Methods for Selecting and Improving Software Clustering Algorithms

Mark Shtern and Vassilios Tzerpos
York University
Toronto, Ontario, Canada
{mark,bil}@cse.yorku.ca

Abstract

Several software clustering algorithms have been proposed in the literature, each with its own strengths and weaknesses. Most of these algorithms have been applied to particular software systems with considerable success. However, the question of how to select a software clustering algorithm that is best suited for a specific software system remains unanswered.

In this paper, we introduce a method for the selection of a software clustering algorithm for specific needs. The proposed method is based on a newly introduced formal description template for software clustering algorithms. Using the same template, we also introduce a method for software clustering algorithm improvement.

1 Introduction

Software clustering is a research area that has been developing for a long time. Many different software clustering algorithms have been developed. In this paper, we focus only on software clustering algorithms that gather software components into subsystems that are important to someone attempting to understand the software system. The output of such a software clustering algorithm is called a decomposition of the software system.

Since software clustering algorithms are often heuristic, the quality of the produced decomposition maybe be dependent on whether the software system was a good match for the heuristic. Although the selection of a software clustering algorithm plays a key role for the construction of a meaningful decomposition, there are no guidelines on how to select an appropriate algorithm. This paper introduces such a method for algorithm selection.

Furthermore, we introduce a method for the improvement of existing software algorithms. Our approach allows the evaluation of the new algorithm in earlier stages before it is fully implemented and finalized.

Our final contribution is the development of a formal description template that allows to formally specify a software clustering methodology. The adoption of the template by software clustering researchers should lead to better communication in this research field and the development of more effective software clustering approaches.

Our discussion will be based on two well-known software clustering algorithms:

- **ACDC**. This is a pattern-based software clustering algorithm that attempts to recover subsystems commonly found in manually-created decompositions of large software systems [9].
- **Bunch**. This is a suite of algorithms that attempt to find a decomposition that optimizes a quality measure based on high-cohesion, low-coupling [4].

The structure of the rest of this paper is as follows. Section 2 presents the formal description template as well as the two methods for algorithm selection and algorithm improvement. Discussion on the benefits of the methods presented takes place in Section 3. Finally, Section 4 concludes the paper.

2 Describing software clustering algorithms

Part of the difficulty in comparatively evaluating software clustering algorithms stems from the fact that each algorithm makes a different set of assumptions about the task at hand, such as requiring different types of information as input. Since there is no established standard for the description of a new software clustering algorithm, it is hard to gauge the relative merits of different approaches.

In this section, we propose a consistent format for the description of software clustering algorithms, not unlike the one used for design patterns [1]. In Sections 2.1 and 2.2 respectively, we use the terminology introduced here to develop methods for the selection of appropriate algorithms, as well as the improvement of existing algorithms.

The template for the description format is as follows:

- **Algorithm Name**. While a descriptive name is preferable, algorithm naming is entirely up to the algorithm developer.

- **Algorithm Intent**. A description of the goals behind this software clustering algorithm. Examples of such goals include:

  - To decompose a software system into subsystems
  - To recover the architecture of a software system
  - To discover possible ways to restructure the software

  Explicitly stating algorithm intent can simplify selecting a software clustering algorithm for reverse engineers, as well as understanding a clustering algorithm for software clustering researchers.

- **Factbase properties**. A description of the input that the software clustering algorithm expects (an algorithm's input is often called a factbase since it contains facts extracted from the software system). For example, software clustering algorithms often expect that the factbase includes only dependencies between modules [8, 9, 4, 3].

- **Clustering objective**. A description of the ideal output of the software clustering algorithm. Based on experience or heuristics, the designer of the algorithm decides what a meaningful decomposition of a software system should look like. For example, Schwanke [7] concluded that a software decomposition with maximum cohesion and minimum coupling is important for software maintainers.

- **Process Description**. A brief description of the main idea behind the algorithm's implementation. For example: "ACDC creates clusters by detecting established subsystem patterns in the given software system. Software entities that are not clustered this way are assigned to the subsystem that depends the most on them".

- **Decomposition Properties**. A description of the properties of the decompositions that the software clustering algorithm creates. These properties are either a direct result of the clustering objective (e.g. Bunch creates decompositions whose value of the objective function is higher than that for other algorithms [4]) or an artifact of the algorithm's implementation (e.g. Bunch creates decompositions that are more balanced than most algorithms [11]).

- **Algorithm Restrictions**. Since software clustering algorithms are heuristic algorithms, their performance may depend on the type of software system being clustered. Such dependencies need to be documented here. For example, Bunch may not be well-suited for event-driven systems [6].

- **Failed assumptions**. A description of ideas that did not work very well while developing the algorithm. This information is often omitted in publications but could be invaluable to the software clustering researcher.

- **Detailed algorithm description**. The nuts and bolts of the algorithm's implementation are presented here. This will often be the only section of significant length in an algorithm description.

2.1 Algorithm Selection

In this section, we present a method that utilizes the notions of decomposition properties and algorithm restrictions introduced above in order to allow a reverse engineer to select an appropriate clustering algorithm for a given software system (or conversely to reject an algorithm that is not well suited for the task).

The proposed algorithm selection method has three stages:

1. **Construct prototype decomposition**. A prototype decomposition is a crude, high-level decomposition of the software system that can be constructed with a small amount of effort utilizing expert knowledge, existing documentation, and the reverse engineer's experience. It only needs to describe the top level subsystems in the software system, i.e. nested subsystems or dependencies between subsystems are not required. Several efficient methods for the construction of such a view have been presented in the literature [5].

   The purpose of the prototype decomposition is to stand in as a model for an authoritative decomposition. It will be used to verify the usability and applicability of decompositions created by the candidate software clustering algorithms.

2. **Apply algorithm restrictions**. The formal description of any candidate software clustering algorithm will contain a number of algorithm restrictions. If the given software system or its prototype decomposition match any of these restrictions, then the algorithm is rejected.

3. **Match decomposition properties**. A candidate software clustering algorithm is selected if the prototype decomposition can be produced by the algorithm. This

means that the decomposition properties of the candidate algorithm must be consistent with the prototype decomposition, something that is usually easy to verify, either manually or with the help of existing tools. The reverse engineer needs to decide what level of discrepancy is too high for the algorithm to be selected.

Many software clustering algorithms have configuration parameters that need to be calibrated once an algorithm has been selected. The proposed method can help with this task as well. The reverse engineer can measure the effect that changes to the configuration parameters have on the aforementioned discrepancy between the decomposition properties of the candidate algorithm and the prototype decomposition. Parameter values that minimize this discrepancy can be utilized for the clustering task.

2.2 Algorithm Improvement

Software clustering researchers strive to improve the effectiveness of their algorithms, either by conceiving new clustering objectives, or by improving the process that attempts to realize existing objectives. A typical researcher work flow is shown below:

1. Design a new clustering objective or a new clustering process.
2. Implement the new approach.
3. Apply the new approach to a reference system with an existing authoritative decomposition.
4. Compare the produced decomposition to the authoritative decomposition.
5. Repeat from step 1 until satisfactory results are obtained.

The main drawback of the above work flow is that an implementation is required before the new approach can be evaluated. However, the logic of a software clustering algorithm is usually complicated. As a result, the implementation phase often requires a large time investment. It would be beneficial if new clustering ideas could be evaluated without a time consuming implementation. In the following, we present a new work flow that allows for an early evaluation of clustering ideas.

The proposed work flow has eight stages:

1. **Idea conception**.

   Based on her experience and domain knowledge, a software clustering researcher can conceive of novel clustering objectives, decomposition properties, or factbase properties that could help improve the effectiveness of existing software clustering algorithms.

2. **Reference system selection**.

   The reference software systems will be used to verify the feasibility of the new clustering idea. Their selection depends on the research goal. When the goal is to evaluate a new clustering objective, various types of software systems should be selected as reference systems. When the goal is to evaluate new approaches to achieve existing clustering objectives, then the authoritative decomposition of the selected reference systems has to comply with the stated clustering objective.

3. **Authoritative decomposition construction**.

   If an authoritative decomposition exists for a reference system, then it can be used directly. Otherwise, a prototype decomposition needs to be constructed as explained in Section 2.1.

4. **Idea verification**.

   The purpose of this stage is to determine whether the authoritative decompositions of the selected reference systems have properties that reflect the clustering idea being considered. This can be done either manually or by developing a verification application. In either case, the amount of time required for this stage should be significantly less than fully implementing and testing the clustering idea. As a result, a researcher can evaluate the feasibility of the new clustering idea much earlier.

5. **Verification result analysis**.

   If the verification stage determined that the reference systems are compatible with the clustering idea, then the full implementation of the idea can commence in the next stage. However, if an incompatibility was detected, then one of the following two actions should be taken:

   (a) A new set of reference systems is selected and the verification process repeated. If successful, then implementation can begin, but the approach's final documentation will need to include an algorithm restriction documenting the fact that this approach is not appropriate for all software systems.

   (b) The new clustering idea is rejected. In this case, the Failed assumptions section of the approach's documentation needs to include this negative result.

6. **Algorithm implementation**.

   The full implementation of the new clustering idea takes place during this stage.

7. **Algorithm evaluation**.

   The effectiveness of the new clustering idea can be evaluated by comparing its results to the results of existing approaches by using established evaluation criteria, such as MoJo [10] or the Koschke-Eisenbarth measure [2].

8. **Final documentation**.

   The last stage of our proposed work flow requires that a formal description of the new clustering approach is created following the description template presented in Section 2. This way the software clustering research community can be made aware of all results, both positive and negative, obtained during this process.

3 Discussion

In this section, we outline some of the advantages of using the formal description template as well as the algorithm selection and improvement methods presented in this paper.

Selecting strategies for algorithm improvement Suppose that a researcher is interested in improving her software clustering algorithm but is unsure how to proceed, i.e. which aspect of the algorithm to optimize. The researcher should select different types of software systems with authoritative decompositions that were created based on the current clustering objective. If the decomposition properties of the algorithm are not found in the authoritative decompositions, then effort must be spent to improve aspects of the algorithm that are responsible for the decomposition properties.

Mixing algorithms The decomposition of a software clustering algorithm into clustering objectives, factbase properties and decomposition properties gives researchers the freedom to mix such elements from different algorithms. When a software clustering algorithm is considered for improvement it can possibly address some of its weaknesses by borrowing strong elements from other algorithms (it can also avoid pitfalls by studying failed assumptions of other algorithms).

Evaluating software clustering without authoritative decompositions Almost all clustering evaluation methods in the literature require an authoritative decomposition. In its absence, evaluating clustering results is practically impossible, partly because it is hard to identify what the correct decomposition for a given clustering objective is. When every software clustering algorithm will have a more formal definition of the decomposition properties it aims to produce, it will be easier to develop methods to compare algorithms. For instance, algorithms that construct software decomposition based on the principle of high cohesion and low coupling can be compared by measuring how well they comply to that principle.

Verification of implementation After a software clustering algorithm is implemented, it has to be tested. The testing of a software clustering algorithm is difficult because it is often hard to know whether an unsuccessful result is due to an implementation bug or to an inappropriate clustering objective. When decomposition properties have been identified, then a researcher has new tools to verify the implementation of a software clustering algorithm. When the produced decomposition does not comply with the decomposition properties, that indicates a bug in the implementation.

Development of new types of software decompositions The formal description template introduced in this paper allows for the separation between developing a new type of software decomposition and developing a software clustering algorithm that produces such a decomposition. This separation may encourage other members of the reverse engineering community to develop new types of software decomposition without being experts on software clustering. This has the potential to introduce fresh ideas into this research field.

Shared failed assumptions Most of the publications on software clustering algorithm describe success stories. However, there are often many failures that are left undocumented since they did not lead to successful results. This knowledge could be very useful to other researchers. We hope that the “failed assumptions” section in the formal description template will encourage the sharing of such information.

4 Conclusions

This paper presented a formal description template for software clustering algorithms. Using this template, we introduced and discussed two methods for algorithm selection and algorithm improvement.

References

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley, 1995.

[2] R. Koschke and T. Eisenbarth. A framework for experimental evaluation of clustering techniques. In *Proceedings of*

the Eighth International Workshop on Program Comprehension, pages 201–210, June 2000.

[3] C.-H. Lung, M. Zaman, and A. Nandi. Applications of clustering techniques to software partitioning, recovery and restructuring. *J. Syst. Softw.*, 73(2):227–244, 2004.

[4] S. Mancoridis, B. Mitchell, Y. Chen, and E. Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *Proceedings of the International Conference on Software Maintenance*. IEEE Computer Society Press, 1999.

[5] N. Medvidovic and V. Jakobac. Using software evolution to focus architectural recovery. *Automated Software Engg.*, 13(2):225–256, 2006.

[6] B. S. Mitchell and S. Mancoridis. Craft: A framework for evaluating software clustering results inthe absence of benchmark decompositions. In *Proceedings of the Eighth Working Conference on Reverse Engineering*, pages 93–103, Oct. 2001.

[7] R. W. Schwanke. An intelligent tool for re-engineering software modularity. In *Proceedings of the 13th International Conference on Software Engineering*, pages 83–92, May 1991.

[8] A. Shokoufandeh, S. Mancoridis, T. Denton, and M. Maycock. Spectral and meta-heuristic algorithms for software clustering. *J. Syst. Softw.*, 77(3):213–223, 2005.

[9] V. Tzerpos and R. C. Holt. ACDC: An algorithm for comprehension-driven clustering. In *Proceedings of the Seventh Working Conference on Reverse Engineering*, pages 258–267, Nov. 2000.

[10] Z. Wen and V. Tzerpos. An optimal algorithm for MoJo distance. In *Proceedings of the Eleventh International Workshop on Program Comprehension*, pages 227–235, May 2003.

[11] J. Wu, A. E. Hassan, and R. C. Holt. Comparison of clustering algorithms in the context of software evolution. In *ICSM '05: Proceedings of the 21st IEEE International Conference on Software Maintenance*, pages 525–535, Washington, DC, USA, 2005. IEEE Computer Society.