设计模式检测的映射机制分类研究

肖卓宇*,陈　果,徐运标
(湖南工业职业技术学院 信息工程学院,湖南 长沙 410208)

摘　要:针对传统设计模式检测领域多关注结构型设计模式,而缺乏对行为型设计模式、创建型设计模式,及设计模式演化后的设计模式共享实例和设计模式变体检测的问题,为提升设计模式检测精确率,提出一种设计模式映射机制分类检测方法。以结构型 Bridge 设计模式、行为型 Observer 设计模式和创建型 Factory method 设计模式为例,分类进行多阶段的设计模式参与者映射机制描述。引入设计模式子结构,通过 Transverse、Merging、Mapping 操作对设计模式参与者映射机制进行约束,分阶段融合设计模式参与者候选子结构,进而形成设计模式候选者实例,提出一种设计模式映射机制算法 DP_Mapping,构建了设计模式检测映射框架。以 QuickUML2001、Junit、JRefactory、JHotDraw 四种主流基准系统为实验平台,通过结构型 Bridge 和 Adpater 设计模式,行为型 Observer 和 Command 设计模式,及创建型 Factory method 和 Singleton 设计模式为对象,设计了两阶段的设计模式分类检测和设计模式共享实例检测实验。实验结果说明,该方案取得了较好的设计模式检测效果。

关键词:设计模式;设计模式检测;子结构;映射;分类
中图分类号:TP311　　**文献标识码**:A　　**文章编号**:1673-629X(2025)05-0045-09
doi:10.20165/j.cnki.ISSN1673-629X.2025.0003

Research on Classification of Mapping Mechanisms for Design Pattern Detection

XIAO Zhuo-yu*,CHEN Guo,XU Yun-biao
(School of Information Engineering,Hunan Industry Polytechnic,Changsha 410208,China)

Abstract:A classification detection method based on a design pattern mapping mechanism is proposed to address the limitations in traditional design pattern detection, which primarily focuses on structural design patterns but lacks adequate attention to behavioral, creational design patterns, as well as the detection of evolved design patterns, shared instances, and variants. To enhance the precision of design pattern detection, the proposed method takes the structural Bridge design pattern, the behavioral Observer design pattern, and the creational Factory method design pattern as examples to describe a multi-stage mapping mechanism for design pattern participants. By introducing design pattern substructures, the mapping mechanism is constrained through Transverse, Merging, and Mapping operations, which progressively integrate candidate substructures of design pattern participants to form candidate instances of design patterns. An algorithm named DP_Mapping is proposed, and a mapping framework for design pattern detection is established. Experiments are conducted on four benchmark systems, such as QuickUML2001, JUnit, JRefactory, and JHotDraw, using structural design patterns such as Bridge and Adapter, behavioral design patterns such as Observer and Command, and creational design patterns including Factory method and Singleton. The experiments are designed for two stages: classification detection of design patterns and detection of shared design pattern instances. The results demonstrate that the proposed approach achieves improved detection effectiveness for design patterns.

Key words:design pattern;design pattern detection;substructure;mapping;classification

0　引　言

软件设计旨在将需求转化为详细的系统实现,核心在于规划源代码的组织结构。鉴于设计决策的复杂性,设计模式应运而生,并作为解决常见设计难题的可复用方案[1-2]。设计模式检测(Design Pattern Detection,DPD)能助力软件分析,辅助软件系统设计决策,故众多研究围绕设计模式检测展开[3-5]。

Scanniello 等[6]研究了文档类型对源代码中设计模式实例理解度的影响。文献[7]提取设计模式集合中的问题域,并引入线索分阶段检测设计模式。

收稿日期:2024-10-23　　**修回日期**:2025-02-25
基金项目:湖南省自然科学基金(2024JJ8099)
作者简介:肖卓宇(1979-),男,教授,硕士,CCF 会员(49244M),通信作者,研究方向为程序理解、软件演化、知识工程、机器学习等。

Feitosa 等[8]通过众多 Java 类案例,评估了 GoF 设计模式检测方法的稳定性。文献[9]等分析了设计模式使用与系统结构复杂性的关系。Zhu 等[10]则利用软件演化数据,探讨了源码中设计模式类的缺陷。陈时非等[11]基于 Transformer 的跨语言设计模式分类模型,融合代码与自然语言理解,优化输入特征,实现设计模式分类。文献[12-13]提出设计模式特征的向量化表示,利用机器学习模型检测设计模式。文献[14-15]基于图论的子图同构思想,并结合距离相似积分检测设计模式。此外,一些研究逐步关注设计模式演化后的变体[14,16-17]、设计模式微结构特征等[9,18]的检测。

虽然研究取得了一定进展,但也存在瓶颈。为此,将现有工作存在的主要问题归纳如下:

(1)设计模式特征的约束映射机制欠完善,导致模式检测仅涵盖部分设计模式。

(2)由于设计模式参与者共享模式实例,导致设计模式检测的准确性不够理想。

(3)设计模式检测实验设计存在局限性,导致相关方法或工具的泛化能力偏弱。

为此,提出一种设计模式检测的映射机制分类研究方法,引入设计模式子结构,并基于 Transverse、Merging、Mapping 操作提出设计模式检测的映射机制算法,构建了设计模式检测框架。通过 QuickUML2001、Junit 3.7、JRefactory 2.6.24 和 JHotDraw 5.1 四种基准测试系统对结构型 Bridge 和 Adpater、行为型 Observer 和 Command、创建型 Factory method 和 Singleton,共计三类六种设计模式进行了检测实验。

该文的主要贡献如下:

(1)提出一种设计模式检测的映射机制分类研究方法。

(2)引入模式子结构思想,通过 Transverse、Merging、Mapping 操作构建了设计模式检测的映射机制框架。

(3)基于四种主流基准系统设计了设计模式分类检测和设计模式共享实例检测实验。

1 设计模式参与者映射机制

逆向工程视角下,设计模式检测对于增强代码复用性、提升可读性及灵活性具有重要意义。该检测的精确性高度依赖于中间表示——UML 类图,其中 UML 类和接口作为设计模式的构成要素(即:参与者)被应用。参与者间的交互关系直接映射了设计模式内部的结构联系,这些关系基于软件设计的初衷与意图构建,共同形成了各类不同的设计模式架构。

Gamma 等[19]提出将设计模式分为结构型、行为型及创建型三类。结构型模式聚焦于对象间的组合与架构,通过定义对象间的组装方式,增强系统的灵活性和可扩展性;行为型模式则深入探索类或对象间的交互机制与职责划分,确保系统行为的高效与协同;创建型模式则专注于对象的创建逻辑,力求以灵活、高效的方式生成对象实例。

为验证设计模式参与者映射机制的泛化性,以结构型 Bridge 模式、行为型 Observer 模式和创建型 Factory method 模式为例,分类进行映射机制描述。

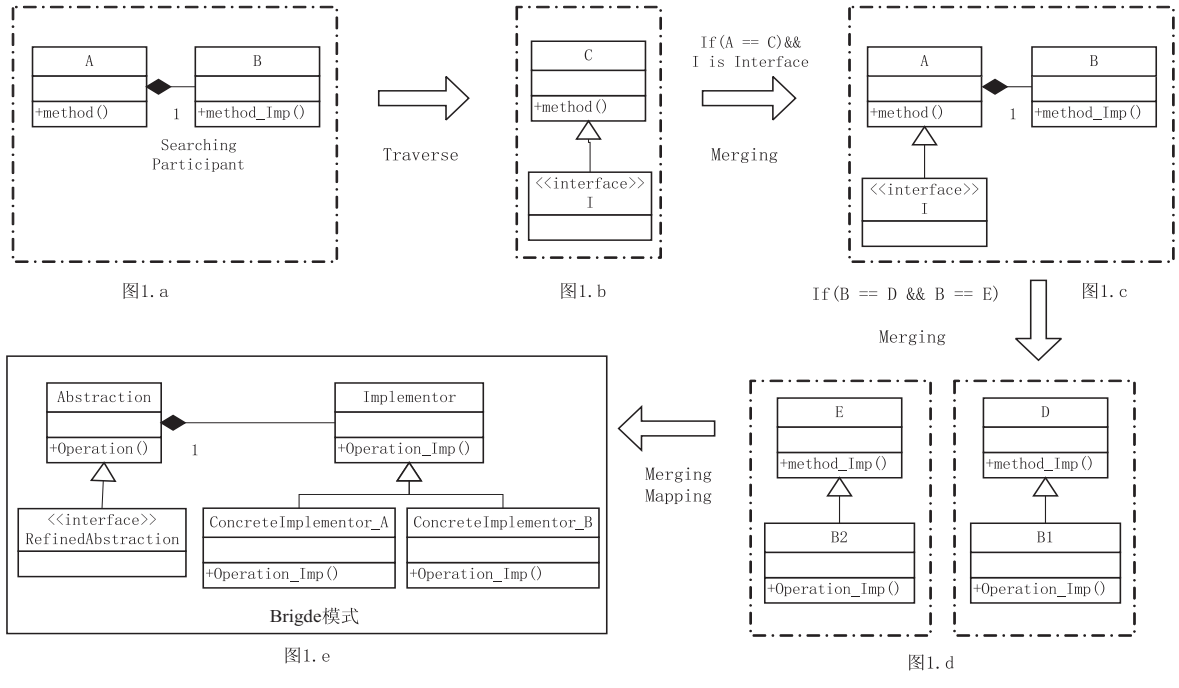

图 1 结构型 Bridge 模式映射机制

1.1　结构型模式映射

图 1 描述了 Bridge 模式映射机制。图 1. a 中类 A 搜寻与其存在聚合关系的类 B,当成功后,形成如图 1. a 所示的子结构 1。图 1. b 中接口 I 也在搜寻与其存在继承关系的类 C,如成功,则可形成如图 1. b 所示的子结构 2。此时,需要进一步判断图 1. a 中的类 A 和图 1. b 中的 C 是否扮演了 Bridge 模式的相同角色,如是同一角色,可将子结构 1 和子结构 2 进行融合,并形成图 1. c 所示的子结构 3。同理,图 1. d 中的类 E 与 B2、类 D 与 B1 存在派生关系,而类 E 和类 D 都扮演了 Bridge 模式的 B 角色。最终,通过归纳子结构的参与者类、接口及关系,形成了图 1. e 的结构型 Bridge 模式。

如表 1 所示,图 1 中的类 A 扮演了 Bridge 模式的 Abstraction 角色;图 1 中的类 B、类 D、类 E 扮演了 Bridge 模式的 Implementer 角色;图 1 中的接口 I 扮演了 Bridge 模式的 RefinedAbstraction 角色;图 1 中的类 B1 和 B2 扮演了 Bridge 模式的 ConcreteImplementer 角色。

表 1　Bridge 模式角色映射

设计模式参与者角色	映射类
Abstraction	A
Implementer	B,D,E
RefinedAbstraction	I,
ConcreteImplementer	B1, B2

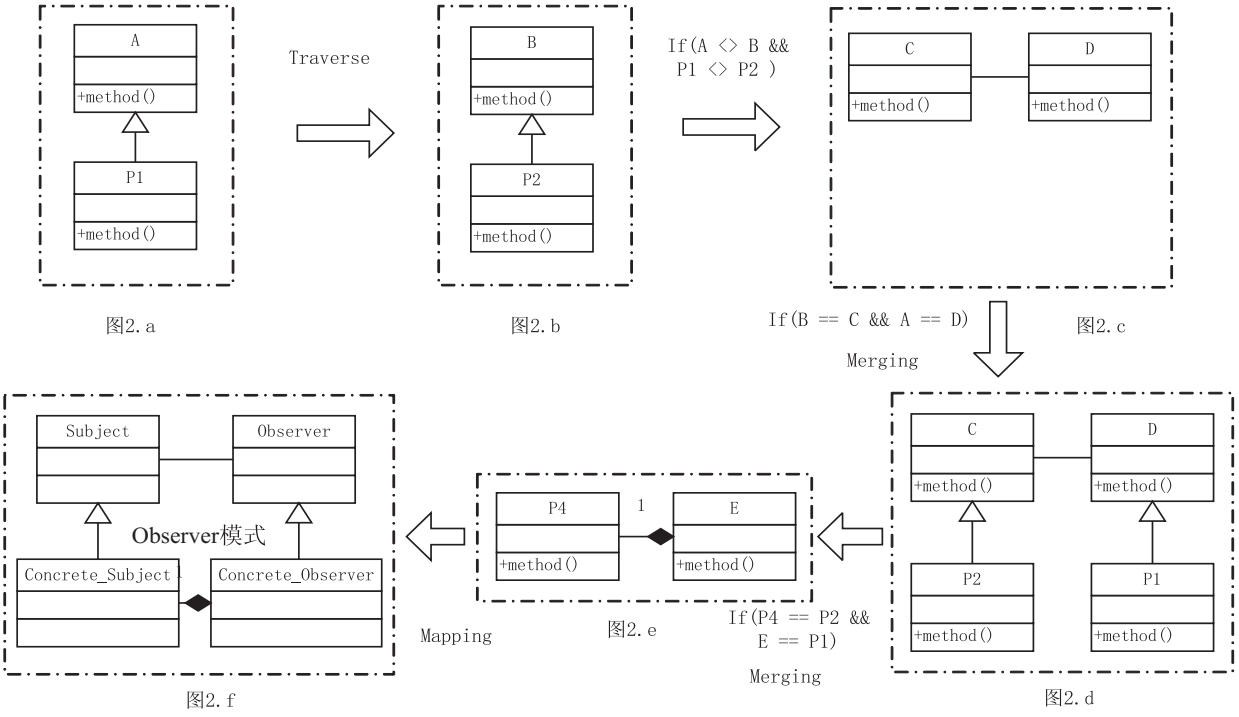

图2. a　图2. b　图2. c　图2. e　图2. f　图2. d

图 2　行为型 Observer 模式映射机制

1.2　行为型模式映射

图 2 描述了 Observer 模式映射机制。图 2. a 中类 A 搜寻与其存在继承关系的类 P1,当成功后,形成如图 2. a 所示的子结构 1。同时,图 2. a 中类 B 搜寻与其存在继承关系的类 P2,当成功后,形成如图 2. b 所示的子结构 2。进而搜索发现,图 2. c 中的类 C 与类 D 存在关联关系,并形成图 2. c 的子结构 3。类 A 与类 B 虽然是不同的参与角色,但二者分别扮演了图 2. c 中的类 C 和类 D 角色,进而形成了图 2. d 中的子结构 4。图 2. e 中的类 P4 和类 E 存在聚合关系,此外,类 P4、类 E 分别和图 2. d 的类 P2、P1 扮演相同的参与角色。最终,通过归纳子结构的参与者类、接口及关系,形成了图 2. f 的行为型 Observer 模式。

如表 2 所示,图 2 中的类 A、类 D 扮演了 Observer 模式的 Subject 角色;图 2 中的类 B、类 C 扮演了 Observer 模式的 Observer 角色;图 2 中的类 P4、类 P2 扮演了 Observer 模式的 Concrete_Subject 角色。图 2 中的类 E、类 P1 扮演了 Observer 模式的 Concrete_Observer 角色。

表 2　Observer 模式角色映射

设计模式参与者角色	映射类
Subject	A,D
Observer	B,C
Concrete_Subject	P4,P2
Concrete_Observer	E,P1

1.3　创建型模式映射

图 3 描述了 Factory method 模式映射机制。图 3. a 中类 A 搜寻与其存在继承关系的类 P1,当成功后,

形成如图 3. a 所示的子结构 1。同时,图 3. a 中类 B 搜寻与其存在继承关系的类 P2,当成功后,形成如图 3. b 所示的子结构 2。进而搜索发现,图 3. c 中的类 C 与类 D 存在聚合关系,并形成图 3. c 的子结构 3。类 A 与类 B 虽然是不同的参与角色,但二者分别扮演了图 3. c 中的类 C 和类 D 角色,进而形成了图 3. d 中的子结构 4。最终,通过归纳子结构的参与者类、接口及关系,形成了图 3. e 的创建型 Factory method 模式。

如表 3 所示,图 3 中的类 D、类 A 扮演了 Factory method 模式的 Create 角色;图 3 中的类 B、类 C 扮演了 Factory method 模式的 Product 角色;图 3 中的类 P1 扮演了 Factory method 模式的 Concrete_Create 角色。图 3 中的类 P2 扮演了 Factory method 模式的 Concrete _Product 角色。

表 3 Factory method 模式角色映射

设计模式参与者角色	映射类
Create	D,A
Product	B,C
Concrete_Create	P1
Concrete_Product	P2

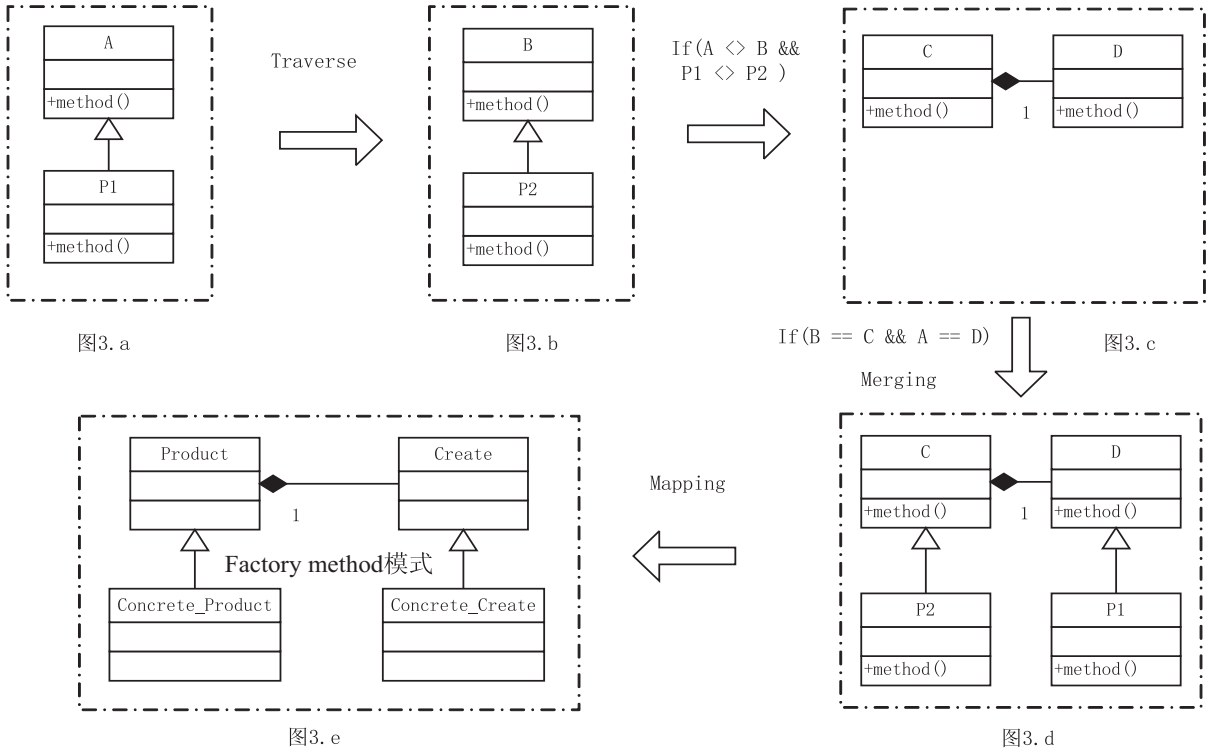

图3. a 图3. b 图3. c 图3. e 图3. d

图 3 创建型 Factory method 模式映射机制

2 设计模式映射机制算法

算法:DP_Mapping 描述了设计模式映射机制的处理流程。该算法以源码为输入,以获取的设计模式类别为输出。第 3 行表示通过工具 DeMIMA[20] 对源码进行抽取,并形成不同的 UML 类。第 4 行表示遍历抽取的每一个类,如该类是抽象类,则作为当前节点类。第 5 ~ 12 行表示,遍历与当前抽象类的存在关系的全部类;如不存在,则删除该抽象类节点,并选择一个新的抽象类进行作为当前类,进而遍历和新抽象类存在关系的类;如存在,则与该抽象类其存在关系的类进行合并(Merging),形成新的子结构。随着子结构的不断合并,子结构中的参与者及关系不断融合,其设计意图和动机愈加明显,并易于被捕获。第 13 行表示将 UML 类及关系映射为前期工作归纳的设计模式参与者特征指标 Metrics[21]。第 14 行表示将获取的特征指标 Metrics 和 P-mart[22] 库进行匹配。第 15 行表示将匹配后的设计模式标签进行返回,并完成设计模式检测。

算法 1:DP_Mapping (Design pattern Mapping)

1 :Input:Source code //输入 UML 类图

2 :Output:design paten label //输出设计模式分类

3: Extracting source code //抽取源码,并获取设计模式参与者类和接口

4: Traverse the first abstract class //遍历第一个发现的抽象类,并设置为当前访问类

5:Foreach *i* In Adjacent class //遍历当前访问类的每一个邻接点类

6:IF not exist(*i*) // 如果当前节点不存在相邻类

7: Remove Adjacent class // 对不存在相邻节点

的类进行删除

8:Else

9:　　Add Adjacent class　　//对存在相邻类的类进行添加

10:　　Merge classes to form substructures　　//对和当前类存在联系的类进行融合,并形成子结构

11:EndIF

12:EndFor

13:　Mapping design pattern participants to feature metrics //映射设计模式参与者特征指标

14:　Matching P-mart Repository　//匹配设计模式库 P-mart

15:Return design paten label　　// 返回检测得到的设计模式类型

框架,主要包括六个步骤:

STEP 1:使用工具 DeMIMA[20] 对源码进行抽取;

STEP 2:源码抽取后形成 UML 类图;

STEP 3:通过遍历(Transverse)、合并(Merging)和映射(Mapping)操作构建设计模式参与者子结构,并通过循环使子结构进一步扩展(Expanding),以形成具有明显设计意图或动机的新子结构;

STEP 4:通过前期工作[21]归纳的设计模式特征指标,将 STEP3 中获取的设计模式子结构进行映射;

STEP 5:将 STEP4 获取结果与设计模式规则库 P-mart[22] 匹配;

STEP 6:获取设计模式类型结果。

3　设计模式检测的映射框架

图 4 描述了设计模式检测的参与者映射分类研究

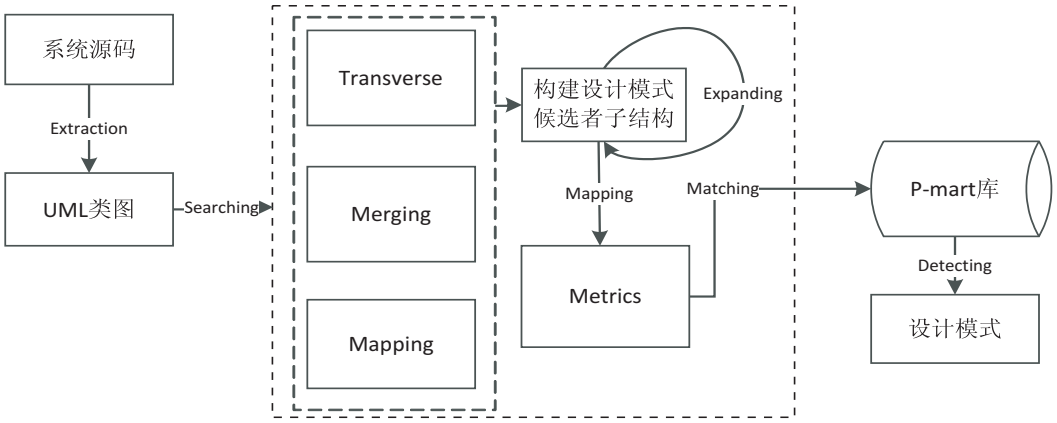

图 4　设计模式映射机制检测框架

4　实验设计

实验采用 JRefactory 2.6.24 等四个国内外相关研究具有广泛影响力的基准系统,具体参数包括在线网址、LOC、包数量、类数量、方法数、属性数、文件大小,详见表 4。其中,LOC 表示源码系统的代码行数,如 QuickUML2001 中的代码行数为 46 572 行。

表 4　测试系统主要参数

	QuickUML2001	Junit 3.7	JRefactory2.6.24	JHotDraw 5.1
网址	www.sourceforge.net/projects/quj/	www.junit.org	www.ant.apache.org	www.jhotdraw.org
LOC	46 572	9 742	216 244	30 860
包数量	13	9	58	16
类数量	204	43	562	136
方法数	1 082	425	4 881	1 314
属性数	422	114	1 367	331
文件大小/MB	3.39	1.46	12.5	4.85

4.1　设计模式映射机制检测

为验证检测方法的普适性与有效性,实验涵盖结构型 Bridge、行为型 Observer 及创建型 Factory method 模式。此外,考虑到模式的多样性,额外增加了结构型 Adpater、行为型 Command、创建型 Singleton 模式进行比较实验,见表 5、表 6。

表 5 中类型栏的 S、B、C 分别表示结构型(Structural,S)模式、行为型(Behavioral,B)模式、创建型(Creative,C)模式。“/”表示模式在该检测系统不存在。

表 5 QuickUML2001 和 Junit 3.7 设计模式映射机制检测

模式名称	类型	QuickUML2001					Junit 3.7				
		基准	文中	文献[15]	文献[14]	文献[12]	基准	文中	文献[15]	文献[14]	文献[12]
Bridge	S	1	1	1	0	1	0	0	0	0	0
统计/%			100	100	0	100		/	/	/	/
Adpater	S	29	26	18	20	22	11	10	8	8	9
统计/%			89.7	62.1	69	75.9		90.9	72.7	72.7	81.8
Observer	B	1	1	0	0	1	0	0	0	0	0
统计/%			100	0	0	100		/	/	/	/
Command	B	18	12	6	6	8	0	0	0	0	0
统计/%			66.7	33.3	33.3	44.4		/	/	/	/
Factory	C	18	11	2	4	5	2	1	0	0	1
统计/%			61.1	11.1	22.2	27.8		50	0	0	50
Singleton	C	1	1	0	0	1	0	0	0	0	0
统计/%			100	0	0	100		/	/	/	/

表 6 JRefactory2.6.24 和 JHotDraw 5.1 设计模式映射机制检测

模式名称	类型	JRefactory2.6.24					JHotDraw 5.1				
		基准	文中	文献[15]	文献[14]	文献[12]	基准	文中	文献[15]	文献[14]	文献[12]
Bridge	S	0	0	0	0	0	4	4	2	3	3
统计/%			/	/	/	/		100	50	75	75
Adpater	S	16	16	13	13	14	14	12	9	9	11
统计/%			100	81.3	81.3	97.5		85.7	64.3	64.3	78.6
Observer	B	0	0	0	0	0	2	2	1	1	1
统计/%			/	/	/	/		100	50	50	50
Command	B	25	21	5	11	14	14	10	2	7	9
统计/%			84	20	44	56		71.4	14.3	50	64.3
Factory	C	87	51	3	17	23	2	1	0	1	1
统计/%			58.6	3.4	19.5	26.4		50	0	50	50
Singleton	C	12	6	3	4	3	2	1	0	1	0
统计/%			50	25	33.3	25		50	0	50	0

通过分析表 5 和表 6,并结合图 5 ~8 可知:

(1)对于 QuickUML2001、JRefactory 2.6.24 等四个测试系统,包括本研究在内的四种工具或方法在检测设计模式时表现各异。图 5 ~8 描述了工作和文献[15]、文献[14]、文献[12]分别检测 QuickUML2001、Junit 3.7、JRefactory 2.6.24、JHotDraw 5.1 四个系统中的 Bridge、Adpter 等六个设计模式的精确率折线图。由图 5 ~8 可知,比较实验对 Bridge 和 Adapter 两类结构型模式检测成效较好,对 Observer 和 Command 两类行为型模式次之,而对 Factory Method 和 Singleton 两类创建型模式检测能力较弱。究其原因发现创建型模式涉及复杂创建逻辑,常隐于源码中,项目扩展加剧其隐蔽性与复杂性,检测难度大。行为型模式直观表现为对象交互,但多对象复杂协作亦增加检测挑战。结构型模式则通过明确、固定的对象组合方式,简化检测过程,其特征较易捕获。故创建型模式检测难度最高,行为型次之,结构型相对容易。

(2)相对较易检测的结构型模式实验也和期望存在偏差。尽管文中工作在 QuickUML2001 中成功识别了唯一的 Bridge 模式实例,但在 Junit 3.7 和 JRefactory 2.6.24 中该模式并未存在,表明基准实例不足。同时,JHotDraw 5.1 中结构型 Adapter 模式实例丰富,虽然本研究检测效果优于其他工具,但仍只检测出 14 个 Adapter 模式中的 12 个,未完全覆盖。研究发现,未被检测到的 Adapter 模式与 Command 模式存在共享参与者类的问题,即单一类同时参与两个设计模式的协作,类似现象也见于 Strategy 和 State 模式。此类模式间类的共享性问题对自动检测工具提出了挑

战。为此,后续在 4.2 节中设计了专题实验,专注于探究 Adapter 与 Command 模式间共享实例的检测策略,旨在提升模式检测的准确性和泛化能力。

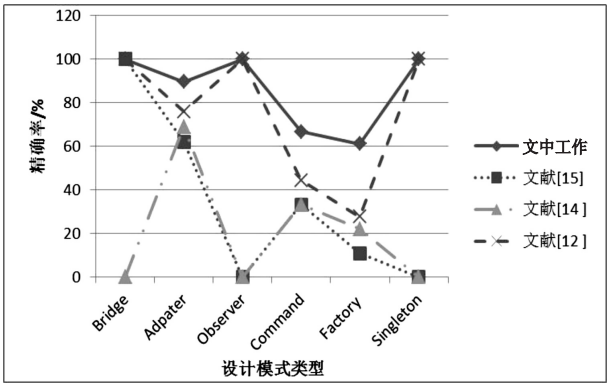

图 5　QuickUML2001 系统检测比较

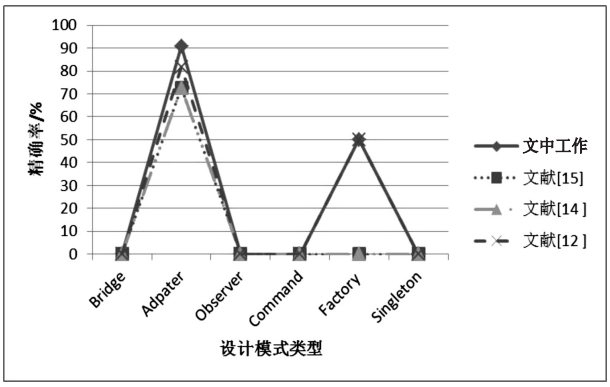

图 6　Junit 3.7 系统检测比较

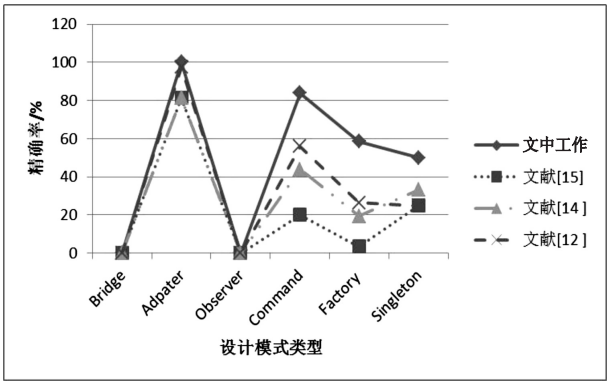

图 7　JRefactory2.6.24 系统检测比较

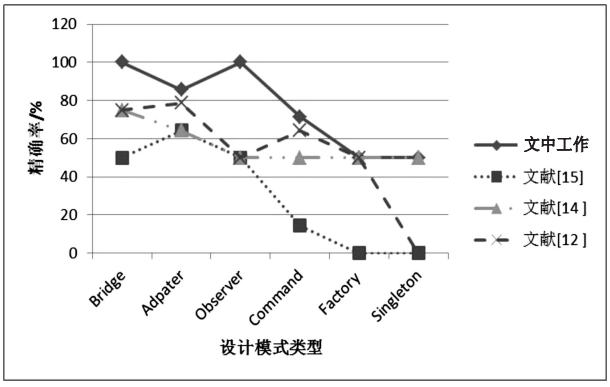

图 8　JHotDraw 5.1 系统检测比较

(3)标准设计模式演化后的变体因其灵活性与多样性,对检测准确性构成了显著挑战。变体间在结构构造、参与者角色及相互关系上的细微差异,导致传统检测方法难以精准捕捉,进而可能影响检测结果的精确性,增加误报或漏报的风险。通过深入研究发现,Adpater、Observer、Command、Singleton、Factory method 等模式都存在变体演化情形。该类问题,尤其是行为型和创建型模式变体更加难以被正确检测。后期拟进一步剖析模式变体实例特征,构建详尽特征值规则体系,以强化检测工具的检测能力及适应性。并有效提升检测精确率,降低假阴性和假阳性检测结果,优化设计模式检测性能。

4.2　设计模式参与者共享模式实验

设计模式参与者角色存在共享不同设计模式的场景,即一个类同时担任了不少于两个不同或相同实例中的设计模式参与者角色。如 Adapter 和 Command 模式、Strategy 和 State 模式易出现参与者角色共享实例的情形。考虑到众多研究以 JHotDraw 测试系统为基准,为此,本研究考虑以 JHotDraw 系统为例,展开设计模式参与者共享实例检测实验。

依据 GoF 设计模式规范[19],表 7 以人工方式归纳了 JHotDraw 系统中 Adapter 和 Command 模式共享实例的类。序号 S1 中 CH. ifa. draw. figures. TextFigure 和 CH. ifa. draw. standard. OffsetLocator 分别扮演了 Adapter 模式的 Adapter 和 Adaptee 的参与者角色;此外,CH. ifa. draw. figures. TextFigure 和 CH. ifa. draw. standard. – OffsetLocator 也分别扮演了 Command 模式的 Command 和 ConcreteCommand 的参与者角色。

表 8 展示了本研究与领域主流工作对表 7 中设计模式共享实例检测的对比实验。由表 8 可知:

(1)研究取得 75% 的检测精确率,优于文献[15]、文献[14]、文献[12]的 16.7%、41.7%、33.3%。究其原因,本研究着重关注设计模式参与者角色的映射机制,引入子结构思想,并通过合并扩展子结构的方式能一定程度上提升设计模式检测精确率。

(2)本研究对于 S9 和 S10 两个实例检测失败,此外,文献[15]、文献[14]、文献[12]也没有检测到这两个实例。通过深入研究发现,表 8 中 S9 和 S10 的参与者 CH. ifa. draw. standard. StandardDrawingView 同时扮演了 4 个参与者角色。即:S9 和 S10 两个 Adapter 模式的 Adapter 类参与者,此外,还扮演了两个 Command 模式的 Command 类参与者角色。该类存在不同模式不同实例的参与者共享实例情形较为复杂,导致本研究和其余三种方法都检测失败。未来工作将以此为契机,进一步优化检测方法。

(3)本研究对 JHotDraw 测试系统进行共享实例检测也出现了误差。见表 9,三个实例 S1、S2、S3 实质

上并不存在 Adapter 和 Command 模式共享参与者的情况,但却被误检测。经过分析发现,映射机制存在约束仍不够严谨的问题,后续还需进一步归纳,未来工作将进一步优化映射约束机制,以提升检测效度。

表 7 Adapter 和 Command 模式共享实例集

序号	参与者 1	参与者 2
S1	CH. ifa. draw. figures. TextFigure	CH. ifa. draw. standard. OffsetLocator
S2	CH. ifa. draw. applet. DrawApplet	CH. ifa. draw. standard. ToolButton
S3	CH. ifa. draw. standard. ToolButton	CH. ifa. draw. util. PaletteIcon
S4	CH. ifa. draw. figures. RadiusHandle	CH. ifa. draw. figures. RoundRectangleFigure
S5	CH. ifa. draw. standard. AlignCommand	CH. ifa. draw. framework. DrawingView
S6	CH. ifa. draw. contrib. PolygonHandle	CH. ifa. draw. framework. Locator
S7	CH. ifa. draw. figures. GroupCommand	CH. ifa. draw. framework. DrawingView
S8	CH. ifa. draw. standard. ChangeAttributeCommand	CH. ifa. draw. framework. DrawingView
S9	CH. ifa. draw. standard. StandardDrawingView	CH. ifa. draw. framework. Drawing
S10	CH. ifa. draw. standard. StandardDrawingView	CH. ifa. draw. framework. DrawingEditor
S11	CH. ifa. draw. standard. SendToBackCommand	CH. ifa. draw. framework. DrawingView
S12	CH. ifa. draw. figures. InsertImageCommand	CH. ifa. draw. framework. DrawingView

表 8 共享实例比较实验

共享实例	文中方法	文献[15]	文献[14]	文献[12]
S1	√	√	×	√
S2	×	×	√	√
S3	√	×	×	×
S4	√	×	×	×
S5	√	√	√	×
S6	√	×	√	√
S7	√	×	×	×
S8	√	×	√	×
S9	×	×	×	×
S10	×	×	×	×
S11	√	×	√	√
S12	√	×	×	×
统计%	75	16.7	41.7	33.3

表 9 被误检测的 Adapter 和 Command 模式共享实例

序号	参与者 1	参与者 2
S1	CH. ifa. draw. Standard. ConnectionTool	CH. ifa. draw. Standard. ConnectionFigure
S2	CH. ifa. draw. figures. AttributeFigure	CH. ifa. draw. figures. FigureAttributes
S3	CH. ifa. draw. Standard. HandleTracker	CH. ifa. draw. Framework. Handle

5 效度分析

(1)外部有效性威胁:由于众多编程语言目前缺乏有效设计模式规则库,本研究选择的四个测试系统皆为 Java 源码系统,故精确率等指标的结果在用不同编程语言检测可能会有所偏差。

(2)内部有效性威胁:设计模式参与者共享实例、变体的位置、数量等尚未达成有效共识。此外,子结构特征类型与模式定义也未覆盖 23 种标准设计模式及其变体,导致可能影响方法准确性。

(3)结构有效性威胁:一些新的编程语言特性或功能可能导致设计模式特征检测失配,进而影响了方法的准确性。

6 结束语

提出一种设计模式映射机制的分类检测方法,引

入设计模式子结构概念。通过 Transverse、Merging、Mapping 等操作构建设计模式检测的映射机制框架。研究在 JRefactory2.6.24 等四种主流基准系统的实验中展现出较好的检测效果,提升了设计模式检测的精确性,为相关领域的研究与应用提供了有力支持。未来工作将致力于设计模式共享实例规则库、变体规则库的构建。此外,将进一步丰富设计模式特征指标及检测算法优化的相关工作。

参考文献:

[1] NAGHDIPOUR A,HASHEMINEJAD S M H,BARMAKI R L. Software design pattern selection approaches:a systematic literature review[J]. Software:Practice and Experience,2023,53(4):1091-1122.

[2] LUCIA A D,DEUFEMIA V,GRAVINO C,et al. Detecting the behavior of design patterns through model checking and dynamic analysis[J]. ACM Transactions on Software Engineering and Methodology,2018,26(4):1-41.

[3] 肖卓宇,何　镕,徐运标,等. 引入线索约束的设计模式变体挖掘研究[J]. 计算机工程与科学,2021,43(6):1014-1023.

[4] MAYVAN B B,RASOOLZADEGAN A,EBRAHIMI A M. A new benchmark for evaluating pattern mining methods based on the automatic generation of testbeds[J]. Information and Software Technology,2019,59(1):60-79.

[5] PETTERSON N,LÖWE W,NIVRE J. Evaluation of accuracy in design pattern occurrence detection[J]. IEEE Transactions on Software Engineering,2010,36(4):575-590.

[6] SCANNIELLO G,GRAVINO C,RISI M,et al. Documenting design-pattern instances:a family of experiments on source-code comprehensibility[J]. ACM Transactions on Software Engineering and Methodology,2015,24(3):1-41.

[7] 肖卓宇,何　镕,余　波. 一种多阶段交互式线索驱动的设计模式识别方法[J]. 北京航空航天大学学报,2017,43(9):1746-1756.

[8] FEITOSA D,AMPATZOGLOU A,AVGERIOU P,et al. What can violations of good practices tell about the relationship between GoF patterns and run-time quality attributes?[J]. Information and Software Technology,2019,105(7):1-16.

[9] FONTANA F A,MAGGIONI S,RAIBULET C. Design patterns:a survey on their micro-structures[J]. Journal of Software:Evolution and Process,2013,25(1):27-52.

[10] ZHU H,BAYLEY I. An algebra of design patterns[J]. ACM Transactions on Software Engineering and Methodology,2013,22(3):23-61.

[11] 陈时非,刘　东,江　贺. 基于 CodeBERT 的设计模式语言模型[J]. 计算机科学,2023,50(12):75-81.

[12] ZANONI M,FONTANA F A,STELLA F. On applying machine learning techniques for design pattern detection[J]. Journal of Systems and Software,2015,88(5):102-117.

[13] CHIHADA A,JALILI S,HASHEMINEJAD S M H,et al. Source code and design conformance,design pattern detection from source code by classification approach[J]. Applied Soft Computing,2015,26(1):357-367.

[14] MAYVAN B B,RASOOLZADEGAN A,YAZDI Z G. The state of the art on design patterns:a systematic mapping of the literature[J]. Journal of Systems and Software,2017,125(3):93-118.

[15] ISSAOUI I,BOUASSIDA N,BEN-ABDALLAH H. Using metric-based filtering to improve design pattern detection approaches[J]. Innovations in Systems and Software Engineering,2015,11(1):39-53.

[16] 肖卓宇,何　镕. 多阶段可松弛的设计模式变体检测方法[J]. 华中科技大学学报:自然科学版,2018,46(1):26-31.

[17] 肖卓宇,何　镕,陈　果,等. 引入特征机制的设计模式变体挖掘方法[J]. 计算机工程与设计,2021,42(4):1020-1027.

[18] FONTANA F A,MAGGIONI S,RAIBULET C. Understanding the relevance of micro-structures for design patterns detection[J]. Journal of Systems and Software,2011,84(12):2334-2347.

[19] GAMMA E,HELM R,JOHNSON R,et al. Design pattern:elements of reusable object-oriented software[M]. India:Pearson Education,1995:1-22.

[20] GUÉHÉNEUC Y H,ANTONIOL G. DeMIMA:a multilayered approach for design pattern identification[J]. IEEE Transactions on Software Engineering,2008,34(5):667-684.

[21] 肖卓宇,何　镕,陈　果,等. 带特征指标约束描述的设计模式分类挖掘[J]. 山东大学学报:工学版,2020,50(6):48-58.

[22] GUÉHÉNEUC Y G. P-mart:pattern-like micro architecture repository[C]//Proceedings of the 1st EuroPLoP focus group on pattern repositories. Kloster Irsee:ACM,2007:1-3.