Software Packaging Approaches – A Comparison Framework.

Conference Paper · September 2011

DOI: 10.1007/978-3-642-23798-0_44 · Source: DBLP

Shouki Ebad
Northern Border University
35 PUBLICATIONS **228** CITATIONS
SEE PROFILE

Moataz Ahmed
Aljand
41 PUBLICATIONS **354** CITATIONS
SEE PROFILE

Software Packaging Approaches —A Comparison Framework

Shouki A. Ebad[1,1], and Moataz Ahmed[1]

[1] Information and Computer Science Department,
King Fahd University of Petroleum and Minerals,
Dhaharan 31261, Saudi Arabia
{shouki, moataz}@kfupm.edu.sa

Abstract. Effective software modularity brings many benefits such as long-term cost reduction, architecture stability, design flexibility, high maintainability, and high reusability. A module could be a class, a package, a component, or a subsystem. In this paper, we are concerned with the package type of modules. There has been a number of attempts to propose approaches for automatic packaging of classes in OO development. However, there is no framework that could be used to aid practitioners in selecting appropriate approaches suitable for their particular development efforts. In this paper we present an attribute-based framework to classify and compare these approaches and provide such aid to practitioners. The framework is also meant to guide researchers interested in proposing new packaging approaches. The paper discusses a number of representative packaging approaches against the framework. Analysis of the discussion suggests open points for future research.

Keywords: automatic software packaging, software architecture, software modularization, optimization

1 Introduction

From the object-oriented (OO) software engineering perspective, it is well recognized that good organization of classes into identifiable and collaborating modules improves the understandability, architecture stability, maintainability, testability, and reusability, all leads to more long-term cost-effective development [1]. The software architecture design effort addresses the problem of structuring the software system under development into modules; this is to include issues such as interconnections between modules, assignment of behaviors to modules, and scalability of modules to larger solutions [2]. A module could be a class, a package, a component, or a subsystem. When considering OO development, the design is a collection of related classes. If there are only few classes in the entire application, we may regard the class decomposition as the architecture. However, as the number of classes grows, such a set of classes can no longer suffice as the architecture because it would be too complex to comprehend at once according to the divide and conquer principle. In this case, the architecture is looked at as packages along with their interconnections; such packages are recursively divided into sub-packages and so on until a level of packages that is easily comprehendible for detailed design is achieved. At the lowest level, a package would be a collection of related classes. Terminology-wise, packages, at the

[1] Corresponding author. Present addresses: Information and Computer Sciences Department, King Fahd University of Petroleum and Minerals, Dhahran 31261, P.O. Box 1594, Saudi Arabia. Tel.: +966 3 860 2464.(Off), + 966 3 860 5940 (Res). fax: +966 3 860 2174

highest level, which can be offered as standalone applications, are referred to as *sub-systems*. Packages, at lower levels, which are functionally cohesive enough, are referred to as *components*. Lowest levels packages are referred to as simply *packages*.

Considering OO development, it is typical for the requirements analysis and system modeling activity to produce a conceptual class model as input to the architecture design activity. In this case, the architect conducts bottom-up packaging of conceptual classes into higher level packages for later detailed design effort. The packaging process are typically guided by some objectives; for instance, it is very common for architects to design for *architecture stability* where the system is structured in a way that permits changes to a system to be made without the architecture having to change [7]. In this case, the effectiveness of the packaging process could be measured by measuring how highly cohesive (i.e., strong intra-package dependencies) and loosely coupled (i.e., weak inter-package dependencies) the resultant packages. Our literature survey, however, revealed that effective bottom-up packaging could be challenging due to two major problems: 1) the lack of effective metrics to be used in guiding the packaging process; and 2) the exponential number of possible packages to be examined for best structure. The first problem is beyond the scope of this paper. In this paper we survey the approaches proposed to address the second problem. The problem has been addressed by many researchers as a problem of NP-hard combinatorial optimization problem to determine the optimal grouping of possibly large, but finite number of classes [1][4][5][8]. There has been a number of attempts to propose packaging approaches. However, to the best of our knowledge, there is no framework that could be used to classify and compare such packaging approaches. In this paper we present a framework to facilitate such classifications and comparisons based on a set of attributes identified as a result of an intensive survey of existing approaches. The framework is meant to aid practitioners in selecting appropriate approaches suitable for their particular development efforts. Moreover, the framework can also guide researchers interested in proposing new packaging approaches. The paper discusses six representative packaging approaches against the framework.

The rest of this paper is organized as follows. Section 2 presents a set of attributes that makes up our comparison framework. A set of representative packaging approaches is discussed in Section 3. Section 4 summarizes our observations on the analysis of the approaches against the framework. Section 5 concludes the paper and offers some directions for future work.

2 Comparison Framework

Judging a packaging approach should not only be based on its effectiveness and efficiency, but also on the underlying characteristics that affect its effectiveness and efficiency. Throughout surveying existing related work on packaging approaches, we identified some attributes that can be used for classifying and comparing different packaging approaches. We expect this set of attributes to help in enhancing existing packaging approaches as well as guiding researchers trying to develop new packaging approaches. Our proposed attributes are discussed in the sequel.

Packaging Goal: In all related work, the goals of packaging come before anything else described. This is important to be able to evaluate packaging achievement. The packaging goal affects the way the packaging process is conducted and the kind of measures to be used to assess the process. Getting on more understandable system and reducing the maintainability effort are examples of the packaging goal.

Underlying Principle: To better understand a concept, it is important to understand the underlying principle upon which the work is built.

Input Artifact: This attribute determines the different inputs required by the packaging approach. Source code and class diagram are examples of the input artifact.

Internal Quality Attribute: This attribute reflects the internal design attribute that guides the packaging process. Cohesion, complexity, length, coupling, and size are examples of such internal attributes.

Search Algorithm: Because packaging could be treated as a search problem, most packaging approaches use heuristic search methods. This attribute indicates the used search algorithm (heuristic or exact) and lists, in case of heuristic, the main features of the used algorithm such as representation and parameter selection.

Fitness Function: In most of the heuristic search techniques, the packaging objective is converted into an objective function which is furthermore converted into a fitness function that is to be optimized to find a solution for the problem. Fitness functions on software packaging are nothing more than software design metrics.

Scalability: According to [4], the software system is small if the number of classes is less than 15 classes. In case of small-sized software systems, the packaging process becomes a simple problem because we can find a polynomial- time algorithm to solve it. The scalability attribute measures the capability of a packaging approach to scale up or scale out in terms of the software size. In general, rough categorization can be used: small (between 15 and 25 classes) or large (more than 25 classes).

Soundness: This attribute reflects whether the packaging approach was shown to group the classes in the way the approach claims it would.

Practicality: This attribute reflects aspects of practicality of the packaging approach. This includes amount of resources (e.g. time, memory) that the packaging approach consumes in generating the right packaging.

Supportability: To know whether the packaging approach is supported by some kind of automation and integrated with CASE tools.

3 Packaging Approaches

We present a summary discussion of six representative packaging approaches based on our set of attributes. The list of considered approaches in our study is not exhaustive, but we gave attention to those works we considered significant and more recent as regards the subject under discussion. We also discuss the shortcomings associated with the considered approaches.

3.1 Doval et al 1999 [5]

They defined the Modularization Quality (MQ) of a system as an objective function to quantify the quality of a given module dependency graph MDG They used MQ as the objective function of their Genetic Algorithm to express the trade-off between intra- and inter-connectivity attributes. MQ achieved this trade-off by subtracting the average inter-connectivity from the average intra-connectivity. Table 1 discusses this approach based on our attributes.

Table 1. Approach of Doval et. al. 1999

Attribute	Comments
Goal	To simplifying the system structure
Principle	Module Dependency Graph (MDG)
Input	Source code
Internal Att.	Intra connectivity and inter connectivity
Search Alg.	GA: numeric encoding, crossover rate = 0.8, mutation rate = 0.004, roulette wheel selection, population size = 10n where n is the number of nodes in the MDG
Fitness	Tradeoff between inter- and intra-connectivity. This trade-off is achieved by subtracting the average inter-connectivity from the average intra-connectivity. It is bound between -1 and 1
Scalable	Scale out; small-sized software system (Mini-Tunis) with 20 modules
Sound	It suffers from the module misplacement problem That makes the solution sub-optimal
Practical	No mention
Support.	The Bunch tool [9] that automatically creates a system decomposition

3.2 Liu et al 2001 [8]

A challenging problem in Email environments is optimally allocating users to servers. This paper presented a method for decomposing a large number of objects (users) into mutually exclusive groups (servers) where within-group dependencies are high and between-group dependencies are low. Their ultimate goal was to minimize network traffic. Based on our attributes, Table 2 shows our discussion of the approach.

Table 2. Approach of Liu et. al. 2001

Attribute	Comments
Goal	To find arrangement of objects on some groups to minimize network traffic
Principle	Graph represented by frequency matrix
Input	Simulated data that represents the messages being sent amongst objects
Internal Att.	Within-group dependencies and between-group dependencies
Search Alg.	Group GA: numeric encoding, random selection, fixed and variable the crossover rate based on the number of function calls
Fitness	The fitness function reflects the work objective by rewarding groups where there is a lot of communication between members
Scalable	Scale up; large-sized system; 250 objects (users) and 5 groups (servers).
Sound	Evolutionary Algorithm is the best compared to clustering (PAM and PAM-M), and Hill Climbing. When a small modification is made to the crossover rate, HC catches EA up.
Practical.	For n objects, n(n+1)/2 elements need to be stored
Support.	Not supported

3.3 Chiricota et al 2003 [3]

They presented a method for finding relatively good clustering of software systems. Their method exploits a metric based clustering of graphs. To evaluate the resultant structure, they used the MQ metric (Approach of Doval et. al. 1999). Table 3 presents a summary discussion of this approach according to our attributes.

Table 3. Approach of Chiricota et. al. 2003

Attribute	Comments
Goal	To achieve this principle “cohesive subsystems with loosely interconnected“
Principle	Small-world graph (undirected)
Input	Source code
Internal Att.	Coupling
Search Alg.	They used the min-cut algorithm consisting in finding a clustering made of several distinct subsets or blocks c_1, . . . , cp such that the number of edges connecting nodes of distinct blocks is kept to a minimum
Fitness	Edges between subsystems are weak edges if their value falls below a given threshold. Once those edges have been deleted, the connected components of the induced graph correspond exactly to the required cluster structure
Scalable	Scale up; three large-sized systems: ResynAssistant, MacOS9, and MFC
Sound	A large number of nodes at the three applications is left isolated
Practical	Short computing time
Support.	Not supported

3.4 Bauer and Trifu 2004 [2]

Because the recovered software architecture is not always meaningful to a human software engineer, this paper proposed an approach that combines clustering with pattern-matching techniques to recover meaningful decompositions. Table 4 discusses this approach based on our attributes.

Table 4. Approach of Bauer and Trifu 2004

Attribute	Comments
Goal	To bring recovered subsystem decompositions closer to what an expert would produce manually
Principle	Un-clustered graph
Input	Source code
Internal Att.	Accuracy (meaningful) and optimality (cohesion and coupling)
Search Alg.	A two-pass MMST (modified minimum spanning tree)
Fitness	It is based on two criteria: accuracy (primary) and optimality (secondary). Decomposition is accurate if it is “meaningful” to a software engineer. This includes: 1) the subsystems contain only semantically related components 2) all semantically related components should be in a single subsystem. Decomposition is optimal if the subsystem: high cohesion and low coupling..
Scalable	Scale up; a large-sized system (Java AWT) with 482 classes
Sound	In terms of optimality, it does not significantly improve the decomposition
Practical	Some phases in this approach are time and memory consuming
Support.	It is not supported

3.5 Seng et al 2005 [10]

They expressed the task of improving a subsystem decomposition as a search problem. Software metrics and design heuristics are combined into a fitness function which is used to measure the quality of subsystem decompositions. Table 5 summarizes this approach based on our attributes.

Table 5. Approach of Seng et. al. 2005

Attribute	Comments
Goal	To determine a decomposition with fewer violations of design principles
Principle	Directed graph
Input	Source code
Internal Att.	cohesion, coupling, complexity, cycles, and bottleneck
Search Alg.	Group GA: an adapted crossover, mutation are split & join, elimination, and adoption, tournament selection, the initial decomposition is the existing one
Fitness	A multi modal function (cohesion, coupling, complexity, cycles, bottleneck).
Scalable	Scale up; a large-sized system (JHotDraw) with 207 classes
Sound	It does not improve the cohesion and coupling
Practical	The execution is fast because of using the efficient tournament selection
Support.	Not supported

3.6 Abdeen et al 2009 [1]

They addressed the problem of optimizing existing modularizations by reducing the inter-package connectivity; this reduction is inspired from well-known package cohesion and coupling principles. Compared to the other approaches, this approach allows maintainers to define some constraints such as package size and the limit of modifications on the original modularization Discussion of the approach based on our attributes is described in Table 6.

Table 6. Approach of Abdeen et. al. al. 2009

Attribute	Comments
Goal	To optimize the decomposition of system into packages so that the resulting organization reduces connectivity between packages
Principle	Directed graph
Input	Existing modularization
Internal Att.	Inter-package connections/cyclic dependencies
Search Alg.	Simulated Annealing, CoolingSchd.(T)=0.9975×T, p>e^(-Tcurrent/Tstart)
Fitness	The average of dependency quality (which relies on CCP and ADP package principles) and connection quality (which relies on CRP and ACP package principles)
Scalable	Scale up; four large-sized systems JEdit , ArgoUML, Jboss, and Azureus with 802, 1671, 3094, and 4212 classes respectively
Sound	It results packages having no classes i.e., empty packages. The percentage of empty packages exceeds 25% of some application classes. In addition, this approach does not allow to remove empty packages from the system
Practical	No mention
Support.	Not supported

4 Observations

Based on the above analysis of each packaging approach, the primary observations of this study can be summarized as follows:

- Most packaging approaches deal with packaging as an optimization problem.

- Most packaging approaches consider maximizing intra-package cohesion and minimizing inter-package coupling as the optimization objective function.
- Most packaging approaches use Genetic Algorithms as a heuristic search method.
- All packaging approaches are graph based.
- Most packaging approaches use the source code as their input artifact. This reveals that there is a lack of approaches that would help in packaging early during the architectural design phase; this is the time when packaging might be needed the most for better architectural and component design. This remains a research area in need of more efforts where only conceptual models are available.
- Selection of parameters of the heuristic search method such as GA or SA is very crucial. Different parameters settings may result in completely different structures; due to being trapped in local optima as in the approach of Liu et. al.
- Most packaging approaches are scalable.
- The soundness of discussed packaging approaches seems to be questionable. For examples, Doval et al may result in module misplacement, Chiricota et al 2003 may result in isolated nodes, Bauer and Trifu may result in no significant improvement, and Abdeen et al. may result in empty packages.
- Except Doval et al, all packaging approaches are not supported by CASE tools.

Table 7 summarizes the packaging approaches surveyed based on our attributes.

Table 7. Summary of the existing packaging approaches based on our attributes

Study	Goal	Input	Principle	Internal Att.
Doval et. al. 1999	To simplify the system structure	source code	Graph	Intra connectivity and inter connectivity
Liu et. al. 2001	To minimize network traffic	simulated data	Graph	Within-group dependencies and between-group dependencies
Chiricota et. al. 2003	To achieve a design principle (Cohesion & Coupling)	source code	Graph	Coupling
Bauer & Trifu 2004	To produce meaningful decomposition	source code	Graph	Accuracy (meaningful) & optimality (cohesion & coupling
Seng et. al. 2005	To decompose the system with fewer violations of design principles	source code	Graph	Cohesion, coupling, complexity, cycles, and bottleneck
Abdeen et. al. al. 2009	To achieve a design principle (Cohesion & Coupling)	existing modularization	Graph	Inter-package connections/cyclic dependencies

Table 7. (Continue)

Search Alg.	Fitness	Scalable	Sound	Practical	Support
GA	Tradeoff between inter- and intra-connectivity	No	No	N/A	Yes
GA	Penalizing groups where there is no communication	Yes	No	No	No
Min-cut	Deleting the weak edges	Yes	No	Yes	No
MMST	High internal cohesion & low external coupling	Yes	No	No	No

GA	A multi modal fitness function	Yes	No	Yes	No
SA	Averaging of some quality metrics	Yes	No	N/A	No

5 Conclusion and Future Work

In this paper we presented an attribute-based framework to allow classifying and comparing approaches for packaging classes during software development. The paper also provides an analysis of a representative set of packaging approaches in light of the framework. The results of the study provides practitioners with an overview of prominent work in the literature and offer help with regard to making decisions as which approach would be appropriate for their particular development efforts. Moreover, this analysis is meant to serve as guide for researchers interested in developing new packaging approaches. The analysis indentifies some open issues for future work.

Clearly, package-level metrics play crucial role in guiding the packaging process. Unfortunately, package level metrics did not get that much attention of researches as class level metrics did [6]. As a follow-up to the work presented in this paper, the authors are currently working on analyzing the metrics used in the packaging approaches. The authors are working on developing package-level metrics that can guide packaging process towards the development of highly stable architectures.

References

1. Abdeen H., Ducasse S., Sahraouiy H., Alloui I., " Automatic Package Coupling and Cycle Minimization," WCRE '09, pp. 103-122, IEEE CNF, 2009
2. Bauer M. and Trifu M., "Architecture-aware adaptive clustering of OO systems," CSMR'04, pages 3–14. IEEE Computer Society, 2004
3. Chiricota Y., Jourdan F., Melancon G, "Software components capture using graph clustering," The 11th IEEE Int'l Workshop on Program Comprehension (IWPC'03), 2003
4. Clarke J., Dolado J., Harman M., Jones B., Lumkin M., Mitchell B., Mancoridis S., Rees K., Roper M., Shepperd M., "Reformulating software engineering as a search problem," IEEE Proceedings on Software, 150 (3), pp. 161–175, Jun 2003
5. Doval, D.; Mancoridis, S.; Mitchell, B., "Automatic clustering of software systems using a genetic algorithm," STEP '99, IEEE Computer Society, 1999
6. Genero M., Piattini M., Calero C., "A Survey of Metrics for UML Class Diagrams," Journal of Object Technology JOT 2005, 4(9), Nov/Dec 2005
7. Lethbridge T., Laganière R., Object-Oriented Software Engineering: Practical Software Development using UML and Java, McGraw Hill, 2005
8. Liu X., Swift S., Tucker A., "Using evolutionary algorithms to tackle large scale grouping problems," GECCO'01, 2001
9. Mancoridis S., Mitchell B., Chen Y., Gansner E.,"Bunch: A clustering tool for the recovery and maintenance of software system structures," ICSM '99, IEEE Computer Society Press, 1999
10. Seng O., Bauer M., Biehl M., Pache G.,"Search-based Improvement of Subsystem Decompositions," Proc. of the GECCO'05, 2005