每个类,现象几乎从不减弱;在版本迭代的过程中,失血和充血现象的增强亦存在正相关性.

(4)实验结果表明 ADM 不仅引起领域内部的失血、充血问题,而且带来领域之间的耦合问题. 实验揭示了大部分使用 ADM 的应用没有实现彻底分离业务逻辑和数据的设计,仍有 4 至 7 成的领域模型包含不完全的领域行为,损害了程序的可理解性.

2　基于失血领域模型的分层 Web 应用

图 2 展示了从开源 Web 应用 Shopizer[6] 中析出的多个与订单领域相关的类及其依赖关系,算法 1 和算法 2 展示了其中两个类的代码. 算法 1 的 Order 类位于图 2 中的领域层. 算法 2 的 OrderServiceImpl 类处在图 2 中的服务层,它实现了 OrderService 接口.

ADM 的主要组成部分为类的属性(Properties),除存取方法外,ADM 仅有少数或没有方法. 算法 1 中灰色部分标明了 Order 类的 ADM 特征,该类仅有存取方法. 通过 调用存取方法,充血服务类可维护一或多个ADM

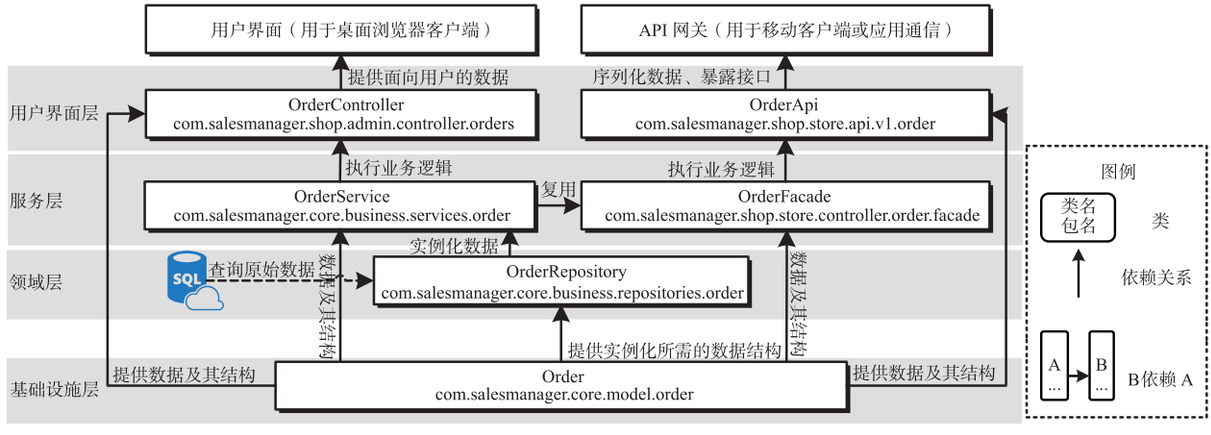

图2　Shopizer订单(Order)领域相关类的依赖关系

算法 1 领域层类 Order. java 的示例代码

```
1. @ Entity @ Table (name = "ORDERS)
2. public class Order {
3.   @ Column (name = "CUSTOMER_ID")
4.   private Long customerId;
5.   @ OneToMany(mappedBy = "order")
6.   private Set < OrderStatusHistory > orderHistory;
7.   public Long getCustomerId() {
8.     return customerId;
9.   }
10.  public void setCustomerId(Long customerId) {
11.    this. customerId = customerId;
12.  }
13.  public Set getOrderHistory() {
14.    return orderHistory;
15.  }
16.  public void setOrderHistory(Set orderHistory) {
17.    this. orderHistory = orderHistory;
18. }}
```

算法 2 服务层类 OrderServiceImpl. java 的示例代码

```
1. @ Service("orderService")
2. public class OrderServiceImpl implements OrderService{
3.   @ Inject private CustomerService customerService;
4.   @ Inject private OrderRepository orderRepository;
5.   private Order process(Order order,Customer customer) {
6.     Set < OrderStatusHistory > statusHistorySet = new HashSet < >
       ();
7.     OrderStatusHistory statusHistory = new OrderStatusHistory()
8.       . setStatus(OrderStatus. ORDERED)
9.       . setDateAdded(new Date())
10.      . setOrder(order);
11.    statusHistorySet. add(statusHistory);
12.    order. setOrderHistory(statusHistorySet);
13.    customer = customerService. create(customer);
14.    order. setCustomerId(customer. getId());
15.    return orderRepository. create(order);
16.  }
17. }
```

并实现领域行为. 算法 2 的 process 方法实现订单初始化及创建的领域行为,图中灰色部分标明了其充血的代码特征. 根据算法 2 中的第 5、10、12、14 行,易见 Order 类中的两个属性均由 OrderServiceImpl 类维护.

将方法隔离在对象外的设计抛弃了封装(Encapsulation)的面向对象特性. 尽管彻底割裂逻辑和数据的设计较为直观,但 Order 领域模型却因此失去了应有的行为. 例如:Order 类的默认状态(例如 OrderStatusHistory)应由 Order 类的成员而非第三方维护;查找和关联 Customer 实例的行为应属于领域层的 Order 类.

3 量化分析方法

使用数据驱动的度量指标进行分析可适应软件系统的复杂性和多样性[7]. 量化分析失血和充血现象需要识别类所属的层,并根据度量指标计算 Code Smell 强度. 本文度量 3 种 Code Smell,即 Feature Envy、Blob 和 Data Class,其定义分别为:

(1)Feature Envy 表现为一个函数,与其所在类相比,它更依赖非所在类的操作[8];

(2)Blob 表现为一个类,它实现了多种不同职责,体积庞大且复杂,其函数间的内聚性低[8~10];

(3)Data Class 表现为一个仅存放数据的类[10].

3.1 检测 Feature Envy 强度

JDeodorant 方法[11]提出了一种检测 Feature Envy 的方式:若被检类对外部类的成员(函数、属性)的访问次数大于对自身成员的访问次数,则被检类对外部类成员发生耦合,可判定 Feature Envy 被检出.

对于被检类 $c_{\text{candidate}}$,获取其对自身成员(方法、属性)的访问次数并去重,将结果作为 ATLM(Access To Local Members)的值. 遍历系统的类全集 C,对于 C 中的每个元素 c_i,检测 $c_{\text{candidate}}$ 对 c_i 成员(函数、属性)的访问次数 acc_i,并依据 acc_i 的值对 C 中的类从大到小排序. 若排序第一的类非 $c_{\text{candidate}}$,则判定 $c_{\text{candidate}}$ 受 Feature Envy 影响.

ATFM(Access To Foreign Members)的值为 $c_{\text{candidate}}$ 中所有 acc_i 的求和结果.

Feature Envy 强度的计算方式如式(1):

$$I_{\text{fe}}(C) = \text{ATFM}(C) - \text{ATLM}(C) \tag{1}$$

3.2 检测 Blob 强度

DECOR[12]方法提出,检测 Blob 应考虑四个因素:(1)类的体积;(2)类的内聚性;(3)类名的文本特征;(4)类与 Data Class 的关联程度. 因为(3)和(4)将于 3.4 节考虑,所以本节仅面向(1)和(2).

类的体积可根据属性及方法成员的总数评估,即 NMD(Number of Methods Declared)与 NAD(Number of Attributes Declared)之和,NMD + NAD 度量的阈值取 Palomba 等人[8]文献中的固定值.

学者们提出了大量计算类内聚性的度量,经过权衡,本文使用 LCOM5(Lack Cohesion Of Method)度量,原因如下:(1)LCOM5 是 LCOM 系列度量中最完善的指标[13,14],LCOM 是使用最广泛的类内聚性度量[15];(2)尽管有文献报告存在 SCOM 等更有区分度的度

量[13],但它们和 LCOM5 等经典方法类似,均无法避免至少一种典型的区分错误[16],其优劣势因度量对象而异;(3)替换内聚性的度量可能破坏 DECOR 方法的完整性. LCOM5 的含义为:类访问自身成员越频繁,类的内聚性越强. 其值域为[0,2],数值越大,类的内聚性越低,检测阈值取第三分位点(75%)值[13].

如式(2),将 LCOM5 与属性及方法成员总数乘积的值作为 Blob 的强度:

$$I_{\text{blob}}(C) = (\text{NMD}(C) + \text{NAD}(C)) \times \text{LCOM5}(C) \quad (2)$$

3.3　检测 Data Class 强度

Lanza 等人[9]提出,Data Class 的判定需要同时满足两个条件:(1)类的复杂度低;(2)类有较多的存取方法及属性,且仅有少量或完全没有其它方法.

对于类的复杂度,本文采用 WMC(Weighted Method Count)度量;对于存取方法及属性的比例,本文采用 NOPA(Number of Public Attributes)与 NOAM(Number of Accessor Methods)之和及 WOC(Weight of Class)度量.

WMC 是类所有方法的环路复杂性(CYCLO)度量值总和,对于 CYCLO,本文参考开源工具 PMD[17]基于顺序代码遍历实现的计算方式. WOC 是类的有效(除构造器、存取方法外)公共方法总数与类的所有公共方法总数之比. NOPA 是类所有公共属性的数量,NOAM 是类所有存取方法的数量.

以上度量的检测阈值取 Lanza 等人的著作[9]中对商业软件统计得到的值.

Data Class 强度的计算方式如式(3):

$$I_{\text{dc}}(C) = (1 - \text{WOC}(C)) \times (\text{NOPA}(C) + \text{NOAM}(C)) \quad (3)$$

3.4　代码分层和领域推断

代码分层的代表性途径包括依赖分析[18]和文本特征分析[12,19,20]. 根据 Web 应用的分层特点,本文参考上述文献提出一种综合的分层方式.

首先生成类的依赖有向图. 为系统中的每个类均生成 1 个结点,对于任意 2 个不同的类,若存在依赖关系,则生成 1 条 2 个类对应结点间的依赖路径. 然后对依赖图进行分层. 先将图分为 3 层,将无出度和无入度的结点分别置于顶层(即用户界面层)和底层(即领域层),其余节点置于中间层. 移除同层结点的依赖路径后,使用相同方式,再次调整结点的层. 对于中间层的结点,忽略其与顶层和底层结点的依赖路径,使用相同方式将其分为 2 层:将分离出的底层作为一层(即基础设施层中的持久类),将中间层和顶层作为另一层(即服务层). 至此,可得到一个 4 层的依赖图. 对于层中未知角色的结点,据表 1 推断其为层数对应的角色. 将不在第 2 层的持久角色类归为工具类.

系统中的领域亦可根据依赖图和类名推断. 将领域层类的类名根据驼峰命名法的大小写特征进行分词,若其它层中类的类名包含分解所得的任意词语,且两个类存在依赖关系,则判定它们属于同一领域. 除领域层的类外,领域应包含至少一个属于其它 3 层的类. 否则,应根据依赖关系将领域层的类合并至其它领域或丢弃.

表 1　基于类名推断类角色的方式

角色/推断方式	领域	持久	服务	用户界面	工具	未知
类或包名	小写包含 domain、vo、entity、entities,或包含 DO.	小写包含 dao,repo 或 repository.	小写包含 service.	小写包含 controller,ctrl,api 或 servlet.	小写包含 util.	不具上述特征的类.
扩展名	. java	. java	. java	. java	. java	任意
预期层数	1	2	3	4	N/A	N/A

3.5　基于 Code Smell 的强度量化失血和充血现象

获得类的 Code Smell 强度并对其分层后,即可进行失血和充血现象的判定.

属于领域层的 Data Class 类是 ADM. 若 ADM 同时与某个非领域层类发生耦合,则 ADM 发生强度为 I_{dc} 的失血现象.

若 Feature Envy 类属于服务层且与 ADM 耦合,则判定其为充血的服务层类. 其充血程度分为低内聚程度 I_{blob} 和高耦合程度 I_{fe}. 特别地,对于和多个 ADM 耦合或检出 Blob 的充血类,判定其发生严重的充血现象.

4　实验

本文使用开源代码质量评估工具 PMD 6.6.0[17]检测 Code Smell 的强度,并通过 Python 脚本统计度量信息和生成图表. 实验对 PMD 作了少量修改:(1)自行实现了 LCOM5 的计算;(2)PMD 内置的 ATFD 指标仅统计属性且过滤了函数,本文根据 ATFM 定义取消了其对函数的过滤.

实验试图解答以下六个问题.

Q1:本文的分层方式以及对失血和充血现象的检测是否准确?

Q2:失血和充血现象在 Web 应用中是否严重、是否普遍?

Q3:失血现象和充血现象是否相关?

Q4:在软件版本的迭代过程中,失血和充血现象呈现何种发展趋势?

Q5:是否存在比 ADM 更好的领域建模方式?

Q6:使用应用框架是否导致失血和充血现象?

4.1 实验数据集

本节描述实验所用的 2 个数据集.

数据集 1 可被用于回答 Q1 至 Q3: Aniche 等人[19]使用查询工具 BOA 自开源社区 GitHub 随机采集了 120 个使用 MVC 设计模式的 Java 开源项目,其中有 8 个项目已被原作者移除或隐藏,含有效项目共 112 个. 112 个项目中存在 16 个非 Web 应用,在余下的 96 个 Web 应用中,有 5 个应用没有采用分层结构,因此数据集共有 91 个应用.

数据集 1 的采集以"是否存在 10 个以上的控制器类"为标准[19],未考虑程序的使用率、通用程度、使用场景和开发社区的活跃度. 其中,37 个项目的源码仓库包含多于 1 次的 Release 信息,仅有 6 个分层 Web 应用在 1 年内仍有开发活动且用于专业用途. 因此数据集 1 难以全面回答 Q3 至 Q6,需要另外采集数据.

数据集 2 的采集以数据集 1 中 Commit 和 Fork 数最多的应用 Shopizer 为标准,另外挑选了 GitHub 中近六个月至少有一次代码提交的 9 个分层 Java Web 项目进行分析,共计 10 个项目. 对于每个项目,实验分析其最近的 10 个发布版本,若发布版本不足 10 个,则分析其所有版本,共计 96 个版本.

在数据集 2 的 10 个项目中,Dataverse 和 Fenixedu-academic 用于回答特定的研究问题,它们的数据不被计入任何量化的结论.

4.2 量化工具的准确性

为了回答 Q1,本文从数据集 1 中随机选择了 10 个项目以确认量化工具的准确性.

本文使用精确率(Precision)和召回率(Recall)度量检测的效果,指标的值越大说明检测效果越好. 令预测为正的正样本数量为 TP(True Positive,正确率)、预测为正的负样本数量为 FP(False Positive,误报率)、预测为负的正样本数量为 FN(False Negative,漏报率),精确率和召回率分别如式(4)和式(5):

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5}$$

表 2 列出了分层算法的检测效果. 实验对每个项目分别计算指标后,统计每种指标的均值和方差.

表 2 分层方式的准确性结果

类型/度量值	领域模型类	服务层类	界面层类	领域
精确率:均值/方差	98.23%/0.002	95.25%/0.006	93.09%/0.003	97.36%/0.005
召回率:均值/方差	98.48%/0.001	96.62%/0.002	98.70%/0.000	97.21%/0.002
总样本数	1966	682	1046	222

表 2 中领域数据的精确率仅体现领域的识别错误,而非领域内部的分层错误. 表 2 的对比基准为人工检测,由本文的第一作者及一名拥有三年工作经验的 Web 应用开发者分别独立进行后,针对 14 例分歧讨论完成. 由于部分类的依赖和名称特征不明显,实验出现了部分漏检和错检. 发生误判的主要原因是无法精确识别工具类,导致难以将其和界面、服务层的类区分,这一问题同时造成识别出的 218 个领域样本中,5 个领域内部发生了分层错误. 综上所述,尽管难以做到完全准确,但本文的自动分层方法可以较精确地识别出 Web 应用中服务、界面、领域层的类及应用中存在的领域.

PMD 内置的度量指标已被开源社区和相关研究广泛验证和讨论[13,19]. 对于自行实现的 LCOM5 度量指标,本文通过两个工具[21,22]对 10 个 Web 应用的结果进行核对,并对存在误差的类进行了人工核查. 除了 9 例因处理静态类和内部类的方式不同导致的计算误差外,实验表明本文的工具可以准确计算 LCOM5 度量的强度值.

图 3 展示了 91 个项目的最新版本中除去测试用例和接口、抽象类后 5 种度量值的分布,并用实线标注了使用的阈值及其含义. 计算 Feature Envy 强度的 ATFM-ATLM 指标不使用与分布相关的阈值,故不于箱线图中

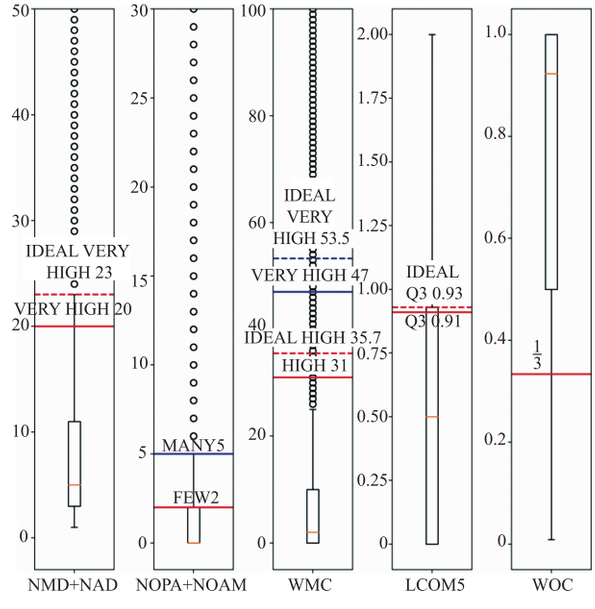

图3 91个Web应用度量值的分布及其阈值

展示. 为了明确本文所选阈值对数据集的适切性,实验根据相关文献[9,14]计算阈值的方法统计了 91 个项目的理想(Ideal)阈值,并在图 5 中用虚线标记.

综上所述,分析方法可以得出准确的基础数据. 可能影响检测效果的因素有:(1)项目体积过小或实现功能过于简单. 本文的检测阈值均参考相关文献基于企业级或商业软件项目的统计结果. 通过比对,亦发现这些阈值对数据集是适切的. 即便如此,检测不典型的应用仍需调整阈值;(2)项目的分层结构模糊. 尽管多数 Java Web 应用使用分层结构开发,应用中包和类的命名方式也遵循一定模式,但仍有部分例外情况(如数据集 2 中的 Dataverse),此时检测完全依据类的依赖信息完成,结果的精确度会降低.

4.3　实验过程和结果分析

为了回答 Q2,实验对数据集 1 项目最新版本中 ADM 和失血、充血类的比例进行检测.

数据集 1 的 91 个项目中受到失血和充血问题影响(ADM 比例和充血类比例均大于 0%)的项目共 71 个,占项目的 78.02%;ρ 值有效($P<0.05$,且 $0<|\rho|<1$)项目共 58 个,占有效项目的 63.73%. 表 3 列出了 ρ 值有效的 58 个项目的检测结果. 造成 ρ 值无效的原因主要有:(1)检测出的类数量不足;(2)两组数据确无关联,如 I_{dc} 和 I_{blob} 的相关性.

对于 Q2,在 91 个分层结构明显的 Web 应用中,有超过 75% 的项目受到失血和充血现象的影响. 在其服务层和用户界面层中,严重充血的类比例约占 50%,失血现象和充血现象普遍且严重.

表 3　对 58 个有效开源项目最新版本的检测结果

	类总数/领域数	充血类中严重充血类的占比		领域层类中 ADM 的数量占比	非领域层耦合中 ADM 的贡献率	I_{blob} 和 I_{fe} 的 ρ	I_{dc} 和 I_{blob} 的 ρ	I_{dc} 和 I_{fe} 的 ρ
		服务层	界面层					
均值	580.93/26.62	49.85%	51.10%	53.04%	44.52%	0.73	0.04	0.10
方差	275898.05/372.25	0.07	0.08	0.05	0.04	0.12	0.15	0.14

回答 Q3 需分析严重充血类的低内聚程度 I_{blob}、高耦合程度 I_{fe} 及其耦合对象 ADM 失血程度 I_{dc} 的关系. 本文利用 Spearman 等级相关方法(Spearman's rank correlation coefficient)[23]分析三者间的两两相关性,该方法可用于未知概率分布的两组顺序数据. 取任意两组顺序等长数据作为输入值,可输出其相关系数 ρ 及相关性的显著程度 P. 当 $P<0.05$ 时,可认为两组数据的差异具有显著性. 相关系数 ρ 的值域为 $[-1,1]$,绝对值越大相关性越强.

表 4 列出数据集 2 应用的基本信息和软件组成,表 5 列出数据集 2 的相关性系数 ρ. 若 $P\geq0.05$,实验将表 5 中的对应数据标为空缺. Dataverse 的分层特征较为模糊,每层的样本量较少,故在表 5 中产生了较多的空缺数据.

表 4　数据集 2 中应用的基本信息和项目组成

项目名	Commit/Fork/Release 次数	最新版本	类总数/领域个数	充血类中严重充血类的占比		领域层类中 ADM 的数量占比/%	非领域层耦合中 ADM 的贡献率/%	功能	业务框架
				服务层	界面层				
Shopizer	308/1192/6	2.2.0	811/31	72.22%	68.18%	60.97	68.85	电子商务	Spring Boot
OpenLegislation	3192/88/28	2.17	787/36	95.65%	61.11%	32.04	32.70	电子政务	Spring 4
LibrePlan	9657/148/32	1.4.1	1294/50	66.67%	100%	13.91	41.33	项目计划	Spring 4
OpenCMS	22750/321/228	10.5.4	3382/54	84.61%	76.92%	18.84	43.63	内容管理	–
Thingsboard	1514/676/17	2.1	797/31	100%	63.64%	8.70	35.67	物联网控制台	Spring Boot
Sakai	46898/535/21	12.3	4951/45	71.43%	92.00%	30.86	38.85	远程学习	–
OpenClinica	8521/167/30	4.5.2	1436/38	81.48%	65.06%	58.41	71.94	医疗机构管理	Spring4
Apollo	1944/2873/14	1.0.0	458/24	68.32%	78.27%	78.92	57.79	信息系统管理	Spring Boot
Dataverse	12042/180/30	4.9.2	675/19	33.33%	55.56%	42.30	44.30	数据开放平台	–
Fenixedu-academic	37921/102/344	22.2.2	3354/127	4.76%	14.81%	15.46	45.26	学籍管理	Fénix[24]

对于 Q3,表 3 和表 5 中相关性 ρ 值显示现象的强度是相关的:在同一版本的不同类间,严重充血类中低内聚和高耦合的强度、高耦合和失血现象的强度正相关;在不同版本的相同类间,三个指标的增量在多数情况下均两两相关. 参照表 4 和表 5 的充血类及 ADM 比例,可见失血和充血类的分布在数量上没有明确的相关性. 实验还统计了领域内部的界面、服务和领域层设计问题的关联,但未发现显著相关的数据,原因是领域的设计问题不仅源于领域内部.

表 5 数据集 2 中应用的相关性值

项目名	I_{blob} 和 I_{fe} 的 ρ	I_{dc} 和 I_{blob} 的 ρ	I_{dc} 和 I_{fe} 的 ρ	版本数	ΔI_{blob} 和 ΔI_{fe} 的 ρ	ΔI_{dc} 和 ΔI_{blob} 的 ρ	ΔI_{dc} 和 ΔI_{fe} 的 ρ	ΔI_{dec}
Shopizer	1.00	–	–	6	0.78	0.70	0.79	0.70%
OpenLegislation	0.81	0.71	0.94	10	0.67	–	0.31	0.00%
LibrePlan	–	–	0.85	10	–	0.73	0.64	3.84%
OpenCMS	0.80	–	–	10	0.95	0.58	0.47	1.72%
Thingsboard	0.99	0.62	–	10	0.81	0.61	0.74	2.38%
Sakai	0.94	–	0.63	10	0.87	0.51	0.38	1.47%
OpenClinica	0.57	–	0.71	10	0.70	0.28	0.51	0.45%
Apollo	1.00	0.96	0.96	10	0.70	0.36	0.42	1.11%
Dataverse	–	–	–	10	0.78	–	–	0.00%
Fenixedu-academic	–	–	–	10	–	–	–	3.57%

回答 Q4 需获取多个版本类中现象强度的增量(低内聚程度增量 ΔI_{blob}、高耦合程度增量 ΔI_{fe} 和领域层的失血程度增量 ΔI_{dc}),并分析增量间的关系. 实验忽略在软件迭代过程中被删除的类. 若类名变更,则将变更后的类视为新的类. 若包名变更,则将包名变更前后的类视作同一个类. 获取增量后,可计算失血或充血现象减少的类占比 ΔI_{dec},计算方式为"三个强度指标中任一出现负增长的类个数"与"发生失血和严重充血现象的类总数"之比.

对于 Q4,表 5 中 ΔI_{dec} 的值显示在项目的版本迭代中,失血和充血现象减弱的比例很小. 这也印证了相关研究[8,19]关于结构性(Structural)的 Code Smell 难以消除的结论.

回答 Q5 需统计各领域的类中耦合对象为 ADM 且 $I_{fe}>0$ 的耦合强度值,并进一步得出领域内的 ADM 对非领域层 I_{fe} 的平均贡献率.

对于 Q5,尽管部分文献[24,25]指出基于 ADM 的建模方式更易于理解,因而更容易维护,但本文的实验没有显示 ADM 在软件的维护阶段有类似优势. 实验发现由 ADM 建构的领域存在大量外源性的耦合问题,领域内部的失血和充血问题不仅严重,而且不会衰减. RDM 与 ADM 方式相反,使用领域数据和业务逻辑内聚的建模方式,Cemus 等人[26]发现 RDM 可大幅减少 ADM 带来的领域间设计问题.

表 4 中的 Fenixedu-academic 基于 RDM 开发[27],从其失血和充血类的占比情况可以看出,它受现象的影响显著小于其它基于 ADM 的应用. 亦有文献提及基于切面的建模方式是更好的实践[26,28],然而因为难以采集到合适的实验数据,故不作讨论. 对于本文实验未涉及的小型应用,因其业务逻辑简单,所以不会存在严重的失血和充血问题. 若维持规模不变,ADM 则可能体现其建模方式直观的优势.

对于 Q6,表 4 中基于 Spring 系列框架的 Web 应用存在更严重的失血和充血现象. 然而,现象在无框架的应用中仍会发生,原因可能是:一方面,基于应用框架的 Web 应用遵循框架既定的模式,其代码分层特征更为明确,故判定效果更好;另一方面,部分应用框架可以和对象关系映射(Object Relational Mapping,ORM)框架配合使用,使用代码生成器可根据数据库表结构便捷地生成适合 ORM 的 ADM,因此助长了 ADM 的使用[1].

5 结论和后续工作

本文将 Feature Envy、Blob 和 Data Class 三种 Code Smell 量化为强度指标,据此提出一种量化分析 Web 应用失血和充血现象的方法. 在实验部分,本文分析了一个公开数据集中的 91 个 Java Web 项目及 10 个 Java Web 应用,得出至少有 75% 的应用在不同程度上受失血和充血现象的影响,两种现象的强度存在正相关性. 在分析了上述 10 个 Java Web 应用的 96 个发布版本后,本文得出关于现象生命周期的结论:宏观上,对于项目整体,失血和充血现象随着应用版本的迭代持续加强;微观上,对于项目的每个类,现象几乎从不减弱;在版本迭代的过程中,失血和充血现象的增强亦存在正相关性. 虽然 ADM 是最常见的领域建模方式[29],但它不是最佳实践. 因其表达业务逻辑的局限性,ADM 彻底分离业务逻辑和数据的目的也几乎没有实现.

本文的后续工作包含四个方向:(1)拓展涉及的程序语言种类,探寻编程语言的类型系统等特性与失血和充血现象的关系;(2)根据代码的语义信息判定程序涉及的业务内容,进而提高分层精度;(3)改进检测算法对不同项目的适应性,对于不同体积和复杂度的项目,实现 Code Smell 检测阈值的动态调整;(4)分析开源项目中的代码提交和变更对失血和充血现象及 Web

应用可维护性的实际影响,更深入地揭示检测阈值和量化方式的实际效果.

参考文献

[1] Anemic Domain Model[EB/OL]. https://www.martinfowler.com/bliki/AnemicDomainModel.html,2019-08-06.

[2] Evans E. Domain-Driven Design:Tackling Complexity in the Heart of Software[M]. Boston:Addison-Wesley Professional,2004.

[3] Fowler M. Patterns of Enterprise Application Architecture[M]. Boston:Addison-Wesley Longman Publishing Co,Inc,2002.

[4] Tufano M,Palomba F,Bavota G,et al. When and why your code starts to smell bad (and whether the smells go away)[J]. IEEE Transactions on Software Engineering,2017,43(11):1063 – 1088.

[5] Fowler M,Beck K,Brant J,et al. Refactoring:Improving the Design of Existing Code[M]. Boston:Addison-Wesley Professional,1999.

[6] Shopizer[EB/OL]. https://github.com/shopizer-ecommerce/shopizer,2019-08-06.

[7] 李勇,黄志球,王勇,房丙午. 数据驱动的软件缺陷预测研究综述[J]. 电子学报,2017,45(4):982 – 988.
LI Yong,HUANG Zhi-qiu,WANG Yong,FANG Bingwu. Survey on data driven software defects prediction[J]. Acta Electronica Sinica,2017,45(4):982 – 988. (in Chinese)

[8] Palomba F,Panichella A,Zaidman A,et al. The scent of a smell:An extensive comparison between textual and structural smells[J]. IEEE Transactions on Software Engineering,2018,44(10):977 – 1000.

[9] Lanza M,Marinescu R. Object-Oriented Metrics in Practice:Using Software Metrics to Characterize,Evaluate,and Improve the Design of Object-Oriented Systems[M]. Berlin:Springer Science & Business Media,2007.

[10] Palomba F,Zanoni M,Fontana F A,et al. Toward a smell-aware bug prediction model[J]. IEEE Transactions on Software Engineering,2019,45(3):194 – 218.

[11] Fokaefs M,Tsantalis N,Chatzigeorgiou A. Jdeodorant:Identification and removal of feature envy bad smells[A]. Proceedings of IEEE International Conference on Software Maintenance(ICSM)[C]. NJ:IEEE,2007. 519 – 520.

[12] Moha N,Gueheneuc Y G,Duchien A F. Decor:A method for the specification and detection of code and design smells[J]. IEEE Transactions on Software Engineering,2010,36(1):20 – 36.

[13] Al Dallal J. Measuring the discriminative power of object-oriented class cohesion metrics[J]. IEEE Transactions on Software Engineering,2011,37(6):788 – 804.

[14] Henderson-Sellers B,Constantine LL,Graham I M. Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design)[J]. Object Oriented Systems,1996,3(3):143 – 158.

[15] Fernández L,Peña R. A sensitive metric of class cohesion[J]. International Journal Information Theories & Applications,2006,13:82 – 91.

[16] Izadkhah H,Hooshyar M. Class cohesion metrics for software engineering:a critical review[J]. Computer Science Journal of Moldova,2017,25(1):1 – 1.

[17] PMD[EB/OL]. https://pmd.github.io,2019-08-06.

[18] Hayashi S,Minami F,Saeki M. Detecting architectural violations using responsibility and dependency constraints of components[J]. IEICE Transactions on Information and Systems,2018,101(7):1780 – 1789.

[19] Aniche M,Bavota G,Treude C,et al. Code smells for model-view-controller architectures[J]. Empirical Software Engineering,2018,23(4):2121 – 2157.

[20] 孙小兵,刘湘月,李斌,张伟佳. 基于相关主题模型的程序网络自动构建与分析[J]. 电子学报,2017,45(5):1052 – 1056.
SUN Xiao-bing,LIU Xiang-yue,LI Bin,ZHANG Weijia. On automatic construction and analysis of program network via relational topic model[J]. Acta Electronica Sinica,2017,45(5):1052 – 1056. (in Chinese)

[21] Jpeek[EB/OL]. https://github.com/yegor256/jpeek,2019-08-06.

[22] Mendes T S,Gomes F G S,Gonçalves D P,et al. VisminerTD:a tool for automatic identification and interactive monitoring of the evolution of technical debt items[J]. Journal of the Brazilian Computer Society,2019,25(1):2.

[23] Zar J H. Significance testing of the Spearman rank correlation coefficient[J]. Journal of the American Statistical Association,1972,67(339):578 – 580.

[24] The Anaemic Domain Model is no Anti-Pattern,it's a SOLID design[EB/OL]. https://blog.inf.ed.ac.uk/sapm/2014/02/04/the-anaemic-domain-model-is-no-anti-pattern-its-a-solid-design/,2019-08-06.

[25] Wirfs-Brock R. Are software patterns simply a handy way to package design heuristics? [A]. Proceedings of the 24th Conference on Pattern Languages of Programs[C]. Vancouver:The Hillside Group,2017. 3:1 – 3:15.

[26] Cemus K,Cerny T,Matl L,et al. Aspect,rich,and anemic domain models in enterprise information systems[A]. Proceedings of the International Conference on Current Trends in Theory and Practice of Informatics[C]. Berlin:Springer,2016. 445 – 456.

[27] Martins J,Pereira J,Fernandes S M,et al. Towards a sim-

ple programming model in Cloud Computing platforms [A]. Proceedings of the 1st International Symposium on Network Cloud Computing and Applications [C]. US: IEEE, 2011. 83 – 90.

[28] 叶蔚,罗睿辞,刘学洋,张世琨. BuOA:一种企业级 Web 应用体系结构风格[J]. 电子学报,2013,41(11):2120 – 2126.
YE Wei, LUO Rui-ci, LIU Xue-yang, ZHANG Shi-kun. BuOA: An architecture style for enterprise web applications[J]. Acta Electronica Sinica, 2013, 41(11): 2120 – 2126. (in Chinese)

[29] Wang F, Yan L H, Zhou P, et al. The investigation of WEB software system based on domain-driven design [A]. Proceedings of the International Conference on Web Information Systems and Mining [C]. Berlin: Springer, 2011. 11 – 18.

作者简介

黄子杰 男,1994 年 2 月出生,上海人. 2016 年于上海师范大学获理学学士学位,现为硕士研究生,研究方向为软件测试、软件可靠性验证和数据挖掘.
E-mail:hzjdev@ foxmail. com

陈军华 男,1968 年 3 月出生,上海人. 副教授、硕士生导师. 现任上海师范大学信息与机电工程学院副院长. 研究方向为数据库理论及应用、分布式数据库等.
E-mail:chenjh@ shnu. edu. cn

高建华(通信作者) 男,1963 年 3 月出生,浙江绍兴人. 教授、博士生导师、CCF 高级会员. 1985 年湖南大学计算机应用专业本科毕业,1988 年上海科学技术大学计算机应用专业硕士研究生毕业,同年在东华大学信息科学与技术学院工作,1998 年东华大学控制理论与控制工程专业博士研究生毕业. 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.
E-mail:jhgao@ shnu. edu. cn