ISSN 2278-3091
Volume 8, No.1.4, 2019
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse6781.42019.pdf
https://doi.org/10.30534/ijatcse/2019/6781.42019

A Comparison of Functionality-Based Packaging Using GA and Adaptive KNN Clustering as Two Approaches to Package Software

Shouki A. Ebad[1], Moataz Ahmed[2]
[1]Computer Science Department, Faculty of Science, Northern Border University, Saudi Arabia
shouki.abbad@nbu.edu.sa
[2]Department of Info. & Comp. Science, King Fahd University of Petroleum & Minerals, Saudi Arabia
moataz@kfupm.edu.sa

ABSTRACT

In large scale object-oriented (OO) software, package forms the essential structural component of the system. Packaging operation aims at grouping OO classes to provide well-identified functions to the rest of the software. It is meant to improve the software quality such as high maintainability and high reusability. Most of the packaging approaches are based on an optimal balance between cohesion and coupling quality attributes. In this article, we compare of two packaging approaches: functionality-based packaging relied on genetic algorithm, GA (measured with a metric named OverallPackaging) and adaptive k-nearest neighbor (A-KNN) clustering. The comparison was done in terms of cohesion and coupling at package level. Both approaches are applied on an open-source system called Trama. In term of cohesion and coupling at the package level, both approaches achieved good results compared with the original packaging of Trama. However, the A-KNN-based approach achieved better results than OverallPackaging.

Key words: search-based software engineering (SBSE), software packaging, package cohesion, package coupling.

1. INTRODUCTION

Software packaging is grouping object-oriented (OO) classes into packages so that a package does one task, which is completely carried out in the package [[1]]. Packaging is interchangeably referred to as clustering [[2]]-[[4]], modularization [5]]-[[7]], and decomposition [8]]. Because this process is done by changing the software structure and architecture without affecting its internal behavior, packaging is mainly meant to achieve quality objectives such as high maintainability and high reusability [[1]]-[[4]]. Software architecture has become a more mature area with regards to applying empirical research [[9]]. Packaging is often employed during the architectural design stage of the software development lifecycle [[1]]. Some employed the same process on the source code level; i.e., after the source code is available; it is called refactoring [10]]-[[12]].

Although the effort spent on packaging is worthwhile since it, in general, improves software quality, each packaging approach has a different impact on software quality [[11]]. Therefore, in order to optimize the software structure for maintainability, for instance, architects should strive to build well-structured software. Finding such a number of solutions exhaustively is impracticable because of the combinatorial nature of the packaging. We thus approximated the number of optimal packagings using a search mechanism that minimizes the solution space by choosing a subset of solutions that could guide to a good solution. However, finding the best packaging of a system might not be cheap to be done manually [[13]]. For this, packaging is often treated as an optimization problem where the fitness function that leads the search using considered heuristic techniques is equivalent to a relevant software metric [1]], [[10]][14]]. Different automatic approaches aimed at performing software packaging have been proposed; a critical analysis of the existing approaches is in [[15]].

The objective of this paper is then to compare two approaches used recently to package OO classes. Our comparison would be in terms the cohesion and coupling amount at package level. Herein, we are considering package cohesion to be the connection amount in packages and package coupling to be the connection amount out packages. We applied functionality-based packaging and A-KNN clustering on an open source project, Trama, to investigate their impact on the cohesion and coupling at package level. Therefore, the main contribution of our work here is in this direction. The study is organized as follows. In Section 2, we review the literature of packaging approaches. Section 3 gives an overview of the two packaging approaches used in our research. In Section 4, we explain the process to extract some of the structure artifacts required for one of the two approaches. In Section 5, we present a comparative experiment; this includes the material, heuristic algorithms, tools and the analysis of results. Section 6 discusses some limitations of the study. In Section 7, we finally present the conclusions and future work.

2. RELATED WORK

Using GGA, [[8]] proposed a decomposition approach with a multi-modal function that included some attributes including cohesion and coupling [[5]] defined the fitness function using several measures inspired by the concept of package coupling and package cohesion [[16]]. Their approach allows maintainers to define certain constraints on the suggested packaging. [[12]] packaged classes using the similarity measure of a previous study [[17]] with fixed number and

variable number of packages. [[18]] worked on level of class, attribute, method/parameter names, comments, and statements. They grouped source files based on the lexical information using a hierarchical clustering algorithm. [[19]] automated the architecture recovery process of systems. They used (1) latent semantic indexing (LSI) to get similarities among software parts, (2) the k-means clustering algorithm to create groups of entities, and (3) fold-in and fold-out mechanisms to improve computational time. [[20]] focused on a specific restructuring: given a package with less cohesion, partition it into smaller packages having higher cohesion. To measure package cohesion, they used conceptual coupling between classes (CCBC) [[22]] and information-flow-based coupling (ICP) [[22]] to capture semantic relationships and structure relationships between classes, respectively. They then used a metric to find classes that should put in a package. [[7]] used NSGA-II to implement a remodularization approach to maximize package cohesion, minimize package coupling, minimize package cycles, avoid Blob packages, and minimize the changes of the original design. [[1]] proposed a new packaging approach based on UCs, which in turn are realized by sequence diagrams (SDs). This approach came to reflect the functionality perspective of systems. [13]] presented the results of objectively and subjectively studies about modularization based on package cohesion and coupling. As results (1) most of the analyzed systems exhibit low values for cohesion and coupling at package level (2) most of the developers confirmed that other attributes guided the modularization of their systems.

Our literature survey reveals that the available works do not consider the package level in their modularization/ packaging but class level [5]], [[8]] or method and attribute level [[18]]. Others focused on architecture recovery [[19]]. Although few works concentrated on the package level [[1]], [[12]], their approaches still require more validation. This paper tried to fill this gap. To the best knowledge, there is a little research to compare new packaging approaches in term of package cohesion and package coupling through applying them on a real-world software system. Even though our comparison needs more to be worked on including applying on a big-sized software system, the paper has covered a topic that is interesting and relevant to the domain of software engineering, has some recent references that support that.

3. BACKGROUND

3.1 Functionality-based Approach

According to Jacobson [[23]], functionalities that users need of the OO software are represented by UCs. A UC is reflected by a SD or more that in turn reflect how the objects work to provide functions [[1]]. Unlike other packaging approaches which are applicable at source code level, this approach is proposed to be applied at the architectural design stage to put classes inside packages using SDs. That packaging should decompose the software into packages; each package does one task, which is, as much as possible, completely carried out in the package. To this end, a metric is designed to have two parts: (a) a UC is covered by the minimum number of packages, (b) classes in a package are related so that they contribute to the same set of UCs. These aspects reflect loose coupling and high cohesion, respectively. The packaging quality metric of a package Pi is

$$PackagingQlty(Pi) = wU \times \text{degree of UC coverage by } Pi + wC \times \text{degree of class relevancy of } Pi \quad (1)$$

where wU and wC are the weights of the first and second parts, respectively, so that wU, wC $\in$ [0, 1] and wU + wC = 1.

UC coverage and class relevancy are aggregated to calculate the PackagingQlty. The average PackagingQlty for all software packages represents the new metric as follows:

$$OverallPackaging(system) = Avg\ (PackagingQlty(Pj)) \quad \square \quad Pj \text{ in system} \quad (2)$$

The packaging process tries to maximize Eq. (2). The OverallPackaging metric would be used as the fitness function of the algorithm. The OverallPackaging components and examples of application are found in [1]. This approach relies on the search-based mechanism i.e., heuristic algorithms. Herein, we used a genetic algorithm (GA) as a famous heuristic algorithm proved its success with many optimization problems. Cohesion metrics at the package level have been reviewed in details by [[24]].

3.2 A-KNN-based Approach

A-KNN clustering approach is proposed by [[12]] based on the similarity metric proposed by [[17]]. According to the authors, the approach decreases the computation cost compared with the other approaches. It works as follows: it considers each class as a package; each class is identified with an ID representing the package ID. In the second iteration for k = 3 (k is the number of nearest neighbor to be chosen), the three nearest neighbors to the class that would be packaged are chosen, and their identification are checked. When 2 out of the 3 packages have the same identification, the current class with the same identification of those two classes are identified. However, if the 3 entities do not have the same identification, the current class with the same identification of the closest class (NN) is identified. the packaging process is repeated until no more changes happen. Therefore, the algorithm generates a package at the highest level of the hierarchy. The similarity metric that the approach depends on is described in terms of the connection amount in/out the package.

4. EXTRACTING THE ARCHITECTURAL ARTIFACTS

When the software size increases, packaging process does not only become important but also difficult. In trying to apply our approach to real systems, “data scarcity” issue appeared in our research; UCs and SDs are not available for public access. Therefore, we have the source code of real-world project, JHotDraw and then reverse engineered it to find the required SDs along with the list of classes. Our packaging approach is applied and the remarks are analyzed

according to ASM values. AltovaUModel[1] is used in this extraction and reverse engineering. Although AltovaUModel can produce SDs from source code, such SDs depend on methods behavior at run time. Many statements representing the run time context are often found in the source code of real software systems such as control and loop statements. After generating the SDs, a transient task should then be performed; filtering. The objective of filtering process is to remove all run-time variables and messages displayed in UML-SDs. This makes the produced "run-time" SDs mimic the "functional" SDs that represent the "functional" perspective of UCs via messages and objects starting from the pre-condition to the post-condition. Details of packaging process at early design phase are found in [[1]], [[15]]. Besides, we used XMI2UC tool to find UCs from the XMI documents produced by AltovaUModel. [[25]]. Figure 1 describes this reverse engineering process could be described.

5. THE EXPERIMENT

5.1 Material

We compare both approaches, functionality-based packaging and A-KNN; described in the previous sections. From now on we call the first approach (OverallPackaging). The packaging process should then try to maximize the OverallPackaging metric [[1]].

To this end, we apply Overallpackaging on the same project that Alkhalid et al. [[12]] applied their technique. In particular, they used an open source project, Trama[2]; which consists of 15 classes, 6 packages, 200 methods, and ~6000 LOC. Table 1 describes the classes, considered packages, and the connection amount inside/outside each package.

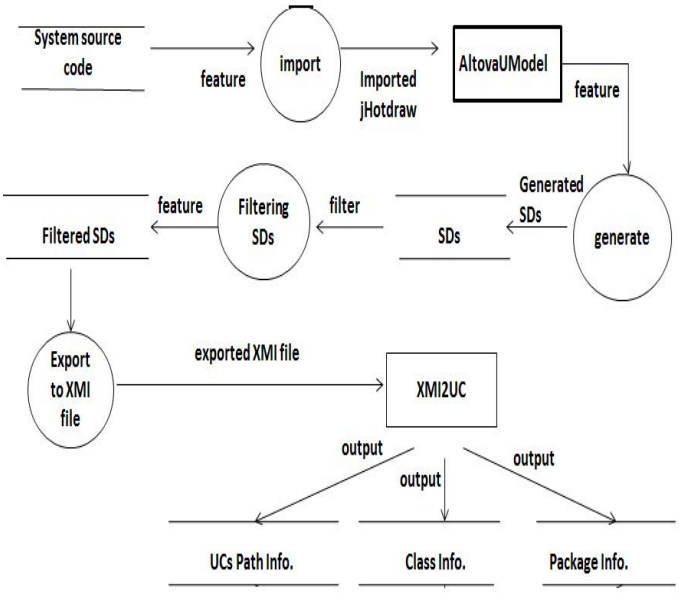

Figure 1: the reverse-engineering process modeled by data flow diagram (DFD)

[1] http://www.altova.com/umodel.html
[2] http://sourceforge.net/projects/trama/

Table 1: Trama raw data

Pkg.#	Pkg.	Class number	Class	Connections in pkg.	Connections out pkg.
1	Negocio	1	ControleProjeto	4	7
		2	ControleTela		
		3	Main		
		4	Matriz		
2	negocio.leitor	5	LeitorDeModelo	0	0
3	negocio.leitor.Interface	6	PluginInterface	0	0
4	Persistencia	7	DadosMatriz	0	0
		8	PersistenciaProjeto		
		9	Projeto		
5	Visao	10	JTableCustomizado	9	4
		11	ModeloTabela		
		12	Tela		
6	visao.renderizador	13	RenderizadorCelula	0	0
		14	RenderizadorTituloColuna		
		15	RenderizadorTituloLinha		
Total				**13**	**11**

As we mentioned earlier, this approach depends on the connection amount; this is computed according to the amount of the class instances used as the package's attribute. Alkhalid et al [[12]] used the same number of packages.

5.2 Results and Discussion

In Alkhalid et al [[12]] experiment, the connection amount in the packages is increased by 3 while the connection amount out the packages is decreased by 3 too. In our packaging approach and after the reverse engineering process described in Section 3, the XMI2UC tool produced 26 UCs from the Trama source code. We also found the OveralPackaging values of the original Trama packaging and that of Trama packaging suggested by A-KNN were 0.42 and 0.48 respectively. That means the packaging suggested by A-KNN is better than the original Trama packaging from OverallPackaging perspective. For OverallPackaging, we conduct an experiment using based on a group genetic algorithm (GGA), a suitable algorithm to identify groups in data [[26]]. To package Trama classes, we used the Evolver tool, v. 6.0. Table 2 states the distribution of Trama classes in the packages after using the three packagings, the original, A-KNN-based, and GGA-Evolver. The values of the parameter settings are 30, 0.5, 0.06, and 50 for population size, crossover, mutation, and number of trials, respectively. We ran the experiment four times to find the best packaging. We varied the values of some optimization parameters such as population size and number of trials based on the size of the system. Therefore, for packaging Trama, population size and number of trials were 30 and 50 respectively; these small

values were chosen according to the small size of Trama. In this research, we are not interested mainly with the performance but with investigation of cohesion and coupling at package level produced by packaging approaches. We ran the experiment until we reach a good result compared with some previous runs; the goodness here is measured in terms of both OverallPackaging. Whenever found it, we stopped the experiment especially if we take into account that the process is computationally intensive where each run for the Trama experiment took few hours. All GGA-Evolver runs achieved a better OverallPackaging value than that of the original packaging and K-NNA; the best value (0.52) was obtained by Run 1 and Run 4 while the worst value (0.49) was obtained by Run 2. According to Run 1 (the best), it is suggested making two changes on Trama modular structure, (1) shifting ControleTela from the package Negocio to Viesa since the number of ControleTela instances in the Viesa is 1 and 0 in the Negocio; i.e., the connection amount out of Viesa is reduced by 1, (2) shifting DadosMatriz from package Persistence to Viesa because the number of class instances of DadosMatriz is 0 with the two packages, no side effect with this change. Table 3 compares the number of connections between the three different packaging approaches: the original packaging, A-KNN-based approach, and the GGA-Evolver packaging shown in Table 2. From Table 3, we noticed that connection amount in the packages is increased by 1 (13 was changed to 14). The package cohesion has then been improved compared with the original packaging. We noticed that the connection amount out the packages is decreased by 1. Contrast to the the original packaging, package coupling has been improved (11 was changed to 10). It is clear that both GGA-Evolver and A-KNN approaches improved the packaging quality compared to the original packaging; they increased cohesion amount and decreased coupling amount. Additionally, A-KNN-based packaging approach achieved better results than GGA-Evolver. However, the key feature of OverallPackaging was performing packaging at architectural design compared with A-KNN approach that could not be used unless the code is available.

Table 3: Connection amount after applying packaging approaches.

Connections	**Original Packaging**	**A-KNN based Packaging**	**GGA-Evolver packaging**
In pkg.	13	16	14
Out pkg.	11	8	10
Total connections	24	24	24

6. TREATS TO VALIDITY

As with any heuristic-based experimental study, the factor that may lead to threat is the selection of parameter setting of the heuristic technique. Herein, if an approach performs better than another approach, it can be due to something other than the approach itself; possibly this could be due to the poor parameter settings of one or more of the heuristic techniques. A factor influencing the generalization validity is that we considered only one open-source system in our comparison. This may not be a good representative of actual systems.

Table 2: Comparison among different packaging: original, A-KNN based and GGA-Evolver

Class	Original pkgg.	A-KNN based pkgg.	GGA-Evolver			
			1st Run	2nd Run	3rd Run	4th Run
	Package number					
ControleProjeto	1.0	1.0	1.0	1.0	1.0	1.0
ControleTela	1.0	1.0	5.0	1.0	1.0	1.0
Main	1.0	1.0	1.0	1.0	1.0	1.0
Matriz	1.0	4.0	1.0	1.0	1.0	1.0
LeitorDeModelo	2.0	2.0	2.0	5.0	2.0	2.0
PluginInterface	3.0	3.0	3.0	3.0	3.0	3.0
DadosMatriz	4.0	4.0	5.0	2.0	1.0	5.0
PersistenciaProjeto	4.0	4.0	4.0	4.0	4.0	1.0
Projeto	4.0	4.0	4.0	5.0	4.0	4.0
JTableCustomizado	5.0	5.0	5.0	5.0	5.0	5.0
ModeloTabela	5.0	1.0	5.0	5.0	5.0	5.0
Tela	5.0	1.0	5.0	5.0	5.0	5.0
RenderizadorCelula	6.0	6.0	6.0	6.0	6.0	6.0
RenderizadorTituloColuna	6.0	6.0	6.0	6.0	6.0	6.0
RenderizadorTituloLinha	6.0	6.0	6.0	6.0	6.0	6.0
Overallpackaging value	**0.42**	**0.48**	**0.52**	**0.49**	**0.51**	**0.52**

7. CONCLUSION AND FUTURE WORK

Software packaging is grouping the OO classes into packages so that a package does one task, which is completely carried out in the package. Because this process is done by changing the structure without affecting its internal behavior, packaging improves the software quality through improving its architecture, maintainability, and reducing future changes. In this paper we compared two packaging approaches proposed recently: functionality-based packaging proposed that uses UCs as an input to the packaging process, and the A-KNN-based packaging approach. The comparison was done through applying the approaches on Trama, an open-source software project. In terms of cohesion and coupling at package level, A-KNN-based approach achieved better results than the functionality-based packaging (OverallPackaging). However, both approaches achieved better results than the original packaging of the system under study. An essential limitation of the A-KNN-based approach is that it relies on source code artifacts so that the packaging would not be done unless the code is available. Because packaging can be performed to remove the erosion produced from software evolution and fix issues in the design a software system, our future research would work at this direction; investigating the impact of packaging on software stability and evolution. Tuning the optimization parameters to improve the results of this study is another open point for further research. The last point for research is to work on multi-level packaging. While, packaging might continue recursively, an architect might not package the classes but packages.

ACKNOWLEDGMENT

We thank KFUPM to provide us with facilities.

REFERENCES

[1] S. Ebad, and M. Ahmed, **Functionality-based software packaging using sequence diagrams**, *Software Quality Journal*, 23(3), pp. 453-481, 2015. https://doi.org/10.1007/s11219-014-9245-3

[2] M. Bauer, and M. Trifu, **Architecture-aware adaptive clustering of OO systems**, in *Proceedngs European Conference on Maintenance and Reengineering (CSMR 04)*, Karlsruhe, Germany, 2004, pp. 3–14.

[3] Y. Chiricota, F. Jourdan, and G. Melancon, **Software components capture using graph clustering**, *The 11th IEEE International Workshop on Program Comprehension (IWPC)*, USA, 2003, pp. 2017-226.

[4] S. Mancoridis, B. Mitchell, C. Rorres, Y. Chen, and R. Gansner, **Using automatic clustering to produce high-level system organizations of source code**, in *Proceedings of the International Workshop on Program Comprehension (IWPC)*. USA, 1998, pp. 45–53.

[5] S. Mancoridis, B. Mitchell, Y. Chen, and R. Gansner, **Bunch: a clustering tool for the recovery and maintenance of software system structures**, in *Proceedings of the IEEE International Conference on Software Maintenance*, USA, 1999, pp.50–59. https://doi.org/10.1109/ICSM.1999.792498

[6] H. Abdeen, S. Ducasse, H. Sahraouiy, and I. Alloui, **Automatic package coupling and cycle minimization**, in *Proceedings of the 16th Working Conference on Reverse Engineering (WCRE)*. Lille, France, 2009, pp. 103-122. https://doi.org/10.1109/WCRE.2009.13

[7] H. Abdeen, S. Ducasse, H. Sahraouiy, and I. Alloui, **Modularization metrics: assessing package organization in legacy large object-oriented software**, in *Proceeding of the 18th Working Conference on Reverse Engineering (WCRE)*, USA, 2011, pp. 394-398. https://doi.org/10.1109/WCRE.2011.55

[8] H. Abdeen, O. Sahraoui, N. Shata, S. Anquetil, and S. Ducasse, **Towards automatically improving package structure while respecting original design decisions**, in *Proceedings of the 2013 20th Working Conference on Reverse Engineering (WCRE)*, 2013, pp. 212–221, https://doi.org/10.1109/WCRE.2013.6671296

[9] O. Seng, M., Bauer, M. Biehl, and G.Pache, **Search-based improvement of subsystem decompositions**, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'5)*, 2005, pp. 1045-1051.

[10] M. Galster, and D. Weyns, **Empirical research in software architecture: how far have we come?**, in *Proceeding of The 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, Venice,Italy, April 2016, pp. 5-8.

[11] M. Alshayeb, **The impact of refactoring on class and architecture stability**, *Journal of Research and Practice in Information Technology*, 43(4), pp. 269-284, 2011

[12] M. Alshayeb, **Empirical investigation of refactoring effect on software quality**, *Information and Software Technology*, 51(9), pp. 1319–1326, 2009

[13] A. Alkhalid, and M. Alshayeb, and S. Mahmoud, **Software refactoring at the package level using clustering techniques**, *IET Software*, 5(3), pp. 274-286, 2011

[14] I. Candela, G. Bavota, B. Russo, and R. Oliveto, **Using cohesion and coupling for software remodularization: is it enough?**, *ACM Transactions on Software Engineering and Methodology*, 25(3), 2016 https://doi.org/10.1145/2928268

[15] M. Harman, J. Clarke, **Metrics are fitness functions too**, in *Proceeding of the 10th International Symposium on Software Metrics*, USA, 2004.

[16] S. Ebad, and M. Ahmed, **Software packaging approaches – a comparison framework**, in *Proceedings of The 5th European Conference on Software Architecture (ECSA 2011)*, Essen, Germany, 2011, pp. 438-446.

[17] M. Fowler, **Reducing coupling**, *IEEE Software*, 2001.

[18] C-H. Lung, X. Xu, M. Zaman, and A. Srinivasan, **Program restructuring using clustering techniques**, *The Journal of Systems and Software*, 79(9), pp. 1261–1279, 2006. https://doi.org/10.1016/j.jss.2006.02.037

[19] A. Corazza, S.D. Martino, V. Maggio, and G. Scanniello, **Investigating the use of lexical information for software system clustering**, in *Proceeedings of The 15th European Conference on Software Maintenance and Reengineering*, Oldenburg, Germany, March 2011, pp. 35-44.

[20] M. Risi, G. Scanniello, and G. Tortora1, **Using fold-in and fold-out in the architecture recovery of software systems**, *Formal Aspects of Computing* (24), pp. 307–330, 2012

[21] G. Bavota, A.D. Lucia, A. Marcus, and R. Oliveto, **Using structural and semantic measures to improve software modularization**, *Empirical Software Engineering*, 18(5), pp. 901-932, 2013 https://doi.org/10.1007/s10664-012-9226-8

[22] Y. Lee, B. Liang, S. Wu, and F. Wang, **Measuring the coupling and cohesion of an object-oriented program based on information flow**, in *Proceedings of the International Conference on Software Quality*, Maribor, Slovenia, 1995, pp. 81–90.

[23] D. Poshyvanyk, A. Marcus, R. Ferenc, and T. Gyimóthy, **Using information retrieval based coupling measures for impact analysis**, *Empirical Software Engineering*, 14(1), pp. 5–32, 2009

[24] I. Jacobson, ***Object-oriented software engineering: a use case driven approach***. Addison-Wesley, 1992.

[25] S. Ebad, and M. Ahmed, **An evaluation framework for package-level cohesion metrics**, in *Proceedings of the 2nd International Conference on Future Information Technology (ICFIT)*. Singapore, 2011.

[26] S. Ebad, and M. Ahmed, **XMI2UC: an automatic tool to extract use cases from object-oriented source code**, in *Proceeding of the International Conference on Advancements in Information Technology (ICAIT)*, Hong Kong, 2012. https://doi.org/10.7763/IJFCC.2012.V1.50

[27] E. Falkenauer, ***Genetic algorithms and grouping problems***. New York, Wiley, 1998.