

Using Cluster Analysis to Improve the Design of Component Interfaces

Rahmat Adnan Bas Graaf* Arie van Deursen
Delft University of Technology
The Netherlands

radnan@gmail.com

{b.s.graaf,Arie.vanDeursen}@tudelft.nl

Joost Zonneveld
ASML
The Netherlands

Joost.Zonneveld@asml.com

Abstract

For large software systems, interface structure has an important impact on their maintainability and build performance. For example, for complex systems written in C, recompilation due to a change in one central header file can run into hours. In this paper, we explore how automated cluster analysis can be used to refactor interfaces, in order to reduce the number of dependencies and to improve encapsulation, thus improving build performance and maintainability. We implemented our approach in a tool called “Interface Regroup Wizard”, which we applied to several interfaces of a large industrial embedded system. From this, we not only learned that automated cluster analysis works surprisingly well to improve the design of interfaces, but also which of the refactoring steps are best done manually by an architect.

1. Introduction

As a software system evolves over many years, the interfaces between its source modules are modified as well. Over time, interface structure is likely to deteriorate. This might result in fat interfaces with huge numbers of definitions that are not functionally coherent, and which have many different source files (“users”) depending on them. Especially in the case of large software systems this negatively affects build performance and maintainability. In the C programming language, for instance, a source file can use an interface definition by including a complete header file. As a result, for every change to a single definition in that header file, all source files that use one or more of its definitions need to be recompiled.

As an example, at ASML, a company developing manufacturing machines for the production of microchips, over the course of years build time has increased such that it sometimes affects the development speed of its control software. Recompilation after interface changes can run into hours, leading to loss of productivity of the development team.

A solution to this problem is to refactor interfaces such that the definitions in an interface are (1) functionally coherent (for maintainability), and (2) are used by a similar set

of users (for build performance). Although these criteria are correlated they are not equivalent, resulting in trade-offs.

In this paper, we investigate the use of automatic cluster analysis to address this problem. Hierarchical clustering aims at putting together entities (interface definitions) that are similar (in terms of their users) in the same cluster (interface), while entities in different clusters are less similar.

We are interested in the limits and opportunities of the use of cluster analysis for interface refactoring, and the practical difficulties one encounters in doing so. Therefore, we describe in this paper the application of cluster analysis to refactor the interfaces of a complex industrial embedded software system comprising millions of lines of C code. To that end, we have developed an interactive tool called Interface Regroup Wizard (IRW), which supports a software architect in refactoring the interfaces of C software components.

In this paper, we explore three questions: (1) Can an automatic approach such as cluster analysis be applied to improve the design of interfaces? (2) What are the limits of automation for the refactoring of interfaces, that is, to what extent can this be done automatically? (3) What needs to be done in practice before interfaces can be refactored automatically using cluster analysis?

The paper is structured as follows. The next section offers a summary of related work in the area of cluster analysis and interface redesign. In Section 3, we describe our clustering. Then, in Section 4, we introduce the industrial case that motivated our research. In Section 5 we describe the interactive tool we developed to apply cluster analysis to cases like these. The results of the actual case are covered in Section 6. We conclude the paper with a discussion of our findings, a summary of our contributions, and an outlook to future work.

2. Related Work

Cluster analysis has been applied to a range of software engineering problems [8, 3, 4, 7]. The primary contribution of the present paper is not so much in proposing new clustering algorithms. Instead, we demonstrate (1) how cluster analysis can be successfully applied in a different software engineering domain, interface redesign; (2) that it can be applied to

*Presently at Microsoft Development Center Copenhagen

$$\delta_{ad}(C, D) = \begin{cases} \delta_{co}(C, D), & \text{if } 0 \leq \delta_{co}(C, D) < 1 \\ \delta_{me}(C, D), & \text{if } \delta_{co}(C, D) = 1 \wedge 0 \leq \delta_{me}(C, D) < 1 \\ \delta_{av}(C, D), & \text{if } \delta_{me}(C, D) = 1 \wedge 0 \leq \delta_{av}(C, D) < 1 \\ \delta_{si}(C, D), & \text{if } \delta_{av}(C, D) = 1 \wedge 0 \leq \delta_{si}(C, D) < 1 \end{cases}$$

Figure 1. Adaptive linkage

industry strength systems; and (3) which steps can be done automatically, and which steps are best done manually.

3. Clustering Approach

We apply agglomerative hierarchical clustering [5]. In our case, the entities to be clustered are symbols (which are defined in interfaces). The feature set based on which we cluster is the set of using modules (.c-files using an interface symbol; we refer to these files as “users”). Thus, each symbol is an entity, and each using .c-file is a feature.

As distance metric between symbols we use the Jaccard distance between their sets of users. To determine the distance between two clusters, the minimum (“single”), median, average, or maximum (“complete”) distance between their constituting entities can be used, each with different characteristics in terms of *compactness* (the degree of similarity between the elements in the cluster).

For our purposes, architects would typically prefer results consisting of less than seven clusters (see Section 4). This requirement is in conflict with the need for clusters consisting of symbols with similar user sets (i.e., compact clusters). In order to strike a balance between compactness and a small number of clusters, we propose a hybrid approach to determine inter-cluster distance, which we call *adaptive linkage* and is defined in Figure 1 in terms of the previously mentioned cluster distance metrics.

The process of agglomerative hierarchical clustering iteratively merges the two nearest clusters, resulting in a sequence of clusterings (each containing fewer clusters than the previous one). The selection of a clustering is a decision that we leave to the software architect. Following proposals from Handl and Knowles [6], we aid the architect by presenting him a graph that plots each generated clustering against two validation metrics. The first is the intra-cluster *variance*, defined as the root mean square distance between entities and their cluster’s centre. Variance is a measure for the aforementioned compactness. The second metric is *connectivity*, which is a measure for the degree to which near neighbouring symbols have been placed in the same cluster.

4. Interface Redesign in Practice

The context in which we developed our approach consists of the embedded control software of a wafer scanner developed

Table 1. Interface quality criteria

Dependency	The number of .c-files that needs to be rebuilt into object files that, in turn, need to be linked, due to a modified interface has to be minimized.
Sharing	The number of interfaces that is shared by other interfaces should be minimized.
Encapsulation	We measure encapsulation as the number of symbols actually used by users defined at the same hierarchical level relative to the total number of symbols it defines. Encapsulation should be maximized.
Functional coherence	The symbols defined in an interface should be related to the same functionality.
Stability	Interfaces should be stable; the number of changes should be minimized.

by ASML. This software system comprises approximately 20 million lines of code. During the last eight years ASML’s high-end scanners have been advancing from 150 nm resolution and 80 wafers per hour to 45 nm resolution and 130 wafers per hour. This improvement has been realized by newly designed optical, mechatronic and metrology system parts. It is nearly impossible to prepare software interfaces for such changes in the system, requiring regular interface modifications.

The metamodel we use offers a module view on ASML’s software architecture, and distinguishes *interfaces* (offering a range of *symbols* for types, functions, or macros) as well as *components* (which in their C-files make use of symbols that come from the interfaces they require).

In the case of ASML, the software is structured into hundreds of components on four hierarchical levels, ranging from component via building block and functional cluster to the full system. The components provide more than 1500 interfaces defining over half a million different symbols. Analysis of the source code revealed that in total these symbols are used 1.6 million times. As such, each symbol, on average, is used by approximately three users (i.e., .c-files). Note that multiple uses by the same user are counted as one.

A selection of the overall criteria ASML uses to assess the quality of its interfaces are mentioned in Table 1. For these, ASML specified desired values. For dependency and sharing these are specified relative to a system-wide average. For encapsulation, absolute values are specified. Based on the extent to which these measures for a particular interface deviate from the desired values, each interface can be assigned a compliancy score. All interfaces with a compliancy score above a certain threshold need to be redesigned.

5. Interface Regroup Wizard (IRW)

In order to be able to apply clustering techniques as described in Section 3 to the setting described in the previous section, we have developed a tool, called the *Interface Regroup Wizard (IRW)*. It aims at supporting software architects in the redesign of the interfaces of components in their software system. Full details concerning the design and implementation of the tool can be found in Adnan [1].

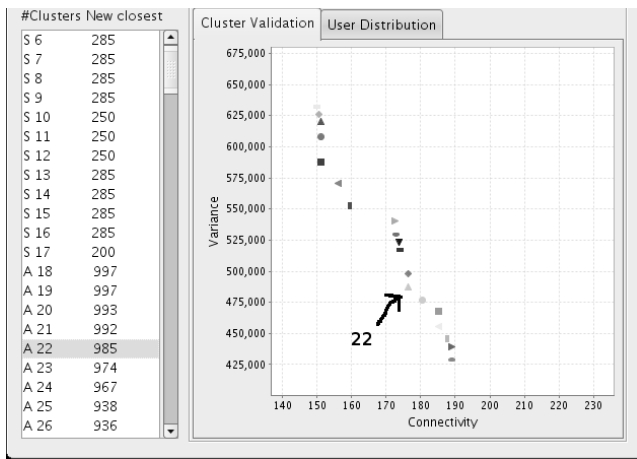


Figure 2. Selecting a clustering

Fact Extraction A fact extraction phase is used to populate a relational database with information concerning (1) the symbol definitions; (2) symbol uses; and (3) the current architectural structure in terms of (sub)components and the interfaces they provide or require.

Clustering The clustering process can be configured using IRW’s main window. The architect can specify, for example, the interface to be regrouped, the clustering method, the hierarchical level of users, whether or not to clusters symbols defined at different hierarchical levels, the maximum distance at which this will be considered, whether or not to include struct members and local users, or to collapse symbols generated from a single source symbol.

During the clustering process the user may be asked questions regarding the merging of clusters containing symbols defined at different hierarchical levels. In principle the tool does not allow this (to improve encapsulation), but when the distance between two clusters at different levels is below a threshold, the user can decide to merge them after all. If there are too many questions the user can decide to cancel the process and lower the distance threshold or prevent clustering of such clusters all together.

Cluster Selection When the clustering process is ready, one of the generated clusterings needs to be selected. To aid the architect with this, the IRW presents a graph that conveys information about the quality of the generated clusterings (see Figure 2), which plots each generated clustering against its variance (vertically) and connectivity (horizontally).

Agglomerative hierarchical clustering starts with a separate cluster for each entity. Such a clustering has low variance and high connectivity and will be plotted in the bottom-right of the graph. Each successive clustering has fewer clusters, a lower connectivity (the clusters of neighbouring entities that gave rise to a penalty might have merged), and a higher variance. This trend can easily be observed from the graph.

The architect can use this graph to select a suitable clus-

tering. By hovering over the plotted clusterings, a tooltip appears that indicates how many clusters that clustering contains. Assuming that the architect has an idea of the upper and lower bounds for the desired number of interfaces, an area of the graph can be selected in order to zoom in for closer inspection of the clusterings within those bounds.

Handl and Knowles [6] propose to search for a clustering for which an extra agglomeration of two clusters results in a large degradation of compactness (i.e., measured by variance), while connectivity is not improved much. Visually such a clustering can be recognized by a ‘knee’ in the validation graph. That clustering might be a good candidate (e.g., the clustering indicated with the arrow in Figure 2).

Candidate clusterings can be further inspected. The IRW can show the contents of the clusters for a clustering. Additionally, it can display such information as (1) the minimum distance between any two clusters for a particular clustering; (2) the hierarchical level at which the interface is visible; (3) a measure for the encapsulation of the proposed interface; (4) the number of symbols at each hierarchical level; and (5) a measure for the number of users that have to be rebuilt in case one of the symbols in the interface changes. The tool also displays average values for encapsulation and dependency for the complete clustering. Using these measures the architect can select a suitable clustering.

Managing Interface Quality Criteria The cluster similarity measures discussed above aim at optimizing compactness and connectivity. Considering the criteria that ASML uses to evaluate interfaces discussed in Section 4, this only relates to *dependency*. Encapsulation, sharing, and functional coherence are addressed by involving the architect in the process by giving interactive control when clusters from different architectural levels are merged and eventually which clustering to select (aided by the different metrics the tool offers).

6. Evaluation

To assess the usefulness of the IRW and to further improve it, we applied IRW to a series of actual ASML interfaces¹. We asked ASML’s high-level architects to identify interfaces in need of a redesign or ones that have recently been redesigned manually. In total we applied the IRW to twenty interfaces of which four had already been redesigned.

For the previously redesigned category, we compared the results with manually redesigned interfaces, which gives good results. For example, for one case we refactored an interface comprising 449 symbols used by 128 different .c files. The distance between our proposal and the refactoring done manually was 23 cluster merges, and 9 symbol moves.

For the remaining interfaces we used the measurable quality criteria defined by ASML (Section 4) to evaluate our results. As an example, one of the interfaces analyzed defines

¹See the technical report [2] for more details.

545 symbols (which is significantly larger than the average size of an interface at ASML). The dependency measure for this interface is 483, i.e., 483 files need to be rebuilt for each single change to its interface. The encapsulation measure is 10%, well below the desired level of 80%, and this interface also shares more other interfaces than desired.

Applying IRW to this interface resulted in the clusterings visualized in Figure 2. Using the heuristic explained earlier, we select the clustering consisting of 22 clusters from this validation graph. Each of the generated clusters complies to ASML's interface criteria. Only one cluster has an encapsulation value of less than 100%. One cluster remains with a relatively high dependency value, but all other clusters have much lower dependency values.

The IRW proposes many more clusters than ASML's architects would typically desire (2–7). Looking closer at these results makes clear that the clustering consists of only a few large clusters and many small clusters. Thus, creating the initial set of (approximately) 20 clusters can be automated, after which the smaller clusters found can be manually merged.

7. Discussion

Overall, the architects at ASML concluded that the interface refactoring proposals generated by the IRW are a valuable starting point for manual refinement and that effort is saved by application of the tool; the required refinements require less effort than a complete manual refactoring.

When we evaluate the IRW-proposals with respect to the number of symbols that end up in a wrong cluster, we conclude that this number increases considerably for clusterings with only a few clusters. The reason for this is that the decisions to merge two clusters become less obvious during the clustering process. For the interfaces of the component discussed in Section 6, for instance, the percentage of identical users of the two clusters closest together dropped below 15% as soon as the remaining number of clusters was 17. Our other experiments showed similar results.

We envision the IRW to be used to obtain a refactoring proposal consisting of a set of clusters that is larger than desired. The architects at ASML typically aim at refactoring an interface into two to seven pieces. It is up to the them to finish the restructuring using domain knowledge. By taking a clustering consisting of more than the desired number of clusters, the final work mainly consists of merging instead of (moving symbols around (which is harder). To summarize, from our experiments we can conclude that automating the restructuring makes sense as long as the decisions made by the tool are obvious in the sense that there is a considerable amount of overlap between the users of the two clusters to be merged.

To aid the architect with the selection of a generated clustering that mainly requires cluster merges instead of symbol moves to finish, the distance between the two closest clusters in the current clustering is particularly useful information

(see Figure 2). It gives an indication of the meaningfulness of the next step in the clustering process. And as the decisions with the type of clustering implemented by the IRW will never be reversed later on (i.e., this value only increases), it gives a clue on what clustering to select.

8. Conclusion

This paper demonstrates how automated cluster analysis can be successfully applied to address the interface redesign problem for industrial, large scale software systems. The main contributions of the paper include (1) an operationalisation of quality criteria guiding interface redesign; (2) a cluster analysis approach tailored towards interface redesign, in which automated analysis yields a first proposal, after which the architect can focus on merging some of the smaller remaining clusters; (3) implementation of our approach in the *Interface Regroup Wizard (IRW)*; (4) application of the approach to 20 industrial components at ASML.

Our future work encompasses further refinements of the tool, in particular related to dealing with layered components, and further applications to existing components, both from within ASML as well as from the open source domain,

Acknowledgements Partial support obtained from the NWO Jacquard Reconstructor project. We thank the functional cluster architects at ASML, and Remco van Engelen for his support in initiating this research.

References

- [1] R. Adnan. Interface Regroup Wizard. Master's thesis, Software Engineering Research Group, Delft University of Technology, October 2007.
- [2] R. Adnan, B. Graaf, A. van Deursen, and J. Zonneveld. Using cluster analysis to improve the design of component interfaces. Technical Report TUD-SERG-2008-026, Delft University of Technology, 2008.
- [3] B. Andreopoulos, A. An, V. Tzerpos, and X. Wang. Clustering large software systems at multiple layers. *Inf. Softw. Technol.*, 49(3):244–254, 2007.
- [4] A. Christl, R. Koschke, and M.-A. Storey. Automated clustering to support the reflexion method. *Inf. Softw. Technol.*, 49(3):255–274, 2007.
- [5] Brian S. Everitt. *Cluster Analysis*. Edward Arnold, 3rd edition, 1993.
- [6] J. Handl and J. Knowles. Exploiting the trade-off – the benefits of multiple objectives in data clustering. In *Proc. 3rd Int. Conf. on Evolutionary Multi-Criterion Optimization (EMO'05)*, volume 3410 of *LNCIS*, pages 547–560. Springer-Verlag, 2005.
- [7] B. S. Mitchell and S. Mancoridis. On the automatic modularization of software systems using the Bunch tool. *IEEE Transactions on Software Engineering*, 32(3):193–208, 2006.
- [8] A. van Deursen and T. Kuipers. Identifying objects using cluster and concept analysis. In *Proceedings of the 21st International Conference on Software Engineering (ICSE 1999)*, pages 246–255. IEEE Computer Society, 1999.