文章编号:1007-5321(2018)04-0119-06 **DOI**:10.13190/j.jbupt.2018-034

设计模式自动识别的研究进展

王智广[1,2], 王 雷[1]

(1. 中国矿业大学(北京) 机电与信息工程学院, 北京 100083;
2. 中国石油大学(北京) 地球物理与信息工程学院, 北京 102249)

摘要: 简要介绍了设计模式自动识别领域的发展历程;从基于图论、形式化技术、软件度量和人工智能的设计模式识别四大类方法出发,对该领域的研究进行了综述;通过适用的模式类型、时间性能和能否完全自动化 3 个指标对四大类方法进行了比较;最后分析了当前研究存在的问题与不足,并指出了今后的研究方向. 对各方法适用的模式类型、优缺点及内在联系进行了分析和讨论,从宏观上阐述了该领域的研究进展.

关 键 词: 设计模式识别; 研究进展; 软件逆向工程; 识别规则; 模式变体; 时间性能
中图分类号: TP311.5 **文献标志码**: A

Survey on Research of Automatic Design Pattern Detection

WANG Zhi-guang[1,2], WANG Lei[1]

(1. School of Mechanical Electronic and Information Engineering, China University of Mining and Technology (Beijing), Beijing 100083, China;
2. College of Geophysics and Information Engineering, China University of Petroleum (Beijing), Beijing 102249, China)

Abstract: The development history of the field of automatic design pattern detection was briefly introduced; the research of this field was respectively reviewed from four major categories of approaches: graph theory-based, formal technologies-based, software metrics-based and artificial intelligence-based design pattern detection; the four major categories of approaches are compared through 3 indexes: applicable pattern type(s), time performance and possibility of fully automated; finally the problems and shortcomings of the current research were analyzed, and the future research directions were pointed out. This paper analyzes and discusses the applicable pattern type(s), the advantages/disadvantages and the inner connections of the various approaches, and expounds the research progress of this field from the macro level.

Key words: design pattern detection; research progress; software re-engineering; detection rule; pattern variant; time performance

设计模式可更加方便地利用成功的设计和体系结构,用于大型软件项目的开发. 从源代码或统一建模语言(UML, unified modeling language)模型中自动识别出相应的设计模式,可以为面向设计模式的软件理解、维护和重构等活动提供自动化支持[1]. 因此,设计模式的自动识别成为目前逆向软件工程的一个研究热点.

近年来,国内外学者已经提出很多设计模式自动识别方法及其支撑工具. 根据这些方法的核心思想可将其大致分为四大类:基于图论的设计模式识别、基于形式化技术的设计模式识别、基于软件度量的设计模式识别和基于人工智能的设计模式识别.

收稿日期: 2018-02-02
基金项目: 国家自然科学基金项目(60873093); 国家科技重大专项项目(2017ZX05018-005)
作者简介: 王智广(1964—), 男, 教授, 博士生导师; 王 雷(1988—), 男, 博士生, E-mail: wanglei0823@foxmail.com.

笔者从这四大类方法出发对当前设计模式自动识别的研究进行了综述,侧重于分析和讨论各方法适用的模式类型、优缺点及内在联系,从而在宏观上将设计模式自动识别的研究进展展现出来.

1 设计模式自动识别的发展历程

设计模式是软件开发者在项目设计过程中总结出来的优秀设计的范例. 四人组(GOF, gang of four)设计模式[2]最早由 Gamma E 等提出,共包含23种设计模式,依据其目的可分为创建型、结构型和行为型3种类型. 创建型模式与对象的创建有关;结构型模式处理类或对象的组合;行为型模式对类或对象的交互和分配职责进行描述. 23种设计模式的详细目录见文献[2].

在GOF设计模式刚刚提出的几年里,就有学者对设计模式的自动识别进行了研究[3],此后越来越多的学者加入到设计模式识别方法中的研究中来. 随着软件系统规模越来越大,复杂性越来越高,时间性能的提升以及设计模式变体的识别又成为目前国内外学者普遍关注的问题. 近年来,机器学习技术得到了长足的发展,为设计模式识别规则的获取提供了一个新的途径.

2 设计模式自动识别的研究

下面将从基于图论的设计模式识别、基于形式化技术的设计模式识别、基于软件度量的设计模式识别和基于人工智能的设计模式识别四大类方法出发对近年来设计模式自动识别的研究进行综述(讨论的时间性能不考虑识别规则的获取过程,能否完全自动化不考虑源代码的动态执行过程).

2.1 基于图论的设计模式识别

UML模型和源代码缺乏精确的语义,使得利用计算机算法自动对设计模式进行识别变得困难. 将系统和设计模式转换为有向图的形式,是一种有效的解决方案. 近年来,国内外学者提出很多基于图论的设计模式自动识别方法.

该类方法的核心思想是:首先将系统和设计模式的类用有向图的顶点来表示,类之间的关系等静态特征用顶点之间的边来表示;然后使用某种图论算法比较系统和设计模式对应的有向图;最后,根据比较结果判断系统中是否存在模式实例. 流程如图1所示.

Yu 和 Bernardi 等[4-7]将系统和设计模式以有向图的形式呈现,通过图同构判定算法在系统图中寻找模式子图. 图同构算法具有较好的鲁棒性和很高的匹配精度,但是只能在系统图中找到与模板设计模式结构完全相同的模式实例.

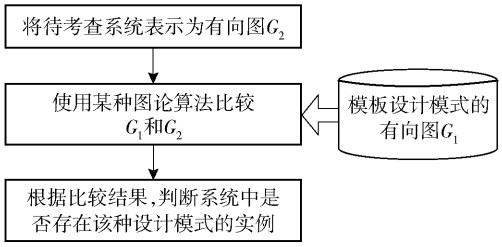

图1 基于图论的设计模式识别流程

Dong J 等[8]考虑了泛化、关联、抽象、不同的调用方式等8个设计模式的特征,将其表示为有向图的形式,得到了这些有向图的邻接矩阵,并将它们集成在一个矩阵中. 使用模板匹配算法计算系统和设计模式之间的互相关来寻找系统中存在的模式实例,该值越大,表示系统和模板越匹配. 与图同构判定算法不同,模板匹配可以在系统中找到与模板设计模式结构不完全相同但相似度较高的模式实例.

Tsantalis N 等[9]将源代码和设计模式的关联、泛化、抽象类、对象创建、抽象方法调用等信息均表示为一个单独的有向图/矩阵,并将系统分为若干子系统,然后使用相似度评分算法[10]计算各子系统与模板设计模式之间的相似度矩阵 $\boldsymbol{S}$. 根据相似度矩阵 $\boldsymbol{S}$ 来判断子系统是否为模式实例. 此外,还可以得到实例中的类和设计模式中角色的对应关系.

基于图论的方法主要适用于结构型和创建型模式的识别. 对于行为型模式,该类方法只能得到其候选实例. 此外,该类方法需要大量的矩阵运算,当系统的规模较大时,将会耗费较多的时间.

2.2 基于形式化技术的设计模式识别

有向图的语义较为简单,表达能力有限,有些特征并不容易表示为有向图的形式. 对于动态行为特征来说更是如此. 形式化技术基于严格定义的数学概念和语言,语义清晰,无歧义,可以对软件系统进行严格地检查和分析. 近年来,国内外很多学者将形式化技术及其支撑工具引入设计模式的识别中来.

该类方法的核心思想是借助形式化验证技术及其支撑工具,判断候选实例是否满足模板设计模式具有的属性,满足则说明该候选实例为模式实例;否则排除. 流程如图2所示.

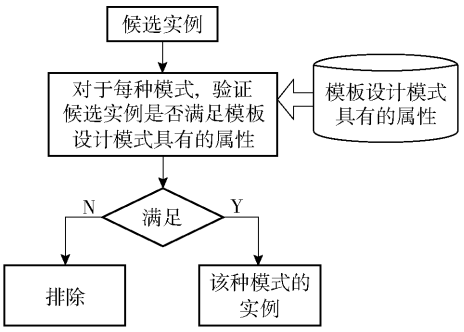

图 2　基于形式化技术的设计模式识别流程

模型检测提供了一个完整系统属性的验证框架，通过状态空间搜索的方法来检测一个给定的计算模型是否满足某个用时序逻辑公式表示的特定属性. Bernardi 等[11-13]将设计模式的行为属性表示为选择性 μ 演算公式，将文献[16-18]中所述的候选实例(Java 文件或字节码)转换为时序规范语言(LOTOS，language of temporal ordering specification)模型，并使用模型检测工具 CADP 验证候选实例是否满足选择性 μ 演算公式，满足则说明该实例为模式实例. Lucia A D 等方法[14]和 Bernardi M L 等方法类似，先使用静态结构分析得到初步的模式实例[15]，然后将设计模式的行为属性表示为线性时态逻辑公式，将候选实例转换为 Promela 模型，并使用模型检测工具 SPIN 验证候选实例是否满足线性时态逻辑公式.

有限自动机是表示有限状态及在这些状态之间的转移和动作等行为的数学模型. Wendehals 和 Orso[16]将行为型模式的行为特征(函数调用)转换为有限自动机的状态和它们之间的转移. 此外，引入一个没有传出转移(outgoing transition)的拒绝状态. 对于行为型模式候选实例，使用单元测试工具 JUnit 执行源代码并获取方法调用跟踪. 如果有限自动机在处理该跟踪时到达拒绝状态，则说明该子系统不是模式实例.

模型检测和有限自动机主要适用于形为型模式的搜索. 一阶逻辑(FOL, first order logic)是一种形式语言系统，具有精确、无二义性以及容易为计算机理解和操作等特点. Bayley I 等[17-18]用图形化扩展巴科斯范式(GEBNF, graphical extended backus normal form)来定义 UML 类图和序列图的语法，然后通过用 FOL 描述的设计模式结构特征和行为特征来增强这种语法正确性约束. 在此基础上，Zhu 等[19]给出一个称为 LAMBDES-DP 的设计模式识别工具. 该工具使用 LAMBDES 系统将待识别子系统的 UML 图转换为一组 FOL 语句，然后借助定理证明器 SPASS 验证该子系统是否符合模板设计模式. 若由一阶谓词表示的模式在子系统上的求值为真，则说明子系统符合该模式.

模型检测和有限自动机在系统规模较大时会产生状态空间爆炸，而使用 FOL 进行模式搜索不存在状态空间爆炸的问题. 但目前的定理证明器(SPASS、ACL2、Coq)还无法完全自动化，需要证明者和定理证明器进行交互.

2.3　基于软件度量的设计模式识别

软件度量标准用于衡量软件产品或软件开发过程的特定属性. 有学者尝试通过软件度量来识别设计模式.

该类方法的核心思想是：首先估算系统(源代码)程序相关的度量指标，如属性的个数、方法的个数、抽象方法的个数、类之间的关系；然后依次比较模板设计模式的各个指标度量值和系统的度量值，根据匹配结果判断系统是否为模型实例. 流程如图 3 所示.

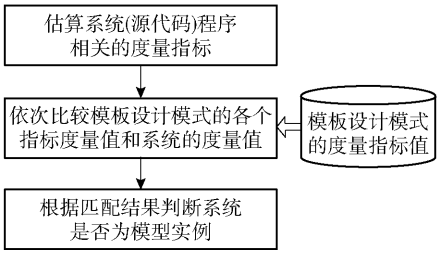

图 3　基于软件度量的设计模式识别流程

Antoniol G 等[20]将设计或源代码转换为抽象对象语言(AOL，abstract object language)代码，然后调用度量抽象器计算 AOL 代码中包含类声明的度量. 采用一个多阶段的约束评估器寻找设计模式实例，前面的阶段使用单个类的度量为后续阶段减少搜索空间，后续阶段应用结构型约束. Kim 等[21]将用于设计模式识别的度量指标细化为 3 类：1) 面向对象(OO，object-oriented)度量，包括继承树深度、子类个数、类之间关系等；2) 结构型度量，包括扇入、扇出、信息流度量等；3) 过程型度量，包括代码行数、圈复杂度、注释行数等. 他们开发了一个称为 PW (Pattern Wizard)的设计模式识别工具. 该工具调用度量发生器 C and C + + Code Counter (CCCC) 计算系统每个类的 3 种产品度量，并将这些度量与 PW 中嵌入的模式签名进行比较，根据匹配程度判断是否

为模式实例.

上述2种方法是独立的识别方法,Issaoui 等[22]试图通过在识别之前引入过滤阶段来优化任何一种设计模式识别方法,该阶段利用语义和结构设计度量来寻找设计模式实例的语义和结构征兆.

使用软件度量来识别设计模式只需要计算系统和模式的度量,并进行比较,时间性能高. 但是软件度量主要用于衡量系统的静态结构属性. 此外,使用软件度量来识别设计模式往往需要通过人工确认来获取最终的实例.

2.4 基于人工智能的设计模式自动识别

以上三大类方法都是使用预先定义的固定条件. 人工智能技术有极大的灵活性和对话能力,有自我解释能力和学习能力,对设计模式识别这种条件不明确的问题非常有效.

Prolog 是一种基于一阶谓词的逻辑程序设计语言,具有很强的逻辑描述能力和推理能力. Hayashi 等[23]将从待识别系统中抽取到的信息表示为 Prolog 中的事实,并将其注册到 Prolog 数据库中,然后执行定义为 Prolog 中规则的检测条件来推断满足设计模式条件类结构的存在. 这些规则用一组谓词指定设计模式应具有的属性,该方法最大的优势在于可以方便地添加和修改识别规则,但是 Prolog 的执行效率偏低,且执行过程无法完全自动化.

在上述方法中,识别规则都是从设计模式的理论描述中获取的. 近年来,有学者试图使用机器学习从实际软件系统中获取识别规则. 流程如图4所示.

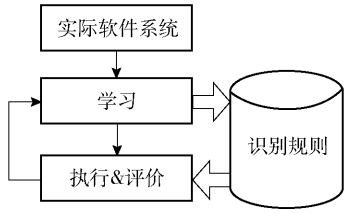

图4 使用机器学习获取识别规则流程

Ferenc R 等[24]手动将候选实例标记为真阳性或假阳性实例,并计算这些候选实例预测因子的值;然后将这些信息提供给学习系统进行学习. Uchiyama S 和 Kubo[25]将应用模式的程序输入度量测量系统中,以获得每个模式角色的度量值,并将度量值输入机器学习模拟器中进行学习. Alhusain S 等[26]使用多个设计模式识别工具对400多个开源应用程序进行分析,然后将工具认可/不认可的模式实例分别作为阳性/阴性训练样本来构建数据集. 针对每种设计模式和每种设计模式的每个角色,输入不同的特征向量来训练单独的人工神经网络(ANN, artificial neural network).

3 设计模式自动识别四大类方法的比较

笔者通过适用的模式类型、时间性能和能否完全自动化3个指标对设计模式自动识别四大类方法进行了比较,如表1所示.

表1 四大类方法的比较

识别方法类型		代表性文献	适用的模式类型	时间性能	完全自动化
基于图论的设计模式识别	图同构	[4]	结构型/创建型	低	是
	模板匹配	[8]			
	相似度评分	[9]			
基于形式化技术的设计模式识别	模型检测	[11-13]	行为型	低	是
	有限自动机	[16]	行为型		
	FOL	[19]	结构型/创建型/行为型	中	否
基于软件度量的设计模式识别	独立方法	[20]	结构型/创建型	高	否
	过滤方法	[22]	N/A*		N/A
基于人工智能的设计模式识别	Prolog	[23]	结构型/创建型/行为型	低	否
	机器学习	[24]	N/A	N/A	N/A

* N/A 表示该项指标对该类方法不适用.

4 问题与不足

1) 有很多学者考虑了设计模式变体的识别,如 Li 等[27]将设计模式的约束分为基本约束和变体约束,以此来提高变体识别的准确率. 然而设计模式的使用非常灵活,在实际的工程项目中往往会有

各种各样的模式变体，现有方法的识别准确率还不够高.

2）有一些学者考虑了时间性能的提升，如 Qiu 等[28]提出一种基于子图发现的新方法来一次识别一组设计模式；Gautam 等[29]应用稀疏矩阵算法进行有效的存储和计算，从而获得更好的性能和准确率；Issaoui 等[22]试图通过在识别之前引入过滤阶段来提高任何一种设计模式识别方法的时间性能；Hayashi 等[23]着眼于设计模式所具有的共同属性，并将其表示为 Pree 的元模式，以减少检测的时间成本和整个检测条件的大小；Alhusain 等[26]通过为每种设计模式的每个角色标识一组候选类来缩小搜索空间. 随着软件系统的规模增大，对设计模式识别的时间性能也提出了更高的要求.

3）使用机器学习获取设计模式识别规则最重要的过程是准备训练样本，而训练样本的获取与标记需要耗费很多时间和人力.

4）很多学者并未研发出与所提方法相对应的支撑工具，或者只给出工具的原型. 大多数方法和工具还无法应用于工程实践.

5　结束语

笔者认为今后可以从以下几个方向展开研究：

1）如何简便有效地获取和标记学习识别规则的训练样本；

2）设计模式变体的识别；

3）设计模式识别时间性能的提升；

4）研发设计模式识别工具.

参考文献：

[1] Rasool G, Streitfdert D. A survey on design pattern recovery techniques [J]. International Journal of Computer Science Issues, 2011, 8(6): 251-260.

[2] Gamma E, Helm R, Johnson R, et al. Design patterns: elements of reusable object-oriented software [M]. Boston: Addison-Wesley, 1995.

[3] Krämer C, Prechelt L. Design recovery by automated search for structural design patterns in object-oriented software [C] // Working Conference on Reverse Engineering. Monterey: IEEE, 1996: 1-9.

[4] Yu D, Zhang Y, Chen Z. A comprehensive approach to the recovery of design pattern instances based on sub-patterns and method signatures [J]. Journal of Systems & Software, 2015(103): 1-16.

[5] Bernardi M L, Lucca G A D. Model-driven detection of design patterns [C] // IEEE International Conference on Software Maintenance. Timisoara: IEEE Computer Society, 2010: 1-5.

[6] Bernardi M L, Cimitile M, Lucca G A D. A model-driven graph-matching approach for design pattern detection [C] // Working Conference on Reverse Engineering. Beverly: IEEE, 2013: 172-181.

[7] Bernardi M L, Cimitile M, Lucca G D. Design pattern detection using a DSL - driven graph matching approach [J]. Journal of Software Evolution & Process, 2014, 26(12): 1233-1266.

[8] Dong J, Sun Y, Zhao Y. Design pattern detection by template matching [C] // The 23rd Annual ACM Symposium on Applied Computing. Fortaleza: DBLP, 2008: 765-769.

[9] Tsantalis N, Chatzigeorgiou A, Stephanides G, et al. Design pattern detection using similarity scoring [J]. IEEE Transactions on Software Engineering, 2006, 32(11): 896-909.

[10] Blondel V D, Gajardo A, Heymans M, et al. A measure of similarity between graph vertices: applications to synonym extraction and web searching [J]. SIAM Review, 2004, 46(4): 647-666.

[11] Bernardi M L, Cimitile M, Ruvo G D, et al. Improving design patterns finder precision using a model checking approach [C] // The 27th International Conference on Advanced Information Systems Engineering. Stockholm: Springer-Verlag, 2015: 1-8.

[12] Bernardi M L, Cimitile M, Ruvo G D, et al. Integrating model driven and model checking to mine design patterns [M] // Berlin: Springer International Publishing, 2015.

[13] Bernardi M L, Cimitile M, Ruvo G D, et al. Model checking to improve precision of design pattern instances identification in OO systems [C] // International Joint Conference on Software Technologies. Lisbon: IEEE, 2016: 53-63.

[14] Lucia A D, Deufemia V, Gravino C, et al. Improving behavioral design pattern detection through model checking [C] // European Conference on Software Maintenance and Reengineering. Oldenburg: IEEE, 2011: 176-185.

[15] Lucia A D, Deufemia V, Gravino C, et al. Design pattern recovery through visual language parsing and source code analysis [J]. Journal of Systems & Software, 2009, 82(7): 1177-1193.

[16] Wendehals L, Orso A. Recognizing behavioral patterns

at runtime using finite automata [C]// Proceedings of the 2006 International Workshop on Dynamic Analysis. Shanghai: ACM, 2006: 33-40.

[17] Bayley I. Formalising design patterns in predicate logic [C]// IEEE International Conference on Software Engineering and Formal Methods. [S. l.]: IEEE, 2007: 25-36.

[18] Bayley I, Zhu H. Specifying behavioural features of design patterns in first order logic [C]// The 32nd Annual IEEE International Computer Software and Applications Conference. Turku: IEEE Computer Society, 2008: 203-210.

[19] Zhu H, Bayley I, Shan L, et al. Tool support for design pattern recognition at model level [C]// The 33rd Annual IEEE International Computer Software and Applications Conference. Seatle: IEEE, 2009: 228-233.

[20] Antoniol G, Fiutem R, Cristoforetti L. Using metrics to identify design patterns in object-oriented software [C]// International Symposium on Software Metrics. Bethesda: IEEE Computer Society, 1998: 23-34.

[21] Kim H, Boldyreff C. A method to recover design patterns using software product metrics [C]// International Conference on Software Reuse: Advances in Software Reusability. Vienna: Springer-Verlag, 2000: 318-335.

[22] Issaoui I, Bouassida N, Ben-Abdallah H. Using metric-based filtering to improve design pattern detection approaches [J]. Innovations in Systems & Software Engineering, 2015, 11(1): 39-53.

[23] Hayashi S, Katada J, Sakamoto R, et al. Design pattern detection by using meta patterns [J]. IEICE Transactions on Information & Systems, 2008, E91-D(4): 933-944.

[24] Ferenc R, Fülöp L, Lele J. Design pattern mining enhanced by machine learning [C]// The 21st IEEE International Conference on Software Maintenance. Budapest: IEEE, 2005: 295-304.

[25] Uchiyama S, Kubo A, Washizaki H, et al. Detecting design patterns in object-oriented program source code by using metrics and machine learning [J]. Journal of Software Engineering & Applications, 2014, 7(12): 1-12.

[26] Alhusain S, Coupland S, John R, et al. Towards machine learning based design pattern recognition [C]// The 13th Annual UK Workshop on Computational Intelligence. Surrey: IEEE, 2013: 244-251.

[27] Li W J, Pan J L, Wang K J. Research on detecting design pattern variants from source code based on constraints [J]. International Journal of Hybrid Information Technology, 2015, 8(5): 63-72.

[28] Qiu M, Jiang Q, Gao A, et al. Detecting design pattern using subgraph discovery [C]// The 2nd Asian Conference on Intelligent Information and Database System. Hue City: DBLP, 2010: 350-359.

[29] Gautam A K, Diwaker S, Gautam A K, et al. Automatic detection of software design patterns from reverse engineering [C]// The 2nd International Conference on Issues and Challenges in Networking, Intelligence and Computing Technologies. Ghaziabad: [s. n.], 2012: 17-22.