

A Framework for Experimental Evaluation of Clustering Techniques

Rainer Koschke and Thomas Eisenbarth

University of Stuttgart, Breitwiesenstr. 20-22, D-70565 Stuttgart, Germany

{koschke,eisenbts}@informatik.uni-stuttgart.de

Abstract

Experimental evaluation of clustering techniques for component recovery is necessary in order to analyze their strengths and weaknesses in comparison to other techniques. For comparable evaluations of automatic clustering techniques, a common reference corpus of freely available systems is needed for which the actual components are known. The reference corpus is used to measure recall and precision of automatic techniques. For this measurement, a standard scheme for comparing the components recovered by a clustering technique to components in the reference corpus is required. This paper describes both the process of setting up reference corpora and ways of measuring recall and precision of automatic clustering techniques.

For methods with human intervention, controlled experiments should be conducted. This paper additionally proposes a controlled experiment as a standard for evaluating manual and semi-automatic component recovery methods that can be conducted cost-effectively.

1. Introduction

Research in the field of component recovery has yielded many techniques to detect modules and subsystems automatically or semi-automatically and their number is still growing [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 17, 18, 21, 22, 23, 25, 28, 29, 30, 32, 34]; see [14] and [15] for an overview of existing techniques. However, many of these published techniques are only qualitatively evaluated based on a single case study. Others that do provide quantitative results cannot really be compared to other techniques because the new technique is applied to a system different from those used for the existing techniques and/or the way of evaluation differs. Consequently, the effectiveness and the strengths and weaknesses of the techniques in comparison to other techniques are not clear.

In order to obtain comparable results, a standardized way of evaluation is needed. Because there are basically three different categories of component recovery methods, namely, manual, semi-automatic, and fully automatic, of which two categories involve human intervention, two different ways of evaluation are needed:

1. benchmark testing for fully automatic techniques
2. controlled experiments for manual and semi-automatic methods

For benchmark testing of fully automatic component recovery techniques, a benchmark is needed that offers two things:

- a set of expected results for diverse systems (reference corpora); the techniques are applied to these systems and propose their candidates
- a standard test by which the candidates proposed by the automatic technique are compared to the components of the reference corpora; the test measures recall and precision with respect to the reference corpus

This paper discusses both aspects and describes a quantitative comparison we have developed over three years and successfully used to evaluate diverse techniques [14].

For the evaluation of methods that involve human intervention, controlled experiments should be conducted. Controlled experiments have the advantage over case studies to yield more comparable results because the influence factors are controlled. On the other hand, controlled experiments are usually more expensive and many computer scientists are not familiar with experiments involving humans. This paper describes a scheme for experiments to evaluate component recovery methods. The proposed experiments require a low number of experimental subjects and are, hence, cost-effective. Because of the smaller samples, the ANOVA (analysis of variance) tests commonly used for experiments are not adequate in such situations and therefore one has to use less known statistical tests, so-called non-parameterized tests.

We have successfully used the experimental layout and its statistical tests described in this paper to evaluate a semi-automatic method [14].

1.1. Terminology

This section defines the terminology used throughout this paper. A **component** is a set of related entities. There are two different kinds of components: **atomic components** that do not further contain other components (also called **logical modules**) and **subsystems** that may be hierarchical, i.e., may contain other components. An atomic component consists of base entities. **Base entities** are rou-

tines, global variables, and types.

A **reference corpus** is a set of components, called **reference components** or simply **references**, that are identified by a software engineer and part of the benchmark. A **candidate component**, or simply **candidate**, is a component that is proposed by an automatic technique.

The result of many component recovery techniques is a set of flat candidates. A **flat candidate** is a set of base entities without any further structure. Hierarchical clustering techniques yield so-called dendrograms. A **dendrogram** is a tree whose leaves are the entities to be grouped and whose inner nodes represent a subset of closer related entities. Each inner nodes is associated with a similarity value that summarizes the similarity among the entities in the subtrees (see Figure 1).

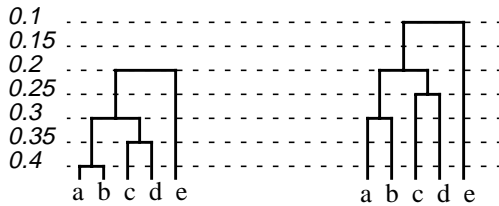


Figure 1: Two example dendrograms.

1.2. Overview

The rest of this paper is organized as follows. Section 2 summarizes related research. Section 3 describes the metric used to compare components. Section 4 handles issues related to conducting controlled experiments to evaluate manual and semi-automatic methods. Section 5 discusses how to obtain a standard benchmark suite.

2. Related Research

The need for benchmark testing for component recovery techniques was already stated in 1995 by Lakhotia and Gravely [16], but even today, after five years, little has been achieved toward the goal of an accepted benchmark. Neither are reference corpora available nor do we have an accepted standard test.

Lakhotia and Gravely proposed an evaluation metric to compare a candidate to a reference based on dendrograms [16]. The metric consists of two parts. One part measures the difference, Δ , between the similarities of elements of the candidate and the elements of the reference; Lakhotia and Gravely list three different ways of measuring the difference based on the maximal difference, the average difference, or the standard deviation without laying down which one should actually be used. The other part measures the degree of overlap between the candidate, C , and the reference, R :

$$\text{overlap}(C, R) = \frac{|\text{elements}(C) \cap \text{elements}(R)|}{|\text{elements}(C) \cup \text{elements}(R)|} \quad (1)$$

where $\text{elements}(X)$ denotes the set of base entities of X . The measure of congruence, μ , summarizes the difference in similarities of the elements and the degree of overlap as follows:

$$\mu(C, R) = \Delta(C, R) \times \text{overlap}(C, R). \quad (2)$$

Equation (2) can be used to measure the similarity between two components. However, Lakhotia and Gravely do not specify how two sets of components should be compared using this equation which is necessary if one wants to compare all candidates of a technique to a reference corpus. For obvious reasons, it would not make sense to use the average congruence between all pairs of candidates and references because then non-corresponding components would be compared.

Another objection to the comparison is that a definition of Δ based on a simple difference between base entities is not appropriate for at least two reasons:

- The similarity of base entities in the two dendrograms to be compared may in fact differ as long as the structure is the same (see Figure 1) because one cannot necessarily expect identical similarity values, in particular, with respect to reference components that have been manually identified by a human.
- Using differences between similarity values would only be appropriate if similarity were on an interval scale. However, when a human assigns similarity values to base entities, the interval scale can hardly be justified. A human may be able to state that one pair of elements is more similar than another pair, but the exact difference between the respective similarities is rather arbitrary.

Because the similarity assigned by a human is rather on an ordinal scale, a comparison based on ranking the pairs of similar elements would be more appropriate. However, this would require a software engineer not only to group related elements together when setting up a reference corpus but also to rank $n(n-1)/2$ pairs for each component with n elements. Hence, though appealing in theory, comparing components based on similarities even using only ranking information is not feasible in practice.

We independently developed an evaluation metric (see Section 3) that is only based on the degree of overlap of corresponding components, i.e., using equation (1) and variants, and applied this metric to compare nine clustering techniques ([8, 14]).

Tzerpos and Holt define an alternative comparison metric that ignores the similarity values, too [31]. Their metric yields an aggregated overall similarity measure based on the minimal number of modifications to transform one

clustering into the other one. To establish this transformation, a heuristic is used to compute the result in a reasonable time. Consequently, the metric value is only an approximation whereas our metric can be computed efficiently and exactly. Another disadvantage of their approach is the inability of identifying the nearly equal candidates and references.

Contributions

This paper takes up again the goal of Lakhota and Gravely of establishing an accepted benchmark suite and a standard evaluation method. The evaluation method identifies corresponding components, including subcomponents, false positives based on their degree of overlap, and measures different aspects of this correspondence that go beyond the data considered by Lakhota and Gravely. The metric presented here is a slight modification of the metric we have used to compare nine different clustering techniques.

The paper also describes a controlled experiment in which the evaluation metric is used to measure and compare the findings of software engineers who used either manual or semi-automatic methods.

3. Comparison of Candidates and References

The comparison of candidates and references comprises the following steps:

1. establishing the corresponding candidates and references
2. classifying the match of corresponding components
3. measuring different aspects of the correspondence

3.1. Correspondence of Components

Candidate components and reference components are compared using an approximate matching to accommodate the fact that the distribution of functions, global variables, and types into components is sometimes subjective and, pragmatically, we have to cope with matches of candidates and references that are incomplete, yet “good enough” to be useful. “Good enough” means that candidate and reference overlap to a large extent and only few elements are missing. More precisely, we treat one component S as a match of another component T if S and T overlap to a large degree (denoted by $S \approx_p T$) according to the following definition of an **affinity relationship** \approx_p :

$$S \approx_p T \text{ if and only if } \text{overlap}(S, T) \geq p \quad (3)$$

where *overlap* is defined by equation (1). In order to identify corresponding subcomponents, i.e., components

that are only similar to a part of another component, the following **partial subset relationship** \subseteq_p is useful:

$$S \subseteq_p T \text{ if and only if } \frac{|\text{elements}(S) \cap \text{elements}(T)|}{|\text{elements}(S)|} \geq p \quad (4)$$

where $0.5 \leq p \leq 1.0$ is a tolerance parameter p that needs to be specified for the comparison. If set to 1.0, S must be completely contained in T . A more pragmatic adjustment is $p = 0.7$, i.e., at least 70 percent of the elements of S must also be in T . This number is motivated by the assumption that at least three elements of a four-element component must also be in the other component to be an acceptable match.

This definition still considers a component with elements $\{a, b, c, d\}$ at least a partial subset of a component with elements $\{a, b, d, e, f\}$ when $p \geq 0.7$ though c is not present in the latter set of elements.

Note that the partial subset relationship is not transitive for $p \neq 1$. For example, $\{a, b, c\} \subseteq_{0.6} \{a, b, d\} \subseteq_{0.6} \{b, d, e\}$, but $\{a, b, c\} \not\subseteq_{0.6} \{b, d, e\}$.

3.2. Classification of Matches

Based on the approximate matching criterion, the matches of references and candidates are classified into the following categories according to the kind of correspondence:

- **Good** when the match between a candidate, C , and a reference, R , is close, i.e., $C \approx_p R$. This case is denoted **1~1**. Matches of this type require a quick verification in order to identify the few elements which should be removed or added to the candidate component.
- **Ok** when the \subseteq_p relationship holds for a candidate, C , and a reference, R , i.e., $C \subseteq_p R$ or $R \subseteq_p C$, but not $C \approx_p R$.

If $C \subseteq_p R$, then the candidate is **too detailed**. This case is denoted as **n~1**. If $R \subseteq_p C$, then the candidate is **too large**. This case is denoted as **1~n**. OK matches require more attention to combine or refine a component. The denotation $n~1$ and $1~n$ reflects the fact that multiple Ok matches may exist for a given R or C .

Altogether, we have three **classes of matches**: $1~1$, $1~n$, and $n~1$ where the latter two are both considered Ok.

Difference to our previous version. In a previous version of the classification (jointly developed with Jean-Francois Girard and Georg Schied and used in [7, 8, 14]), we considered R and C a good match if $R \subseteq_p C \wedge C \subseteq_p R$. However, for this premise, R and C could correspond more roughly than one would intuitively expect. For example, if R and C both have 100 elements and 70 elements of R are in C and

70 elements of C are in R , then $R \subseteq_p C \wedge C \subseteq_p R$ holds. However, the overlap of R and C is only $70/130 \approx 0.54 < p = 0.7$. For a good match, one would rather expect the overlap to be above 0.7. The newly proposed definition of \approx_p is stricter because

$$R \approx_p C \Rightarrow R \subseteq_p C \wedge C \subseteq_p R$$

but not necessarily

$$R \subseteq_p C \wedge C \subseteq_p R \Rightarrow R \approx_p C.$$

Example. Consider the example in Figure 2. C_1 and R_1 are a good match. Because only partial matches are required, there can be another reference R_4 (with $R_4 \cap R_1 = \emptyset$) that is a partial subset of C_1 (of $C_1 \setminus R_1$, more precisely). C_2 is an Ok match with R_2 , and so is C_3 . C_2 , C_3 , and R_2 constitute an $n \sim 1$ match. That is, the technique has produced finer-grained candidates than what was expected. Note that we cannot conclude that $C_2 \cup C_3$ and R_2 are a good match because R_2 could be much bigger than $C_2 \cup C_3$. R_3 and C_4 constitute a $1 \sim n$ match, where no other reference than R_3 can be matched with C_4 . C_5 and R_5 do not match at all.

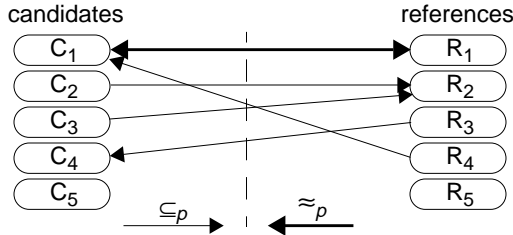


Figure 2: Example correspondences of candidates and references.

As the example indicates, it is not enough just to count the matches in order to judge the detection quality of a technique. For example, R_3 is a partial subset of C_4 and, therefore, considered at least an Ok match. However, C_4 could be huge and the match just be coincidence. The next section proposes a measurement for detection quality based on multiple aspects that accommodates this imprecision.

3.3. Detection Quality

There are several aspects in a comparison of a set of candidates with a set of references to consider when the matches have been established as described in the last section:

- **Number of false positives:** The number of candidates that neither match a reference nor are matched by any reference, i.e., candidates that cannot be associated with any reference. Technically speaking, these are candidates that are neither involved in a $1 \sim 1$, $1 \sim n$, nor $n \sim 1$ match. This number should be 0. For the false positives, the average number of elements of false positive candi-

dates should additionally be ascertained because this number correlates with the time it takes to sieve false positives during human validation.

- **Number of true negatives:** The number of references that neither match a candidate nor are matched by any candidate, i.e., references that are not even partially detected. Technically speaking, these are references that are neither involved in a $1 \sim 1$, $1 \sim n$, nor $n \sim 1$ match. This number should be 0.
- **Granularity of matches:** Are the candidates at the right level of granularity? Technically speaking, there should only be good matches and no Ok matches.
- **Precision of matches:** The degree of correspondence between candidates and reference matches. This is discussed in the following in more detail. The precision should approach 1.0.

The number of false positives determines the precision in terms of information retrieval whereas the other aspects measure recall.

Since the partial subset relationship is used to establish a match, the matching candidates and references need not be equal. That is, there may be elements of the candidate not in the reference and vice versa: $C \setminus R \neq \emptyset$ and $R \setminus C \neq \emptyset$. In other words, there may be a flaw in a good match; even more so for Ok matches because of (let R be a reference and C_i be candidates for which $C_i \subseteq_p R$ holds):

$$\bigcup_i C_i \subseteq_p R \not\Rightarrow R \subseteq_p \bigcup_i C_i$$

Accuracy for two matching components. In order to indicate the quality of imperfect matches of candidate and reference components, an accuracy factor is associated with each match. The similarity between two components, and thus the accuracy of a candidate vis-a-vis a reference component, is computed using the following formula:

$$\text{accuracy}(A, B) = \frac{\text{overlap}(A, B)}{\max(|A|, |B|)} \quad (5)$$

In $1 \sim n$ and $n \sim 1$ matches — and sometimes even in $1 \sim 1$ matches — several components may match with one other component. The accuracy as defined above, however, involves only two single components. Therefore, the definition is extended for sets of components as follows.

Accuracy for two sets of components. The overlap for two matching components can be used to ascertain the accuracy of sets of components:

$$\text{accuracy}(\{A_1, \dots, A_a\}, \{B_1, \dots, B_b\}) = \frac{\text{overlap}(\bigcup_{i=1 \dots a} \text{elements}(A_i), \bigcup_{i=1 \dots b} \text{elements}(B_i))}{\max(\sum_{i=1 \dots a} |A_i|, \sum_{i=1 \dots b} |B_i|)} \quad (6)$$

Accuracy for classes of matches. The accuracy for two sets of components is used to establish the accuracy for a

whole class of matches where the two sets $\{A_1, \dots, A_a\}$ and $\{B_1, \dots, B_a\}$ are corresponding components in a match within a given class of matches.

More precisely, let the matching components of a candidate or reference, X , be defined as follows:

$$\text{matchings}(X) = \{Y | Y \subseteq_p X\} \quad (7)$$

Using the *matching components*, we can specify the degree of agreement for the diverse classes of matches:

- 1~1 match: $\text{accuracy}(\text{matchings}(C), \text{matchings}(R))$
- $n\sim 1$ match: $\text{accuracy}(\text{matchings}(R), \{R\})$
- $1\sim n$ match: $\text{accuracy}(\{C\}, \text{matchings}(C))$

To put it in words: The accuracy is ascertained based on the united matching components. The way of handling $1\sim 1$ matches may first be astonishing, but is motivated by the fact that a $1\sim 1$ match does not necessarily mean that there is no other component that is a partial subset of one of the components in the $1\sim 1$ match. This was already touched in the example of Figure 2 where C_1 and R_1 are a $1\sim 1$ match and R_4 is still a partial subset of C_1 . If there is no such additional $1\sim n$ or $n\sim 1$ match, then:

$$\text{accuracy}(\text{matchings}(C), \text{matchings}(R)) = \text{accuracy}(R, C)$$

That is, we subsume an additional $1\sim n$ or $n\sim 1$ match in a $1\sim 1$ match. This is justified because there is a clear $1\sim 1$ relationship in the first place and the additional $1\sim n$ or $n\sim 1$ match can only be comparatively small.

Such overlapping matches can also exist for pure $1\sim n$ and $n\sim 1$ matches as the example in Figure 3 illustrates. However, the overlap of $1\sim n$ and $n\sim 1$ matches is ignored since there is no dominating correspondence as in the case of overlaps with $1\sim 1$ matches. That is, the two overlapping matches in Figure 3 are handled as two distinct matches ($p = 0.7$).

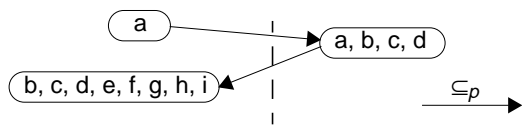


Figure 3: Overlapping $1\sim n$ and $n\sim 1$ matches.

Now that we have the means to establish the accuracy of a single match with respect to its class ($1\sim 1$, $1\sim n$, $n\sim 1$), we can extend the accuracy to the whole class of matches. The classes of matches are defined as follows:

$$GOOD = 1\sim 1 = \left\{ \begin{array}{l} (\text{matchings}(c), \text{matchings}(r)) \\ \text{overlap}(c, r) \geq p \end{array} \right\}$$

$$1\sim n = \left\{ \begin{array}{l} (\{c\}, \text{matchings}(c)) \\ \text{matchings}(c) \neq \emptyset \wedge \forall (r) r \subseteq_p c \Rightarrow \neg(c \approx_p r) \end{array} \right\}$$

$$n\sim 1 = \left\{ \begin{array}{l} (\text{matchings}(r), \{r\}) \\ \text{matchings}(r) \neq \emptyset \wedge \forall (c) c \subseteq_p r \Rightarrow \neg(r \approx_p c) \end{array} \right\}$$

$$OK = 1\sim n \cup n\sim 1 \quad (8)$$

Then, the accuracy for a whole class of matches is defined as the average in accuracy of the members of the class (let M be a class of $1\sim 1$, $1\sim n$, or $n\sim 1$ matches):

$$\text{accuracy}(M) = \frac{\sum_{(a,b) \in M} \text{accuracy}(a, b)}{|M|} \quad (9)$$

Overall recall rate. The detection quality of a technique is described by a vector of the number of false positives and true negatives and the average accuracies of $1\sim 1$, $1\sim n$, and $n\sim 1$ matches according to (9) along with their respective absolute number and average size to indicate the level of granularity. These figures provide a detailed picture for the comparison of the techniques. However, an additional summarizing value is useful for a quick comparison. The following equation characterizes the overall recall rate (*GOOD* and *OK* are defined above):

$$\text{Recall} = \frac{\sum_{(a,b) \in GOOD} \text{accuracy}(a, b) + \sum_{(a,b) \in OK} \text{accuracy}(a, b)}{|GOOD| + |OK| + |\text{true negatives}|} \quad (10)$$

To illustrate the definition of the recall rate, consider the example in Figure 4, in which the matching components of each candidate and reference component of Figure 2 have been merged for the comparison. There are two *OK* matches and one *good* match. R_5 is not matched at all and, therefore, considered a true negative; likewise, C_5 is a false positive because it does not correspond to any reference. The example also illustrates that the denominator of (10) cannot simply be the number of the original references because not only candidates but also references can be united for the comparison, which reduces the number of references actually used for the comparison.

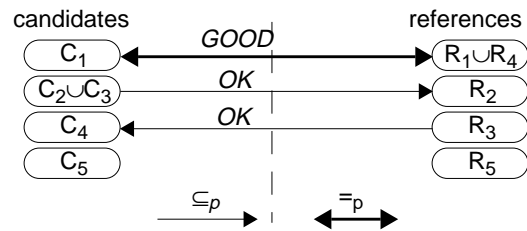


Figure 4: Example merged correspondences of candidates and references.

The recall rate (10) abstracts from the level of granularity – since *good* and *OK* matches are treated equally by this definition – and ignores false positives. The number of

false positives is a different aspect and is not captured by this definition because – depending on the task at hand – a higher number of false positives in favor of a higher recall rate may be acceptable.

3.4. Hierarchical Clustering Techniques

The comparison of components as described above is suitable for comparing flat components. When a hierarchical clustering technique is used, flat components need to be derived from the result dendrogram. The components can be derived using different thresholds to cut subtrees of closely related elements from the dendrogram. Only then, equation (2) and likewise the metrics of Lakhoria and Gravely [16] and Tzerpos and Holt [31] can be used to compare flat components. However, the number and quality of the components derived from the dendrogram may vary with the chosen threshold. In this case, one has to decide upon a suitable threshold. One alternative is to use different thresholds and accept the one that yields the best results. This would measure the best result one could get with the technique. In practice, one could have worse results if the appropriate threshold is not known in advance.

Another alternative is to let the user select subtrees manually that are to be considered candidates, hence, using different thresholds for different subtrees. However, then the technique would no longer be fully automatic and would have an advantage over the other techniques.

For re-modularization techniques, we do not expect references to be hierarchical. An analyst will rarely make an effort to establish the similarities within a module. Hence, the candidates will generally be compared to non-hierarchical references. However, if subsystems are to be compared, a hierarchical correspondence should be established. We have recently extended our definition of correspondence to hierarchical components [14] and are currently working on defining congruence based on this hierarchical correspondence.

4. Controlled Experiments

For evaluating component recovery methods that involve human intervention, controlled experiments should be used. The goal of an experiment is to show that the new technique indeed supports a maintainer or to show that the new technique is superior to other techniques. In order to get meaningful results, one would like to have as many experimental subjects as possible. On the other hand, to conduct the experiment in a cost-effective way, one would prefer smaller samples, and in practice, it is also difficult to find enough participants in the first place. Many researchers will, therefore, use smaller samples. Descriptions on how to conduct and evaluate experiments can be found in

standard textbooks (see for example, [33], [12]). However, most of these textbooks only describe ANOVA tests – but these tests presume larger samples; they are not valid for experiments with, say, 10 participants. For experiments with small samples, non-parameterized tests are the only appropriate tests.

This section briefly describes the layout and evaluation of a small-scale experiment for comparing component recovery methods. The design of the experiment has been successfully used to evaluate our incremental semi-automatic method for component recovery [14].

4.1. Validity

If the experimental subjects are students, the number of experimental subjects is low, and/or only few systems are analyzed in the experiment, one should refrain from generalizing the results too far. The general objective of small-scale experiments is not to yield a definite empirical proof for the usefulness of a method for all kinds of systems and settings but to learn about the strengths and weaknesses of a method and to investigate where further research should be directed.

4.2. Hypotheses

The general hypothesis of an experiment to evaluate a new semi-automatic method M is that M yields more components than other methods in the same amount of time. The other methods can be purely manual or semi-automatic, too. By manual search, we mean that only common cross-reference and textual pattern matching tools, such as `grep`, are used, whereas for the semi-automatic methods, more advanced automatic support is available.

The independent variable is therefore the selection of available methods and the dependent variable is the achieved detection quality.

The null hypothesis and the two-sided and single-sided alternative hypotheses are as follows:

- **Null hypothesis H_0 :** There is no difference in the detection quality among the methods.
- **Alternative hypothesis H_1 :** The detection qualities differ (for two-sided tests).
- **Alternative hypothesis H_2 :** The detection quality of method M is greater than of all other methods (single-sided tests).

4.3. Experimental Materials

The task of the experimental subjects is to recover the components for the same system or set of systems. If the component detection is repeated for a set of systems, appropriate statistical tests must be applied that take learning

effects due to repetition into account (see [19]). If only one system is analyzed, the statistics outlined in Section 4.6 can be used.

For large systems, a researcher may reduce the possible search space for components to a size that can be handled within the given time frame in order to obtain comparable results, i.e., all experimental subjects should be able to look at all source files within the available time. The degree of familiarity with the subject system should be the same for all experimental subjects. If this is not the case, a block design can be used (see [12]).

4.4. Experimental Design

In a simple design, the experimental subjects are randomly assigned to groups that differ in the tools available to the search for components, i.e., members of group G_i may use method M_i . Other designs can be found in [12].

In order to avoid too much variance in the set of experimental subjects, all experimental subjects should be jointly trained.

If the number of components for a system is not known in advance, a termination criterion for the search for components does not exist. In that case, a time limit can be set. Limiting the available time makes the experiment even more realistic since in an industrial setting, one can generally not afford to spend unrestricted time on a problem.

4.5. Measurement of the Dependent Variable

Two distinct ways of measuring the dependent variable should be chosen:

1. using the absolute number of clustered elements for each subject (*individual absolute recall*, short IAR)
2. comparing the components of each individual to the joint set of components of all individuals (*reference corpus recall*, short RCR)

The first alternative does not require agreement among the experimental subjects, hence measures only how many elements were clustered in a given time by each individual.

For the second alternative, the individual results are to be joined and the result of each individual is judged with respect to the joined result. The joined list of components should be individually reviewed by the experimental subjects. The review work should be distributed among the subjects so that each component is at least reviewed by two persons. In order to reduce the effort for the experimental subjects to review the reference corpus, each subject need only review a part of the reference corpus. The reviewing subjects can accept components as a whole or in parts as well as add elements to the components, but should discuss their changes with the original analyst. The set of accepted components form the reference corpus to which the pro-

posed components of each subject are compared by the method presented in Section 3. Hence, the dependent variable can be measured as the recall rate defined in equation (10) with respect to the joined reference corpus.

Because IAR does not require validation of the individual results, false positives cannot be measured for IAR. Furthermore, IAR also relies more heavily on the will of the experimental subjects to co-operate. An experimental subject who simply compiles a high number of larger components without really looking at these components more closely, will distort the results. On the other hand, if not every participant reviews the whole reference corpus, it may happen that people would not agree to certain parts. That is why both RCR as well as IAR should be evaluated.

4.6. Statistical Analysis

The common way to evaluate statistical data of controlled experiments is to apply analysis of variance (ANOVA). The F statistic, for example, may be used to test the hypothesis that the population means for the two groups are equal [33]. However, the F statistics and other statistical tests of ANOVA assume a certain distribution of the population or themselves approach a normal distribution only for large samples. However, normal distribution cannot be assumed for experiment in component recovery methods because too little is known about the programmer population. Furthermore, if the size of the sample is small, one cannot evaluate it with the F statistics.

There are other statistics, so-called *non-parameterized statistics*, that do not assume any distribution and are applicable to small samples. The power of these tests is generally better than the power of parameterized tests. According to Lienert [19], there are basically two kinds of statistics appropriate for a simple completely randomized design with non-repeated measurement (there are variants of these tests for more complex designs) and small samples:

- The *exact U-test by Mann and Whitney* [20] and
- the *exact Fisher-Pitman randomization test* for two independent samples [26].

These two methods differ in the leveraged scaling information of the data. The exact U-test assumes data at an ordinal scale, i.e., the data can only be compared in terms of a greater/lesser relationship, whereas the exact Fisher-Pitman test is based on interval information. Since the recall rate is actually at an interval scale, Fisher-Pitman test seems to be the appropriate test. However, it assumes that the samples are an exact image of the whole population which cannot really be justified when only students participate.

5. Setting up a Benchmark

In order to establish a comparison point for the detection quality of the automatic recovery techniques, software engineers have to manually compile a list of reference components for diverse systems. These reference components cannot only be used for evaluating automatic and semi-automatic recovery techniques but also for other purposes, like statistical analyses or for calibrating parameters of diverse metrics.

This section summarizes briefly how our reference corpora were obtained and validated and how this may be done to add new reference corpora to the benchmark suite.

There are several aspects that have to be considered for obtaining references:

- **Partial vs. exhaustive search.** An exhaustive search strives to find all references of the whole system whereas a partial search tries to identify references within a given subset of a system.
- **People.** How many people analyzed the system? What characteristics do these people have (age, experience, training)? Are they authors of the system? If there were several people, did they analyze separately or jointly?
- **Validity.** Were measures for quality assurance in place, i.e., were the results reviewed by others?
- **Available time.** How much time was available?
- **System characteristics.** What is the size, programming language, age, and application domain of the system? What is the number of user defined types, global variables, and routines?

In order to get the most reliable reference corpora, teams should exhaustively investigate a realistic system in sufficient time and the results should be reviewed. However, this can often not be achieved in practice. Hence, the exact characteristics of any new reference corpus have to be gathered as a quality attribute of the corpus.

This section describes our experiences with setting up a reference corpus. We briefly show how the references were established and why they provide a reasonable basis for a comparison, and conclude with a general framework for jointly setting up a larger benchmark suite.

5.1. Obtaining the Reference Components

The software engineers have to be provided with the source code of each system, a summary of connections between global variables, types, and functions, and standard guidelines on the task to be done. Standard guidelines are important in order to get comparable results. It goes without saying that the guidelines should not be biased toward a specific approach.

To obtain a common basis of comparison, the compo-

nents separately detected by each individual (if people worked separately) have to be merged and then validated by at least two participants. Only those components should be accepted for which a consensus can be reached. Validating the Reference Components

The fact that the references used as comparison point are produced by people raises the question whether others would identify the same components. In order to answer this question, Girard conducted two studies whose scheme can be used to investigate new reference corpora as well [8].

In order to investigate whether multiple software engineers would identify the same atomic components, Girard performed an experiment on a subset of a system. The source files of this subset were distributed along with a cross-reference table indicating the relation among types, global variables, and functions. Four software engineers were given the task of identifying the components present. Then a description of the procedure they followed along with their results were collected. In cases where they seemed to have broken their own procedures, the software engineers were asked to refine either their procedure or their results. Girard also revisited with them those components where a comment indicated that they were unsure or something was unclear and corrected their results according to their conclusions. Finally, the separate results of the analysts were quantitatively compared. For these software engineers, the average agreement was 0.75 (the exact details of this comparison can be found in [8]). These agreements lead us to believe that the reference components are a suitable comparison point.

In order to assess if these experiment results from a system subset can be generalized to a complete system, one software engineer identified components in the whole system. The components he identified were compared to those of the reference components gained by a different group (those obtained by consensus). A quantitative evaluation of the degree of agreement showed that the agreement gained on a smaller subset could indeed be generalized to the rest of the system [8].

The above schemes can be used for new reference corpora as well in order to validate that the references are a suitable oracle.

5.2. Gathering Candidates for the Benchmark

The systems to be selected for the benchmark suite should satisfy the three requirements for a reference corpus stated by Lakhoria and Gravelly [16]:

- The programs should be representative real-world programs.
- They should be available for researchers.

- The set of references need to be known in order to constitute an oracle for a comparison with candidate components.

We have reference corpora for four C systems, namely, Aero, Bash, CVS, and Mosaic (excluding the user interface of Mosaic), with altogether 120 KLOC. We want to make these reference corpora available on the web as part of the Reverse Engineering Infrastructure Initiative [27]. A prerequisite for obtaining our benchmark will be to give feedback on the quality of our references and/or to submit new reference corpora. For each new reference corpus, the figures described in the beginning of this section need to be gathered. We also plan to make the tools for the automatic comparison available. In order to exchange references, a simple ASCII-based exchange format will be specified as part of the initiative for a common exchange format for the reverse engineering community. Information on the reference corpora and the evaluation tools is available at

<http://www.informatik.uni-stuttgart.de/ifi/ps/clustering>

6. Conclusion

Little has been achieved toward a widely accepted quantitative comparison of existing clustering techniques since Lakhota and Gravely have stated the need for a benchmark with an accepted evaluation method in 1995 [16]. The need for such a comparison is even more urgent than 5 years ago: we know of more than 25 different clustering techniques and the list is still growing [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 17, 18, 21, 22, 23, 25, 28, 29, 30, 32, 34].

For a quantitative comparison, we need:

- an accepted evaluation method
- a set of reference components that serve as an oracle
- an organizational framework that creates momentum, gathers reference corpora, and makes these available along with tools to exchange and evaluate reference corpora

In this paper, we presented a way of quantitatively comparing candidate and reference components that establishes corresponding components, handles subcomponents, tolerates smaller divergences, and determines recall and precision appropriately. This metric is used to ascertain the congruence of candidates of an automatic technique and an oracle but can also be used in controlled experiment to compare results of different analysts and methods.

The metric of congruence of candidates and references we have proposed does not leverage hierarchical information as produced by hierarchical clustering techniques. For re-modularization techniques, we do not expect this as a practical drawback because the references will likely be non-hierarchical. However, if subsystems are to be compared, a hierarchical correspondence should be established.

We have recently extended our definition of correspondence to hierarchical components [14] and are currently working on defining congruence based on this hierarchical correspondence.

Moreover, we discussed issues to be considered in controlled experiments to evaluate manual and semi-automatic methods that involve human intervention. In particular, non-parameterized tests are the only appropriate tests for small-scale experiments. We described how a benchmark can be obtained, what information should be gathered, and how quality of the benchmark can be assured. The benchmark may also be useful for other purposes, like statistic analyses, metric evaluation, or calibration of clustering parameters.

It is our hope in proposing our evaluation method and in offering evaluation tools and reference corpora to get feedback from the clustering community that will finally lead to a standard test for all new re-modularization techniques.

Acknowledgment

We want to thank Jean-Francois Girard and Georg Schied for their contributions to the early version of the evaluation metric.

References

- [1] Belady, L.A., Evangelisti, C.J., 'System Partitioning and its Measure', *Journal of Systems and Software*, 2(1), pp. 23-29, February 1982.
- [2] Canfora, G., Cimitile, A., Munro, M., 'An Improved Algorithm for Identifying Objects in Code', *Journal of Software Practice and Experience*, 26(1), pp. 25-48, January 1996.
- [3] Choi, S.C., Scacchi, W., 'Extracting and Restructuring the Design of Large Systems', *IEEE Software*, 7(1), pp. 66-71, January 1990.
- [4] Doval, D., Mancoridis, S., Mitchel, B.S, Chen, Y., Gansner, E.R., 'Automatic Clustering of Software Systems using a Genetic Algorithm', *Proceedings of the International Conference on Software Tools and Engineering Practice*, August 1999.
- [5] Gall, H., Klösch, R., 'Finding Objects in Procedural Programs: An Alternative Approach', *Proceedings of the Second Working Conference on Reverse Engineering*, pp. 208-216, Toronto, July 1995.
- [6] Ghezzi, G., Jazayeri, M., Madrioli, D., 'Fundamental Software Engineering', Prentice Hall International, 1991.
- [7] Girard, J.F., Koschke, R., Schied, G., 'A Metric-based Approach to Detect Abstract Data Types and Abstract State Encapsulation', *Journal on Automated Software Engineering*, 6, pp. 357-386, Kluwer 1999.
- [8] Girard, J.F., Koschke, R., 'A Comparison of Abstract Data Type and Objects Recovery Techniques', *Journal Science of Computer Programming*, Elsevier, to appear.
- [9] Girard, J.F., Koschke, R., 'Finding Components in a Hierarchy of Modules: a Step Towards Architectural

Understanding', International Conference on Software Maintenance, pp. 58-65, Bari, October 1997.

[10] Hutchens, D.H., Basili, V.R., 'System Structure Analysis: Clustering with Data Bindings', IEEE Transactions on Software Engineering, SE-11(8), pp. 749-757, August 1985.

[11] Kazman, R., Carrière, S.J., 'Playing detective: reconstructing software architecture from available evidence', Technical Report CMU/SEI-97-TR-010, ESC-TR-97-010, Software Engineering Institute, Pittsburgh, USA, 1997.

[12] Kirk, Roger E., 'Experimental Design: Procedures for the Behavioral Sciences', 1994.

[13] Koschke, R., 'An Semi-Automatic Method for Component Recovery', Proceedings of the Sixth Working Conference on Reverse Engineering, pp.256-267, Atlanta, October 1999.

[14] Koschke, R. (2000), 'Atomic Architectural Component Detection for Program Understanding and System Evolution', Ph.D. thesis. University of Stuttgart, to appear.

[15] Lakhotia, A., 'A Unified Framework for Expressing Software Subsystems Classification Techniques', Journal Systems Software, Elsevier Science Publisher, 36, pp. 211-231, 1997

[16] Lakhotia, A., Gravely, J.M., 'Toward Experimental Evaluation of Subsystem Classification Recovery Techniques', Proceedings of the Second Working Conference on Reverse Engineering, pp. 262-269, Toronto, July 1995.

[17] Lindig, C., Snelting, G., 'Assessing Modular Structure of Legacy Code Based on Mathematical Concept Analysis', Proceedings of the Nineteenth International Conference on Software Engineering, Boston, 1997.

[18] Liu, S.S., Wilde, N., 'Identifying Objects in a Conventional Procedural Language: An Example of Data Design Recovery', Proceedings of the International Conference on Software Maintenance, pp. 266-271, November 1990.

[19] Lienert, G.A., 'Verteilungsfreie Methoden in der Biostatistik', Verlag Anton Hain, Meisenheim am Glan, Germany, 1973.

[20] Mann, H.B., Whithney, D.R., 'On a Test of Whether One of Two Random Variables is Stochastically Larger than the Other', Annals of Mathematical Statistics, 18, 1947.

[21] Müller, H., Wong, K., Tilley, S., 'A Reverse Engineering Environment Based on Spatial and Visual Software Interconnection Models', Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments, pp 88-98, Tyson's Corner, December 1992.

[22] Müller, H.A., Orgun, M.A., Tilley, S.R., Uhl, J.S., 'A Reverse Engineering Approach to Subsystem Structure Identification'. Journal of Software Maintenance: Research and Practice, 5(4), pp. 181-204, December 1993.

[23] Ogando, R.M., Yau, S.S., Wilde, N., 'An Object Finder for Program Structure Understanding in Software Maintenance', Journal of Software Maintenance, 6(5), pp. 261-83, September-October 1994.

[24] Owen, D.B., 'Handbook of Statistical Tables', Addison-Wesley, 1962.

[25] Patel, S., Chu, W., Baxter, R., 'A Measure for Composite

Module Cohesion', Proceedings of the Fourteenth International Conference on Software Engineering, pp. 38-48, Melbourne, May 1992.

[26] Pitman, E.J.G., 'Significance Test Which May be Applied to Samples From Any Population', Journal of the Royal Statistics Society 4, 1937.

[27] Rugaber, S., Wills, L., 'Building up an Infrastructure for Reverse Engineering', Proceedings of the Third Working Conference on Reverse Engineering, pp. 120-130, Monterey, September 1996.

[28] Sahraoui, H., Melo, W, Lounis, H., Dumont, F., 'Applying Concept Formation Methods to Object Identification in Procedural Code', Proceedings of the Twelfth Conference on Automated Software Engineering, pp. 210-218, Nevada, November 1997.

[29] Schwanke, R. W., 'An Intelligent Tool for Re-engineering Software Modularity', Proceedings of the International Conference on Software Engineering, pp. 83-92, May 1991.

[30] Siff, M., Reps, T., 'Identifying Modules via Concept Analysis', Proceedings of the International Conference on Software Maintenance, pp. 170-179, Bari, October 1997.

[31] Tzerpos, V., Holt, R., 'MoJo: A Distance Metric for Software Clusterings', Proceedings of the Sixth Working Conference on Reverse Engineering, pp. 187-193, Atlanta, July 1999.

[32] Valasareddi, R.R., Carver, D.L., 'A Graph-based Object Identification Process for Procedural Programs', Proceedings of the Fifth Working Conference on Reverse Engineering, pp. 50-58, Honolulu, October 1998.

[33] Winer, B.J., Brown, D.R., Michels, K.M., 'Statistical Principles in Experimental Design', 3rd edition, McGraw-Hill Series in Psychology, 1991.

[34] Yeh, A.S., Harris, D., Reubenstein, H., 'Recovering Abstract Data Types and Object Instances From a Conventional Procedural Language', Proceedings of the Second Working Conference on Reverse Engineering, pp. 227-236, July 1995.