On the Stability of Software Clustering Algorithms

Vassilios Tzerpos

University of Toronto
Toronto, Ontario, CANADA
vtzer@cs.toronto.edu

R. C. Holt

University of Waterloo
Waterloo, Ontario, CANADA
holt@plg.uwaterloo.ca

Abstract

A crucial step in understanding a large legacy software system is to decompose it into meaningful subsystems, which can be separately studied. This decomposition can be done either manually or automatically by a software clustering algorithm (SCA). Similar versions of a software system can be expected to have similar decompositions. We say an SCA is stable if small changes in its input (the software system) produce small changes in its output (the decomposition).

This paper defines stability formally, explains why it is an essential property for an SCA, and gives experimental results from evaluating the stability of various decomposition algorithms suggested in the literature.

1 Introduction

It is a known fact that many software systems are currently in use around the world even though their development ceased many years ago. Such legacy systems present difficult problems when the need arises to modify their functionality, or migrate them to a new hardware platform. The abundance of research projects attempting to address these issues suggests that they are of great importance to the software industry.

One of the main factors that makes legacy software systems hard to deal with is the fact that their documentation is usually obsolete, if existent at all. Most software development projects include a design phase that produces some sort of documentation. However, as the system's development proceeds, the documentation is rarely updated. As a result, design changes are not reflected in it, and it becomes outdated.

Such problems can also affect systems that are still under development. The lack of an up-to-date documentation makes the task of understanding these systems very hard. The research field of software clustering attempts to alleviate this problem by automatically recovering the decomposition of a software system.

Several approaches to the software clustering problem have been proposed in the literature [MM+98, Sch91, LS97, AL97, BH99, CS90, BE81, MOTU93, HB85]. Most of them present encouraging results on various example systems, while some have also been adopted as part of integrated reverse engineering tools, such as the Portable Bookshelf [HF98]. Some effort has also been devoted to comparing such techniques [Lak97, AL99], but further research is required in this area.

One of the biggest challenges a software clustering algorithm (SCA) faces is that of surviving as part of the daily implementation cycle of a real-life software development project. In other words, a good way to measure the effectiveness of an SCA would be to include it in the daily routine of a developing team. Assuming that the system is built nightly, the SCA would produce a new decomposition of the software system that would be shown to the developers the next morning, probably in the form of diagrams produced by a visualization tool.

It is easy to see that for an SCA to be accepted as part of the daily routine, it would have to produce "meaningful" decompositions, something all SCAs strive to do. However, another factor might also play an important role. The SCA will have to adhere to the "rule of minimal change", i.e. the diagrams presented each morning should not be radically different from the ones shown the previous day, since that could lead to confusion, and eventually the rejection of the SCA.

This raises an issue that has not attracted much attention from the software clustering community so far: an SCA's *stability*. It would be interesting to know how much is the output of an SCA affected, when its input is slightly modified. It should be obvious from the above, that it is a desirable feature for an SCA to be stable.

This paper focuses on the issue of stability. We present a measure for it, and attempt to demonstrate its importance. The structure of the rest of the paper is the following: Section 2 presents a definition of the problem at hand, and discusses various facets of the notion of stability. Section 3 introduces a measure for the stability of a given SCA. Section 4 describes experiments we performed with this measure and the obtained results. Finally, Section 5 concludes the paper.

2 Definitions

The notion of stability has been defined and studied in a variety of contexts, such as control systems, sorting algorithms, or even social sciences. In the context of a software clustering algorithm, we define an SCA to be stable, if the obtained clusters are not grossly affected by slight modifications in its input, i.e. the software system in question.

More precisely, let S be the software system, and A the algorithm we are interested in. When we apply A on S we obtain a clustering $C = A(S)$. Subsequently, S is modified slightly. Let us call the new system $S' = S + \Delta S$. Applying the clustering algorithm again, we obtain $C' = C + \Delta C = A(S')$. If ΔC is consistently small, then A is deemed to be stable with respect to S. In the following, we will denote the stability of an algorithm A with respect to a software system S by $\Sigma\tau(A, S)$.

The aforementioned definition of stability raises two important questions:

1. What constitutes a "slight modification" of a software system? Moreover, what extent of changes to a software system is too large to be considered "slight" anymore, i.e. how big of a ΔS do we consider in our definition?

2. How does one measure the effect on the obtained clusters? What would constitute a "gross" effect, i.e. what kind of ΔC would be considered too big for the algorithm to be stable?

We will attempt to answer both questions in the following.

Firstly, it is easy to see that the definition of "slight modification" is dependent on the SCA being examined. For example, if the SCA is based on the naming of the system's resources (as in [AL97]), then "slight modification" refers to the introduction of new elements in the system's name space, or the modification of names of existing resources. At the same time, a change in the dependencies between resources would have no effect on such an algorithm.

However, most SCAs are based on structural criteria, i.e. they view their input as a directed graph where the nodes represent the system's resources (such as files or procedures), and the edges represent dependencies between these resources (such as procedure calls or data references). Concentrating on such SCAs, we distinguish the following types of possible modifications:

1. *Edge deletion.* A dependency between two existing resources is removed.

2. *Node deletion.* A resource is removed from the system (this is usually accompanied by a number of edge deletions).

3. *Edge addition.* A dependency between two existing resources is introduced.

4. *Node addition.* A new resource is introduced to the system (this is usually accompanied by a number of edge additions).

5. *Weight modification.* This applies to SCAs that view their input as a weighted graph, e.g. the weight of an edge is equal to the number of calls between two procedures.

Moreover, it is important to decide what extent of changes we are dealing with. For the purposes of the stability study, we only consider changes that could occur during the course of a few days of development effort. Therefore, we are not concerned with stability issues between subsequent releases of large software systems (it is possible that the structure of a large software system changes drastically between releases). In the next section, we will present a quantitative measure of the amount of modification we consider.

As for the second question raised earlier, we need a way to measure the effect a modification has on the obtained clusters, i.e. measure the size of ΔC. An appropriate metric for hierarchical clustering algorithms was presented in [LG95]. However, in order to include all types of algorithms, we will adopt the MoJo metric [TH99] that measures distance between two different clusterings of the same set of resources.

This distance is defined as the number of $Move$ and $Join$ operations one needs to perform in order to transform one of the clusterings into the other (a $Move$ operation moves an object to a different cluster, while a $Join$ operation joins two clusters into one). As a result, a MoJo distance of 0 means that the two clusterings are identical. Conversely, the larger the value of the MoJo distance is, the more different the two clusterings are. We will denote the distance between two clusterings C_1 and C_2 as $MoJo(C_1, C_2)$.

The distance values that will be considered acceptable for an SCA to be deemed stable will depend on the amount of input modification and the size of the software system in question. A quantitative characterization of this is given in Section 3.

Finally, the problem of determining the stability of an SCA can be approached from a number of different angles. We distinguish and discuss the following:

- *Analytical stability.* This refers to examining the intrinsics of each algorithm and proving that its output will not be significantly affected with any perturbation of its input. Some graph theoretic clustering methods were surveyed in that sense in [RY81]. However, the way an SCA works is not always known (as could be the case with commercial tools). Also, several types of approaches, such as the ones based on genetic algorithms [MM+98], do not lend themselves well to such analysis. As a result, we did not choose such an approach.

- *Practical stability.* An SCA can be deemed stable with respect to a given software system if it has been producing stable results for this system for a long period of time. However, for the purposes of a stability study, it is impractical to try a number of SCAs with a variety of large systems over extended intervals of time.

- *Black-box stability.* This approach treats the SCA as a "black box". It provides randomly perturbated versions of an example system as input[1], and it measures the difference between the obtained clusters and the ones obtained from the original version of the system. By repeating this a number of times, one gets a good idea of an SCA's stability. This approach has the advantage that it can be applied on any SCA, and it only requires "snapshots" of large systems. For these reasons, it was the method chosen for this study.

In the next section, we present a black-box stability measure that will be used to assess the stability of various clustering algorithms.

[1] The perturbation is performed in a way that simulates real-life modifications as much as possible.

3 A stability measure

Measuring the stability of an SCA entails observing its output on two different versions of the same system that are a few days of development apart. However, keeping track of a system's structure on a daily basis in order to be able to apply any SCA to it is impractical. For this reason, our metric will operate as follows:

First, it will apply the SCA in question (denoted by A) on a "snapshot" of an example system S (any release of the system could serve for this purpose). This will produce a clustering $C_0 = A(S)$ of the system's resources. Then, the example system will be modified slightly in a random way. The SCA will be applied on the modified system S_1 and a clustering $C_1 = A(S_1)$ will be obtained. If the MoJo distance between clusterings C_0 and C_1 (we will call it $\Delta C_1 = MoJo(C_0, C_1)$) is small, then the SCA is deemed to have behaved in a stable fashion in this experiment.

The above procedure has to be repeated a large number of times. Each time, the original system S is modified into a new system S_i, and the corresponding ΔC_i is computed. The percentage of these experiments in which the SCA behaves in a stable fashion will be the measure of the SCA's stability with respect to the example system. For example, an algorithm A can be 75% stable with respect to a specific software system S (we would write $\Sigma\tau(A, S)$ =75%). Repeating the experiment with a number of different systems can give us a good idea of an SCA's inherent stability.

We should note that one can hardly expect an SCA to be shown to be 100% stable using this measure. The reason for this is that the random modifications might affect vital parts of the system's structure leading the algorithm astray. A stability value of 80% or more should be adequate to characterize an SCA stable.

Certain parts of the process described above need to be defined more specifically. For instance, the magnitude of the "slight modification" has to be decided. Experience indicates that a day's development on a large software system cannot impact more than 1% of the system's resources and dependencies. Moreover, as mentioned in Section 2, there exist different types of possible modifications. Since development involves adding or deleting resources more often that modifying dependencies between them, we propose the following breakdown on the random modifications[2]:

- 50% are *node additions* (based on the assumption

[2] Since the algorithms we experimented with did not involve weighted edges, we have not included *weight modification* in this schema.

that evolving systems usually grow in size).

- 25% are *node deletions.*
- 15% are *edge additions* (experience shows that developers are more likely to introduce new dependencies rather than remove them).
- 10% are *edge deletions.*

We also need to define what we mean by small value of $\Delta C_i = MoJo(C_0, C_i)$. An upper bound for the value of the MoJo distance between two clusterings of the same set of resources is the cardinality of this set (we denote it by N). If the actual value obtained in each random experiment is less than 1% of N we will deem the SCA to have behaved in a stable fashion.

An extended version of the MoJo metric was used for this study, since the node additions and deletions result in the two clusterings being compared containing different sets of resources. This version considers only the intersection of the two sets which is always at most 1% different than the original set.

The following describes the measure in pseudocode. S is the software system in question, and N is the number of its resources. The number of times we repeat the random part of the experiments is denoted by the parameter TIMES.

```
Apply the SCA on S and obtain clustering C_0
counter := 0

for i : 1 .. TIMES
    Delete up to |0.1 N/100| edges from S
    Delete up to |0.25 N/100| nodes from S
    Add up to |0.5 N/100| nodes to S
    Connect each of these nodes to |0.25 N/100|
        other nodes
    Add up to |0.15 N/100| edges to S
    Apply the SCA on the modified S and
        obtain clustering C_i
    Compute m = MoJo(C_0, C_i)
    if m ≤ |N/100| then
        counter := counter + 1
    end if
end for
The SCA is counter/TIMES × 100 % stable
    with respect to S
```

In the next section we present an implementation of our stability metric and some experiments we performed with it.

Algorithm name	Update rule	Cut-point height
$SL75$	Single linkage	0.75
$SL90$	Single linkage	0.9
$CL65$	Complete linkage	0.65
$CL75$	Complete linkage	0.75
$CL90$	Complete linkage	0.9

Table 1: Characteristics of the five algorithms chosen for our experiments.

4 Implementation and Experiments

In order to test our stability measure we decided to experiment with different algorithms and software systems. We chose two large systems of comparable size, but of different development philosophy:

1. **TOBEY**. This is a proprietary industrial system that is still under development. It serves as the optimizing back end for a number of IBM compiler products. The version we worked with was comprised of 939 source files and approximately 250,000 lines of code.

2. **Linux**. We experimented with version 2.0.27a of this free operating system that is probably the most famous open-source system. This version had 955 source files and approximately 750,000 lines of code.

Our choice of algorithms was limited by what is available publically as an implementation that can run in batch mode (since we need to repeat the random experiments a large number of times). Except for our own clustering algorithm[3], we were fortunate to be able to use the hierarchical clustering algorithm suite developed by Nicolas Anquetil [AL99]. This suite offers a wide selection of association and distance coefficients, as well as update rules. It also allows you to specify the height of the "cut-point" for the dendrogram a hierarchical clustering algorithm produces.

For the experiments presented in this paper, we chose five versions of hierarchical clustering algorithms that use the Jaccard coefficient. They are shown on table 1. Our rationale was that we would like to explore:

- The difference between complete linkage and single linkage. Complete linkage is known to produce

[3]Detailed information on the algorithm has not been published yet. For this paper, we will refer to it as OCA.

	OCA	$SL75$	$SL90$	$CL65$	$CL75$	$CL90$
TOBEY	81.3%	99.1%	100%	74.1%	44.6%	12.5%
Linux	69.4%	97.9%	100%	64.7%	37.1%	10.4%

Table 2: Stability values obtained for the algorithms we examined.

more compact clusters. Raghavan and Yu [RY81] showed that graph theoretic clustering methods that identify more compact clusters are not as stable. We were interested to see whether that would also be true for hierarchical clustering algorithms.

- The impact of the cut-point height value. One of the hardest problems associated with hierarchical clustering algorithms is the choice of the cut-point height value. We wanted to determine the impact it has on the algorithm's stability, something that could possibly indicate a criterion for an effective choice of its value. Common values are larger than 0.5 and less than 1, so we chose the values 0.65, 0.75, and 0.9.

In order to perform the slight modifications necessary for our experiments, we developed a program called *Factshaker* that can randomly modify files in the input format of the chosen algorithms (OCA uses the Rigi Standard Format [MK88] and the other algorithms use GdeM). *Factshaker* can perform any number of modifications. The values described in Section 3 were chosen.

The experiments were performed as follows: The source code of the input systems was used only once in order to extract the relevant information[4] in the appropriate format. The initial clustering was obtained next. Finally, *Factshaker* was employed to modify the input and obtain subsequent clusterings. The random part was repeated 1000 times for each combination of algorithm and input system.

The results of our experiments can be seen in the next page. The legend of each pie chart identifies the algorithm and the input system it refers to. It also presents the observed stability of the algorithm with respect to this input system. The white part of the chart corresponds to stable values, while the black corresponds to unstable ones (for both input systems single digit values happen to be stable, while double digit ones are unstable).

Table 2 summarizes the stability values obtained. Several observations can be made:

[4]In both cases the information extracted referred to source files and their dependencies.

- The various algorithms seem to behave in a similar fashion with respect to the two input systems. In other words, algorithms that seem to be stabler for TOBEY, are also stabler for Linux, and vice versa. This indicates that our measure can give us a good idea of an algorithm's stability.

- The stability values obtained for Linux are consistently slightly lower than the ones obtained for TOBEY. This is true even for algorithm $SL90$ where both values are 100% (contrast Figures 1c and 1i). This discrepancy seems rather strange at first since the two systems are represented by fairly similar graphs (the TOBEY graph is composed of 939 nodes and approximately 9,000 edges, while the Linux one has 955 nodes and around 10,000 edges). However, closer examination reveals that TOBEY contains more nodes with small degree, while many Linux nodes have a rather large degree (Figure 2 presents a graph that demonstrates this fact). This would suggest that the deletion of a node from the Linux graph would be accompanied by the deletion of a larger number of edges on average, which could explain larger discrepancies in the SCA's output making it appear less stable.

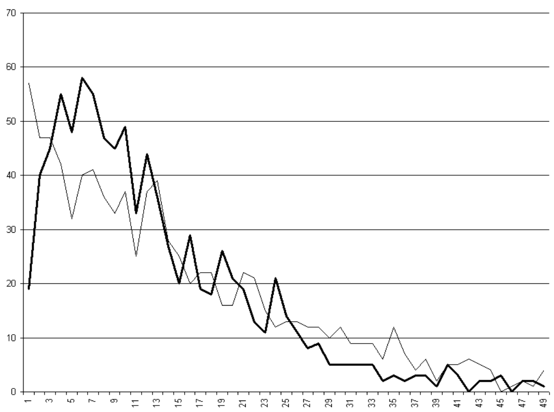
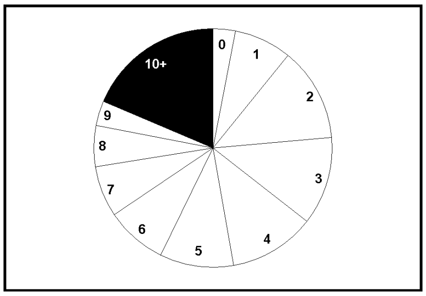
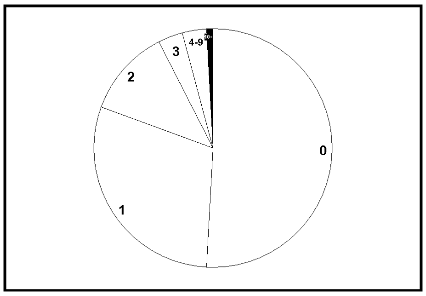


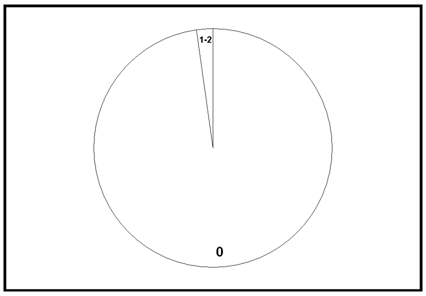
Figure 2: Node degree distribution for TOBEY (thick line) and Linux (thin line). A point's x-value is the node degree and the y-value is the number of nodes in the system's graph that have this degree.



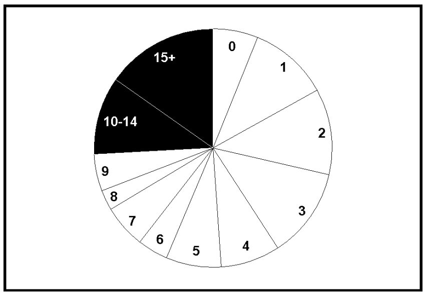
(a) $\Sigma\tau(OCA,\text{TOBEY})$=81.3%



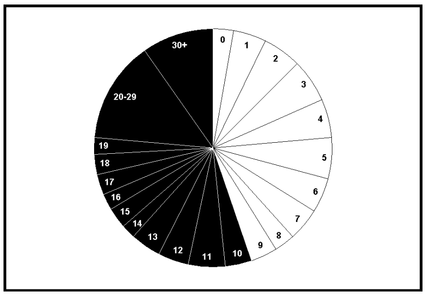
(b) $\Sigma\tau(SL75,\text{TOBEY})$=99.1%



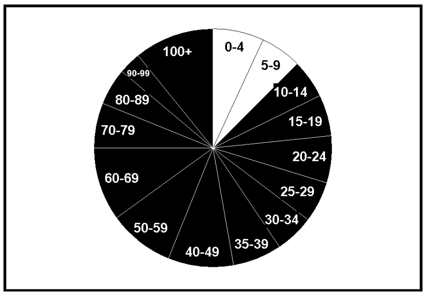
(c) $\Sigma\tau(SL90,\text{TOBEY})$=100%



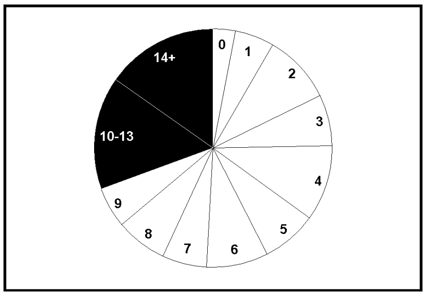
(d) $\Sigma\tau(CL65,\text{TOBEY})$=74.1%



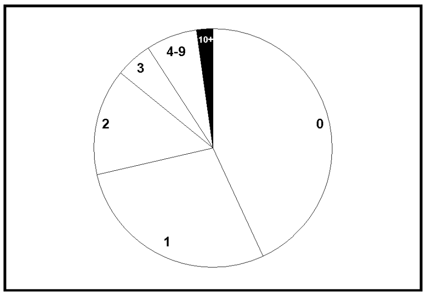
(e) $\Sigma\tau(CL75,\text{TOBEY})$=44.6%



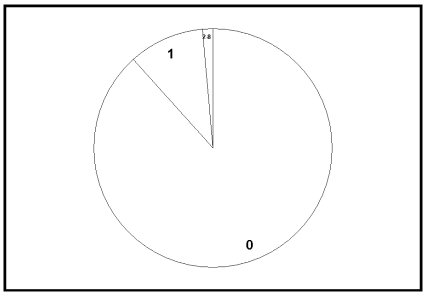
(f) $\Sigma\tau(CL90,\text{TOBEY})$=12.5%



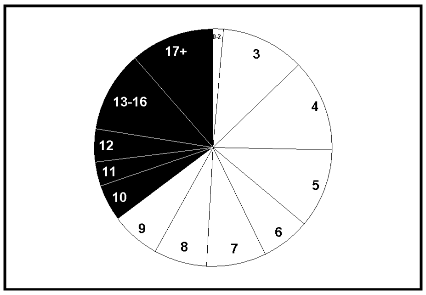
(g) $\Sigma\tau(OCA,\text{LINUX})$=69.4%



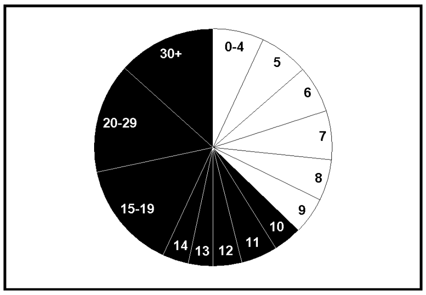
(h) $\Sigma\tau(SL75,\text{LINUX})$=97.9%



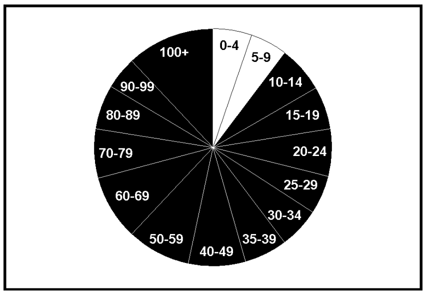
(i) $\Sigma\tau(SL90,\text{LINUX})$=100%



(j) $\Sigma\tau(CL65,\text{LINUX})$=64.7%



(k) $\Sigma\tau(CL75,\text{LINUX})$=37.1%



(l) $\Sigma\tau(CL90,\text{LINUX})$=10.4%

Figure 1: Experimental results.

- For both example systems, it appears that a hierarchical clustering algorithm using the single linkage update rule is stabler than one using the complete linkage update rule. This result agrees with the theoretical predictions by Raghavan and Yu.

  The wide difference in the values between corresponding algorithms (for example, $SL90$ and $CL90$) can be explained by the algorithm's nature. An algorithm using single linkage tends to form clusters much earlier than one using complete linkage. As a result, when the cut-point height was set to 0.9, the single linkage algorithm would produce a small number of clusters (typically 5-7) while the complete linkage one would find many more (consistently over 100). Fewer, more well-formed clusters can easily explain the difference in the stability values obtained.

- It appears that as the cut-point height of a hierarchical clustering algorithm using the complete linkage update rule increases, the stability of the algorithm decreases (at least up to the value of 0.9). That was a rather unexpected result since in theory an algorithm with a cut-point height of 1 should be 100% stable (a cut-point height of 1 means that all objects are grouped in one cluster). Further research is required to determine the stability behaviour of such algorithms with cut-point height between 0.9 and 1.

- Our confidence in our stability measure has increased after these experiments, since the experimental results seem to agree with the theoretical ones, and they do not seem to be affected by the input system. At the same time, these results indicate that stability could be used as a means of comparison for SCAs.

The next section summarizes the paper and presents our conclusions.

5 Conclusions

This paper has raised the important issue of an SCA's stability. We presented reasons why we believe this is an important issue, and why we consider stability to be a desirable feature for an SCA.

We have also presented a measure that evaluates an SCA in terms of its stability. We performed several experiments with this measure. The results indicate that our metric can be a good predictor of an algorithm's stability.

An important corollary is that stability can be used as a means for comparison of different SCAs, as suggested by Jain and Dubes [JD88]. Although it is definitely not the only deciding factor[5], it can provide valuable insight to the usefulness of an SCA.

Future work includes experimenting with more systems and algorithms, in order to corroborate the results presented in this paper, and expand them in an effort to categorize the stability behaviour of various algorithms. It would be interesting to determine if other coefficients behave in a similar fashion as the Jaccard coefficient that was used in these experiments.

Also, we would like to explore further the possibility of using stability as a means of algorithm comparison, by identifying a number of factors one should consider along with stability when attempting to select an SCA for a reverse engineering or program comprehension project.

Acknowledgements

The work presented here has been supported by CSER (Consortium for Software Engineering Research), IBM Canada, the Natural Science and Engineering Research Council of Canada, and CITO.

The authors would also like to thank Nicolas Anquetil for his help concerning the hierarchical clustering algorithm suite.

References

[AL97] Nicolas Anquetil and Timothy Lethbridge. File clustering using naming conventions for legacy systems. In *Proceedings of CASCON 1997*, pages 184–195, November 1997.

[AL99] Nicolas Anquetil and Timothy Lethbridge. Experiments with clustering as a software remodularization method. In *Proceedings of the Sixth Working Conference on Reverse Engineering*, pages 235–255, October 1999.

[BE81] L. A. Belady and C. J. Evangelisti. System partitioning and its measure. *Journal of Systems and Software*, 2:23–29, 1981.

[BH99] Ivan T. Bowman and R. C. Holt. Reconstructing ownership architectures to help

[5]Other factors might include the type of clustering an algorithm produces, or features such as cluster naming.

understand software systems. In *Proceedings of the Seventh International Workshop on Program Comprehension*, May 1999.

[CS90] Song C. Choi and Walt Scacchi. Extracting and restructuring the design of large systems. *IEEE Software*, pages 66–71, January 1990.

[HB85] David H. Hutchens and Victor R. Basili. System structure analysis: Clustering with data bindings. *IEEE Transactions on Software Engineering*, 11(8):749–757, August 1985.

[HF98] R.C. Holt and Gary Farmaner. "The Portable Bookshelf". *http://www-turing.cs.toronto.edu/pbs*, 1998.

[JD88] A. Jain and R. Dubes. *Algorithms for Clustering Data.* Prentice-Hall, 1988.

[Lak97] Arun Lakhotia. A unified framework for software subsystem classification techniques. *Journal of Systems and Software*, pages 211–231, March 1997.

[LG95] A. Lakhotia and J. M. Gravley. Toward experimental evaluation of subsystem classification recovery techniques. In *Proceedings of the Second Working Conference on Reverse Engineering*, pages 262–269, July 1995.

[LS97] Christian Lindig and Gregor Snelting. Assessing modular structure of legacy code based on mathematical concept analysis. In *Proceedings of the 19th International Conference on Software Engineering*, pages 349–359, May 1997.

[MK88] Hausi A. Müller and K. Klashinsky. Rigi - a system for programming-in-the-large. In *Proceedings of the 10th International Conference on Software Engineering*, pages 80–86, 1988.

[MM+98] Spiros Mancoridis, B.S. Mitchell, et al. Using automatic clustering to produce high-level system organizations of source code. In *IEEE proceedings of the 1998 International Workshop on Program Comprehension*. IEEE Computer Society Press, 1998.

[MOTU93] Hausi A. Müller, Mehmet A. Orgun, Scott R. Tilley, and James S. Uhl. A reverse engineering approach to subsystem structure identification. *Journal of Software Maintenance: Research and Practice*, 5:181–204, December 1993.

[RY81] Vijay V. Raghavan and C. T. Yu. A comparison of the stability characteristics of some graph theoretic clustering methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3, 1981.

[Sch91] Robert 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.

[TH99] Vassilios Tzerpos and R. C. Holt. Mojo: A distance metric for software clusterings. In *Proceedings of the Sixth Working Conference on Reverse Engineering*, pages 187–193, October 1999.