Методы восстановления и визуализации архитектуры программной системы

В.Ю. Романов

***Аннотация* — В статье делается обзор методов восстановления и визуализации архитектуры программных систем. Понимание архитектуры жизненно важно для эффективного сопровождения и управление большими программными системами. Однако, по мере развития программных систем с течением времени их архитектура неизбежно меняется. Архитекторам необходимо отслеживать изменения на уровне реализации и соответствующим образом обновлять документацию с описанием архитектуры, что занимает много времени и и зачастую приводит к ошибкам в описании. Для облегчения этого процесса было предложено множество методов автоматического восстановления архитектуры. Несмотря на усилия по повышению точности восстановления архитектуры, существующие решения по-прежнему страдают от двух ограничений. Во-первых, большинство из них используют для восстановления только один или два типа информации, игнорируя потенциальную полезность других источников. В данном обзоре рассмотрены и многокритериальные методы. Во-вторых, они склонны использовать информацию грубо, упуская из виду важные детали. Для выявления таких деталей рассмотрены методы визуализации**

***Ключевые слова* — Восстановление архитектуры; Визуализация архитектуры; Эволюция программного обеспечения;**

I. Введение

Архитектура программного обеспечения — это абстракция программной системы, состоящая из набора архитектурных элементов, связей между этими элементами, а также свойств, как элементов, так и их отношений. Модули представляют собой единицы реализации и часто связаны с исходным кодом, тогда как компоненты представляют собой объекты времени выполнения [1]. Восстановление архитектуры программного обеспечения — это деятельность по восстановлению тех архитектурных элементов вместе с их свойствами и связями, которые составляют все программное обеспечение.

Ряд методов восстановления архитектуры основан на взвешенных метриках программной системы. Методы на основе весов рассчитывают набор метрик между обнаруженными модулями и применяют кластеризацию на основе таких метрик [2]. Большинство предложенных методов используют алгоритмы иерархической кластеризации для восстановления архитектуры данного программного обеспечения. В работе [3] предложен алгоритм нечеткой иерархической кластеризации. Этот алгоритм сначала извлекает информацию, которая определяет систему кластеризации, затем присваивает веса извлеченной информации для создания кластера, повышая его точность. Хотя кластеризация является одним из наиболее важных методов, используемых для восстановления архитектуры программного обеспечения, она по-прежнему требует взаимодействия человека для уточнения результатов кластеризации. В работе [4] предложен алгоритм кластеризации на основе зависимостей, позволяющий удалять модули, требующие взаимодействия с человеком.

Метод кластеризации, позволяющий группировать похожие объекты с использованием матрицы сходства, был предложен в работе [5]. Целью метода было не только восстановить архитектуру существующего программного обеспечения, но и предложить разработчикам программного обеспечения улучшить архитектуру программного обеспечения путем принятия шаблона проектирования на основе результатов кластеризации.

Многокритериальный метод [6] был представлен для улучшения структуры программного обеспечения. С использование этого метода строится граф, в котором программные модули действуют как узлы, а зависимости модулей — как ребра. После этого алгоритм начинает рассчитывать индексы пакетов, такие как индекс сцепления пакета, индекс связности пакета, плотность соединений внутри пакета и индекс размера пакета. После расчета этих показателей такие значения передаются в функцию пригодности, чтобы начать генерировать рекомендации по перемещению классов между пакетами для улучшения связи и уменьшения общих зависимостей между различными пакетами.

В работе [7] был представлен облегченный подход к восстановлению архитектуры программного обеспечения, состоящий из трех отдельных этапов. На первом этапе этого подхода восстанавливаются изолированные элементы и итеративно восстанавливаются только те фрагменты архитектуры программного обеспечения, на которые влияла эволюция системы. На втором этапе восстанавливаются компоненты архитектуры программного обеспечения, и извлекается информация, связанная с соединителями

Статья получена 25 января 2024 г.
В.Ю. Романов - МГУ имени М.В. Ломоносова (e-mail: vladimir.romanov@gmail.com).

элементов, используемыми в программной системе. Наконец, не только восстанавливается архитектура программной системы, но и выполняется ее реконструкция.

Новая тенденция методов восстановления архитектуры - использование нескольких алгоритмов для повышения качества кластеризации. В [8] гибридный метод использует четыре алгоритма кластеризации: Bunch [9], взвешенный комбинированный алгоритм (WCA) [9], K-средние [10] и алгоритм Блонделя [11]. Эти алгоритмы используются на этапе генерации для создания базовых кластеров путем выполнения нескольких методов кластеризации (каждый из которых генерирует базовый кластер) или многократного выполнения одного метода кластеризации с разными настройками параметров. После этого, на этапе консенсуса, лучшие базовые кластеры объединяются для создания окончательного набора кластеров.

II. Методы кластеризации для восстановления архитектуры программных систем

Общепринятой практикой при разработке программного обеспечения является использование свободно распространяемых библиотек в исходных кодах, хранимых в программных репозиториях. Зачастую такие библиотеки представляют собой достаточно большие постоянно развивающиеся программные системы со сложной структурой. По этой причине использование стороннего программного кода осложняет необходимость понимания архитектуры программных систем. Отсутствие описания архитектуры или его устаревание вследствие постоянного развития библиотек делает актуальным задачу автоматического восстановления архитектуры из ее исходных текстов на некотором языке программирования.

Широкое распространение получили подходы кластеризации основанной на поиске и иерархической кластеризации. Подход кластеризации основанной на поиске [12], [13], [14], [15], [16], [17], используется для восстановления архитектуры и рефакторинга программных систем. В общем представлении алгоритмы кластеризации на основе поиска состоят из следующих шагов: Создается случайное начальное решение на основе некоторых начальных параметров, Затем исследуется структура окрестностей такого решения. Если соседнее решение лучше, он становится новым решением, исследование повторяется до тех пор, пока окрестность текущего решения-кандидата не будет предлагать дальнейшее улучшение целевой функции (локальный оптимум), Затем процесс повторяется с другим начальным числом, чтобы найти лучшее решение на основе целевой функции (глобальный оптимум). При иерархической кластеризации меньшие кластеры итеративно объединяют в более крупные кластеры или делят большие кластеры на более мелкие, в зависимости от того, является ли это подходом «снизу вверх» или «сверху вниз». Операции слияния или разделения обычно зависят от алгоритма кластеризации, используемого в существующих исследованиях. В целом алгоритмы иерархической кластеризации можно разделить на два основных подхода: разделительные (сверху вниз) и агломеративные (снизу вверх) иерархические алгоритмы кластеризации.

Понимание архитектуры программной системы только лишь путем анализа структуры системы раскрывает только часть картины, поскольку структура системы говорит нам только о том, как работает код, но не о том, о чем этот код. О чем код, можно узнать по семантике исходного кода. В частности, по используемым в системе идентификаторам. При этом находятся те кластеры, в которых используются похожие термины, и для вычисленных кластеров выявляются наиболее релевантные термины. В ряде работ [18], [19], [20] описываются алгоритмы такой группировки артефактов системы, получившие название семантической кластеризации.

III. Методы визуализации для восстановления архитектуры программных систем

Понимание большого и сложного программного обеспечения является нетривиальной задачей. В работах [21], [22], [23] используется один из методов визуализации для анализа результатов работы алгоритмов кластеризации с помощью карт распределения. Для этого используется общий метод, позволяющий оценить результат анализа программного обеспечения. Понимание того, как данное свойство артефактов программной системы распределяется в большой программной системе, является ключевой информацией для общего понимания системы. В используемом методе показывается, как связывается эталонный раздел системы со сравнительным разделом. Обычно эталонный раздел - это внутренняя структура системы (например, структура пакета классов), а раздел для сравнения задается группировкой или свойством артефактов системы (например, классы, сгруппированные по их авторам). В дополнение к визуализации кластеризации предложены две метрики для количественной оценки разброса (spread) и фокуса (focus) свойства артефакта системы по отношению к разделу системы. Метрика фокуса показывает, является ли свойство хорошо инкапсулированным, и метрики разброса, которая показывает, присутствует ли свойство в нескольких частях системы.

На рисунке 1 показано, как извлеченные концепции предметной области распределяются по структуре пакета: части — это пакеты, элементы — это классы, а цвета относятся к их концепциям.

Рис 1. Карта распределения лингвистических понятий по пакетам системы JEdit.

На рисунке 2 для каждой концепции указаны ее размер, распространение, направленность и описание концепции.

color	size	spread	focus	concept
red	116	25	.605	(main domain concept)
blue	80	4	.883	BeanShell scripting
cyan	68	8	.712	regular expressions
green	63	13	.475	user interface
pink	26	7	.335	text buffers
dark-green	12	3	.456	TAR and ZIP archives
yellow	10	4	.105	dockable windows
magenta	10	3	.484	XML reader
orange	9	1	.900	bytecode assembler

Рис 2. Свойства концепций показанных на рисунке 1.

Из просмотра значений метрик на рисунке 2 видно три наиболее хорошо инкапсулированные концепции, то есть кластеры с наибольшим фокусом (оранжевый, синий и голубой), реализуют четко разделенные концепции, такие как скрипты и регулярные выражения. Концепции с наименьшим фокусом пронизывают всю систему: желтый цвет - показывает закрепляемые окна, пользовательскую функцию графического интерфейса, а розовый касается обработки текстовых буферов. Эти две концепции являются хорошими кандидатами для более тщательного изучения, поскольку возможно их следует реорганизовать в отдельные пакеты.

На рисунке 1 распределение красного цвета, крупнейшего кластера и, следовательно, основной концепции домена приложения, показывает, какие части системы принадлежат ядру, а какие нет. Основываясь на порядке пакетов, мы можем сделать вывод, что две концепции пользовательского интерфейса, зеленый и желтый, более тесно связаны с ядром, чем, например, концепция голубого цвета, которая реализует регулярные выражения.

Совместная кластеризация изменений в модулях программной системы - это метод оценки модульности в архитектуре системы, который показывает, как часто изменения локализуются в модулях, и представляет ли распространение изменений проблемы проектирования. Этот метод основан на кластерах совместного изменения, которые представляют собой сильно взаимосвязанные файлы исходного кода, учитывающие отношения совместного изменения. В работе [22] проведена серия эмпирических анализов большого корпуса популярных программных проектов на GitHub. Используя определенные в [21] метрики фокуса и разброса, а также заданные для них пороговые значения метрик, были обнаружены и описаны шесть паттернов совместного изменения модулей: Encapsulated, Wrapped (Well-Confined), Crosscutting, Octopus, Black Sheep и Squid.

Было выяснено, что кластеры Encapsulated (инкапсулированный) и Wrapped (упакованный) реализуют четко определенные и ограниченные задачи. Примеры таких кластеров для систем Linux и Intellij-Community показаны на рисунках 3 и 4.

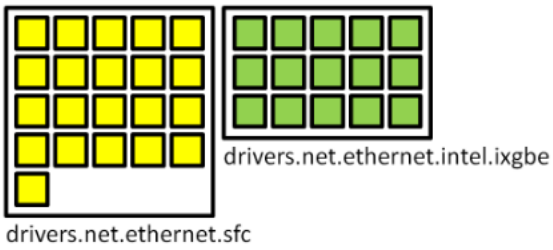

Рис 3. Инкапсулированные кластеры (Linux)

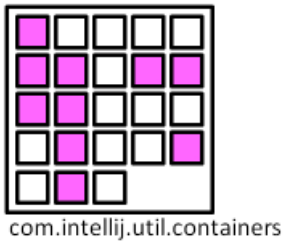

Рис 4. Упакованный кластер (Intellij-Community)

Кластеры Octopus (осьминог) пропорционально многочисленны относительно других паттернов. Они в значительной степени связаны с волновым эффектом, активностью, заинтересованностью и разнообразием в командах разработчиков. Пример кластеров Octopus для системы Python показан на рисунке 5.

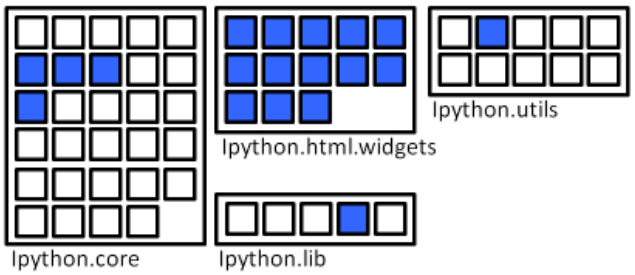

Рис 5. Кластер Octopus (Python)

Хотя кластер Crosscutting (сквозной) разбросан по каталогам, они реализуют четко определенные задачи. Несмотря на то, что они демонстрируют более высокую активность по сравнению с кластерами Wrapped, от них не всегда легко избавиться, что позволяет предположить, что инструменты поддержки могут сыграть решающую роль. Пример кластера Crosscutting показан на рисунке 6.

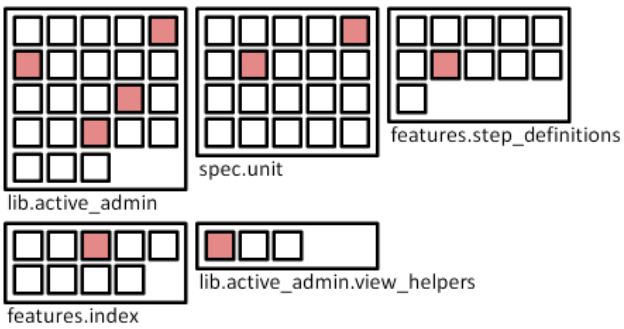

Рис 6. Crosscutting кластер (Active Admin)

IV. Исследование методов и алгоритмов выявления и визуализации шаблонов и дефектов проектирования на основе модели программной системы

Исследование методов и алгоритмов выявления и визуализации шаблонов и дефектов проектирования на основе модели программной системы

Шаблоны проектирования [24] широко применяются в разработке программного обеспечения. Шаблоны представляют собой хорошо зарекомендовавшее решение проблем возникающих при проектировании программной системы. Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код. Это лишь пример решения задачи, который можно использовать в различных ситуациях. Под антипаттернами [25] понимаются шаблоны проектирования, целью которых является документирование распространенной плохой практики в проектировании программного обеспечения. Такие примеры недостатков проектирования называются также ошибками проектирования и дефектами проектирования.

Обзоры методов обнаружения дефектов проектирования сделаны в [26], [27]. В работах [28], [29] предложены подходы к спецификации объектно-ориентированных дефектов проектирования на языке моделирования UML [30]. Благодаря такой спецификации появляется возможность автоматизировать обнаружение и исправление дефектов проектирования за счет использования подхода независимого от языка программирования.

В работах [31], [32] дефекты проектирования описываются с помощью правил. Для описания дефекта проектирования определяются набор классов и отношений между ними. Описанный таким образом дефект затем используется для поиска в UML-модели схожих конфигураций классов и отношений между ними. В работе [33] предложена графическая нотация, которая может быть использована как для спецификации дефектов проектирования с помощью правил, так и для визуализации дефектов найденных в UML-модели программной системы

В [34], [35], [36] предложен подход, основанный на метриках, для выявления дефектов проектирования в 30-моделях программных систем с использованием существующих и ряда новых определенных метрик оценки качества проектирования. Предлагаемые подходы исследует информацию о структуре и поведении описанную с помощью UML-диаграмм классов и последовательностей.

Библиография

[1] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, Judith Stafford. Documenting Software Architectures: Views and Beyond 2nd Edition, 2010, Addison-Wesley Professional, ISBN 0321552687

[2] Lutellier, Thibaud et al., 2015. Comparing software architecture recovery techniques using accurate dependencies. 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 2. IEEE, pp. 69–78.

[3] Y. Wang, P. Liu, H. Guo, H. Li, and X. Chen, “Improved hierarchical clustering algorithm for software architecture recovery,” in Proceedings - 2010 International Conference on Intelligent Computing and Cognitive Informatics, ICICCI 2010, pp. 247–250

[4] K. Kobayashi, M. Kamimura, K. Kato, K. Yano, and A. Matsuo, “Feature-gathering dependency-based software clustering using Dedication and Modularity,” in IEEE International Conference on Software Maintenance, ICSM, 2012, pp.462–471.

[5] J. Dong, Y. Zhao, and Y. Sun, “A matrix-based approach to recovering design patterns,” IEEE Trans. Syst. Man, Cybern. Part ASystems Humans, 2009, vol. 39, no. 6, pp. 1271–1282.

[6] Chhabra, Jitender Kumar et al., 2017. Improving package structure of object oriented software using multi-objective optimization and weighted class connections. J. King Saud Univ.-Comput. Infr. Sci. 29 (3), 349–364.

[7] J. Garcia, I. Krka, N. Medvidovic, and C. Douglas, “A framework for obtaining the ground-truth in architectural recovery,” in Proceedings of the Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture, WICSA/ECSA 2012, pp. 292–296.

[8] Cho, Choongki et al., 2019. Software architecture module-view recovery using cluster ensembles. IEEE Access 7, 72872–72884.

[9] Maqbool, Onaiza, Babri, Haroon Atique, 2004. The weighted combined algorithm: A linkage algorithm for software clustering. In: Eighth European Conference on Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. IEEE, pp. 15–24.

[10] Gupta, Tanvi, Panda, Supriya P., 2019. Clustering validation of CLARA and k-means using silhouette & DUNN measures on Iris dataset. In: 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon). IEEE, pp. 10–13.

[11] Blondel, Vincent D. et al., 2008. Fast unfolding of communities in large networks. J. Stat. Mech.: Theory Exp. 2008 (10), P10008.

[12] F. Beck and S. Diehl, On the impact of software evolution on software clustering, Empirical Software Engineering, vol. 18, no. 5, pp.970–1004, 2013.

[13] M. Harman, S. Swift, and K. Mahdavi, An empirical study of the robustness of two module clustering fitness functions,” in Proceedings of the 7th annual conference on Genetic and evolutionary computation. ACM, 2005, pp. 1029–1036.

[14] B. Mitchell and S. Mancoridis, On the automatic modularization of software systems using the bunch tool, IEEE Transactions on Software Engineering, vol. 32, no. 3, pp. 193–208, March 2006

[15] K. Praditwong, M. Harman, and Y. Xin, Software module clustering as a multi-objective search problem, IEEE Transactions on Software Engineering, vol. 37, no. 2, pp. 264–282, 2011.

[16] N. Teymourian, H. Izadkhah, and A. Isazadeh, A fast clustering algorithm for modularization of large-scale software systems, IEEE Transactions on Software Engineering, 2020.

[17] Alvin Jian Jia Tana, Chun Yong Chonga, Aldeida Aletib Explaining software clustering for remodularisation. J. Syst. Software 186, 111162 (2022).

[18] A. Kuhn, S. Ducasse, and T. Girba, Enriching Reverse Engineering with Semantic Clustering Proc. of 12th Working Conference on Reverse Engineering (WCRE 2005), Pittsburgh, Pennsylvania, USA, pp. 133-142, 2005.

[19] Kuhn A, Ducasse S, Girba T Semantic clustering: identifying topics in source code. Inf Softw Technol 49(3):230–243, 2007

[20] Gustavo Santos, Marco Tulio Valente, Nicolas Anquetil. Remodularization Analysis Using Semantic Clustering. 1st CSMR-WCRE Software Evolution Week, Feb 2014, Antwerp, Belgium.

[21] Ducasse,S., Gîrba,T., and Kuhn,A. Distribution map. In 22nd International Conference on Software Maintenance, 2006, pages 203–212.

[22] Luciana L Silva, Marco Tulio Valente, and Marcelo A Maia. Co-change patterns: A large scale empirical study. Journal of Systems and Software 152 (2019), 196–214

[23] N.J. Agouf, S.Ducasse, A.Etien, A.Alidra, A.Thiefaine Understanding Class Name Regularity: A Simple Heuristic and Supportive Visualization. The Journal of Object Technology, 2022

[24] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. (1994) ISBN 978-0-201-63361-0.

[25] W. Brown, R. Malveau, H. McCormick, and T. Mowbray, AntiPatterns: Refactoring software, architectures, and projects in crisis. John Wiley & Sons, 1998.

[26] J. Din, A.B. Al-Badareen, Y. Yah Jusoh Antipatterns Detection Approaches in Object-Oriented Design: A literature review Computing and

Convergence Technology (ICCCT), 2012 7th International Conference on. P. 926–931.
[27] F. Palomba, G. Bavota, R. Oliveto, A. De Lucia
Anti-pattern Detection: Methods, Challenges, and Open Issues
Advances in Computers. 2014. P. 201–238.
[28] M. Llano and R. Pooley. UML specification and correction of object oriented anti-patterns, in Fourth International Conference on Software Engineering Advances, 2009. ICSEA '09. IEEE, 2009, pp. 39–44.
[29] Xue Qing, Study on the detection and correction of software based on UML in 2010 International Conference on E-Health Networking, Digital Ecosystems and Technologies (EDT), vol. 1. IEEE, 2010, pp.268–271.
[30] OMG Unified Modeling Language TM (OMG UML) Version 2.5 http://www.omg.org/spec/UML/2.5/PDF/
[31] Ballis, Demis; Baruzzo, Andrea; Comini, Marco.
A Rule-based Method to Match Software Patterns Against UML Models.
In: Electr. Notes Theor. Comput. Sci. 219 (2008), p. 51–66
[32] Nacha Chondamrongkul, Jing Sun, Ian Warren, Scott Uk-Jin Lee Semantic-based Architecture Smell Analysis
Proceedings of the 8th International Conference on Formal Methods in Software EngineeringOctober 2020, pages 109–118
[33] Ballis D., Baruzzo A., Comini M. A minimalist visual notation for design patterns and antipatterns Fifth International Conference on Information Technology. New Generations (itng 2008). IEEE, 2008. p.51-56.
[34] Elsayed, E.K. and El-Sharawy, E.E., Detecting Design Level Antipatterns; Structure and Semantics in UML Class Diagrams. Journal of Computers, 13(6), pp.638-655. 2018
[35] R Fourati, N Bouassida, HB Abdallah A metric-based approach for anti-pattern detection in UML designs Computer and Information Science 2011, 2011
[36] M. Salehie, S. Li, and L. Tahvildari, A metric-based heuristic framework to detect object-oriented design flaws, in 14th IEEE International Conference on Program Comprehension (ICPC'06), 2006, pp. 159–168.

Methods for restoring and visualizing the architecture of a software system

V.Yu. Romanov

Abstract — ***The article provides an overview of methods for restoring and visualizing the architecture of software systems. Understanding architecture is vital to effectively maintaining and managing large software systems. However, as software systems evolve over time, their architecture inevitably changes. Architects need to track changes at the implementation level and update the architecture documentation accordingly, which is time-consuming and often leads to errors in the description. To facilitate this process, many automatic architectural recovery techniques have been proposed. Despite efforts to improve the accuracy of architectural reconstruction, existing solutions still suffer from two limitations. First, most of them use only one or two types of information for recovery, ignoring the potential usefulness of other sources. This review also discusses multicriteria methods. Secondly, they tend to use information roughly, leaving out important details. To identify such details, methods for visualizing the architecture of a software system are considered.***

Keywords - ***Architectural restoration; Architecture visualization; Software evolution*;**

REFERENCES

[1] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, Judith Stafford. Documenting Software Architectures: Views and Beyond 2nd Edition, 2010, Addison-Wesley Professional, ISBN 0321552687

[2] Lutellier, Thibaud et al., 2015. Comparing software architecture recovery techniques using accurate dependencies. 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 2. IEEE, pp. 69–78.

[3] Y. Wang, P. Liu, H. Guo, H. Li, and X. Chen, "Improved hierarchical clustering algorithm for software architecture recovery," in Proceedings - 2010 International Conference on Intelligent Computing and Cognitive Informatics, ICICCI 2010, pp. 247–250

[4] K. Kobayashi, M. Kamimura, K. Kato, K. Yano, and A. Matsuo, "Feature-gathering dependency-based software clustering using Dedication and Modularity," in IEEE International Conference on Software Maintenance, ICSM, 2012, pp.462–471.

[5] J. Dong, Y. Zhao, and Y. Sun, "A matrix-based approach to recovering design patterns," IEEE Trans. Syst. Man, Cybern. Part ASystems Humans, 2009, vol. 39, no. 6, pp. 1271–1282.

[6] Chhabra, Jitender Kumar et al., 2017. Improving package structure of object oriented software using multi-objective optimization and weighted class connections. J. King Saud Univ.-Comput. Infr. Sci. 29 (3), 349–364.

[7] J. Garcia, I. Krka, N. Medvidovic, and C. Douglas, "A framework for obtaining the ground-truth in architectural recovery," in Proceedings of the Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture, WICSA/ECSA 2012, pp. 292–296.

[8] Cho, Choongki et al., 2019. Software architecture module-view recovery using cluster ensembles. IEEE Access 7, 72872–72884.

[9] Maqbool, Onaiza, Babri, Haroon Atique, 2004. The weighted combined algorithm: A linkage algorithm for software clustering. In: Eighth European Conference on Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. IEEE, pp. 15–24.

[10] Gupta, Tanvi, Panda, Supriya P., 2019. Clustering validation of CLARA and k-means using silhouette & DUNN measures on Iris dataset. In: 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon). IEEE, pp. 10–13.

[11] Blondel, Vincent D. et al., 2008. Fast unfolding of communities in large networks. J. Stat. Mech.: Theory Exp. 2008 (10), P10008.

[12] F. Beck and S. Diehl, On the impact of software evolution on software clustering, Empirical Software Engineering, vol. 18, no. 5, pp.970–1004, 2013.

[13] M. Harman, S. Swift, and K. Mahdavi, An empirical study of the robustness of two module clustering fitness functions," in Proceedings of the 7th annual conference on Genetic and evolutionary computation.
ACM, 2005, pp. 1029–1036.

[14] B. Mitchell and S. Mancoridis, On the automatic modularization of software systems using the bunch tool, IEEE Transactions on Software Engineering, vol. 32, no. 3, pp. 193–208, March 2006

[15] K. Praditwong, M. Harman, and Y. Xin, Software module clustering as a multi-objective search problem, IEEE Transactions on Software Engineering, vol. 37, no. 2, pp. 264–282, 2011.

[16] N. Teymourian, H. Izadkhah, and A. Isazadeh,
A fast clustering algorithm for modularization of large-scale software systems, IEEE Transactions on Software Engineering, 2020.

[17] Alvin Jian Jia Tana, Chun Yong Chonga, Aldeida Aletib
Explaining software clustering for remodularisation.
J. Syst. Software 186, 111162 (2022).

[18] A. Kuhn, S. Ducasse, and T. Girba, Enriching Reverse Engineering with Semantic Clustering Proc. of 12th Working Conference on Reverse Engineering (WCRE 2005), Pittsburgh, Pennsylvania, USA, pp. 133-142, 2005.

[19] Kuhn A, Ducasse S, Girba T Semantic clustering: identifying topics in source code. Inf Softw Technol 49(3):230–243, 2007

[20] Gustavo Santos, Marco Tulio Valente, Nicolas Anquetil.
Remodularization Analysis Using Semantic Clustering. 1st CSMR-WCRE Software Evolution Week, Feb 2014, Antwerp, Belgium.
[21] Ducasse,S., Gîrba,T., and Kuhn,A. Distribution map. In 22nd International Conference on Software Maintenance, 2006, pages 203–212.
[22] Luciana L Silva, Marco Tulio Valente, and Marcelo A Maia.
Co-change patterns: A large scale empirical study. Journal of Systems and Software 152 (2019), 196–214
[23] N.J. Agouf, S.Ducasse, A.Etien, A.Alidra, A.Thiefaine
Understanding Class Name Regularity: A Simple Heuristic and Supportive Visualization. The Journal of Object Technology, 2022
[24] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John. Design Patterns: Elements of Reusable Object-Oriented Software.
Addison-Wesley. (1994) ISBN 978-0-201-63361-0.
[25] W. Brown, R. Malveau, H. McCormick, and T. Mowbray,
AntiPatterns: Refactoring software, architectures, and projects in crisis.
John Wiley & Sons, 1998.
[26] J. Din, A.B. Al-Badareen, Y. Yah Jusoh Antipatterns Detection Approaches in Object-Oriented Design: A literature review Computing and Convergence Technology (ICCCT), 2012 7th International Conference on. P. 926–931.
[27] F. Palomba, G. Bavota, R. Oliveto, A. De Lucia
Anti-pattern Detection: Methods, Challenges, and Open Issues
Advances in Computers. 2014. P. 201–238.
[28] M. Llano and R. Pooley. UML specification and correction of object oriented anti-patterns, in Fourth International Conference on Software Engineering Advances, 2009. ICSEA ’09. IEEE, 2009, pp. 39–44.
[29] Xue Qing, Study on the detection and correction of software based on UML in 2010 International Conference on E-Health Networking, Digital Ecosystems and Technologies (EDT), vol. 1. IEEE, 2010, pp.268–271.
[30] OMG Unified Modeling Language TM (OMG UML) Version 2.5 http://www.omg.org/spec/UML/2.5/PDF/
[31] Ballis, Demis; Baruzzo, Andrea; Comini, Marco.
A Rule-based Method to Match Software Patterns Against UML Models.
In: Electr. Notes Theor. Comput. Sci. 219 (2008), p. 51–66
[32] Nacha Chondamrongkul, Jing Sun, Ian Warren, Scott Uk-Jin Lee Semantic-based Architecture Smell Analysis
Proceedings of the 8th International Conference on Formal Methods
in Software EngineeringOctober 2020, pages 109–118
[33] Ballis D., Baruzzo A., Comini M. A minimalist visual notation for design patterns and antipatterns Fifth International Conference on Information Technology. New Generations (itng 2008). IEEE, 2008. p.51-56.
[34] Elsayed, E.K. and El-Sharawy, E.E., Detecting Design Level Antipatterns; Structure and Semantics in UML Class Diagrams. Journal of Computers, 13(6), pp.638-655. 2018
[35] R Fourati, N Bouassida, HB Abdallah A metric-based approach for anti-pattern detection in UML designs Computer and Information Science 2011, 2011
[36] M. Salehie, S. Li, and L. Tahvildari, A metric-based heuristic framework to detect object-oriented design flaws, in 14th IEEE International Conference on Program Comprehension (ICPC’06), 2006, pp. 159–168.