Empir Software Eng (2016) 21:72–103
DOI 10.1007/s10664-014-9347-3

Weighing lexical information for software clustering in the context of architecture recovery

Anna Corazza · Sergio Di Martino · Valerio Maggio · Giuseppe Scanniello

Published online: 21 March 2015

Abstract In this paper, we present a software clustering approach that leverages the information conveyed by the zone in which each lexeme appears in the classes of object oriented systems. We define six zones in the source code: Class Name, Attribute Name, Method Name, Parameter Name, Comment, and Source Code Statement. These zones may convey information with different levels of relevance, and so their contribution should be differently weighed according to the software system under study. To this aim, we define a probabilistic model of the lexemes distribution whose parameters are automatically estimated by the Expectation-Maximization algorithm. The weights of the zones are then exploited to compute similarities among source code classes, which are then grouped by a k-Medoid clustering algorithm. To assess the validity of our solution in the software architecture recovery field, we applied our approach to 19 software systems from different application domains. We observed that the use of our probabilistic model and the defined zones improves the quality of clustering results so that they are close to a theoretical upper bound we have proved.

Communicated by: Thomas Zimmermann

A. Corazza · S. Di Martino
Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Napoli, Italy

A. Corazza
e-mail: anna.corazza@unina.it

S. Di Martino
e-mail: sergio.dimartino@unina.it

V. Maggio (✉)
Department of Information and Electrical Engineering and Applied Mathematics, University of Salerno, Fisciano, Salerno, Italy
e-mail: vmaggio@unisa.it

G. Scanniello
Dipartimento di Matematica, Informatica e Economia, University of Basilicata, Potenza, Italy
e-mail: giuseppe.scanniello@unibas.it

Keywords Software understanding · Reengineering · Software clustering · Probabilistic model · Software maintenance

1 Introduction

Some pieces of knowledge on software systems are not explicitly stated in the documentation even if that documentation is up-to-date (Kuhn et al. 2007). Thus, software engineers have to focus on the lexicon (i.e., identifiers and comments) in source code that embeds the domain knowledge provided by developers (De Lucia et al. 2012; Ali et al. 2011; Marcus and Poshyvanyk 2005; Poshyvanyk and Marcus 2006) to carry out maintenance activities (e.g., program comprehension or regression testing). Any solution that can aid software engineers can significantly impact on software development and evolution costs (Jarzabek 2007; Grubb and Takang 2003). In this scenario, the definition of approaches to group related software artifacts is a longstanding and relevant research topic (Bavota et al. 2013a; van Deursen et al. 2004; Koschke 2000; Tonella 2001; Kuhn et al. 2007; Wu et al. 2005; Maqbool and Babri 2007; Scanniello et al. 2013). The rationale is that a software engineer can be supported in his/her maintenance tasks if the entities of a large system sharing some kinds of relationships/properties are automatically grouped together into smaller parts (Wen and Tzerpos 2004; Scanniello et al. 2013). Researchers refer to this relevant topic as software clustering.

In this paper, we propose an approach towards software clustering that groups related source code artifacts relying on the lexical information provided by programmers (e.g., Maletic and Marcus 2001; Scanniello et al. 2010; Kuhn et al. 2007; Marcus and Poshyvanyk 2005; Liu et al. 2009). According to many factors, (e.g., programming attitude, time-to-market, development context) developers may place different care in choosing lexemes for different programming language constructs (De Lucia et al. 2012; Ali et al. 2011). Our proposal estimates the relevance of these constructs (or *zones*, from here on) based on the contained lexical information. To this end, we define a probabilistic model of the lexicon distribution whose parameters are automatically estimated by the Expectation-Maximization (EM) algorithm (Mclachlan and Krishnan 1996) that provides the weights of the zones. We consider the following six zones in the source code of object oriented software: (*i*) Class Name, (*ii*) Attribute Name, (*iii*) Method Name, (*iv*) Parameter Name, (*v*) Comment, and (*vi*) Source Code Statement.

The weights of the zones are then used as multipliers in the Vector Space Model (VSM) (Manning et al. 2008) to compute the lexical similarity among software entities (classes in our case). This similarity is used to group classes by means of a customization of a well-known clustering algorithm: k-Medoid (Kaufman and Rousseeuw 1990). To understand whether the introduction of the zones and the use of our approach improve the quality of clustering results in the context of software architecture recovery, we conducted an experimental assessment on 19 open source software systems written in Java.

The remainder of the paper is organized as follows: we motivate our research and provide some background on both software clustering and probabilistic models in Section 2. The proposed approach and the underlying techniques are described in Section 3, while we present the design and the execution of our empirical investigation in Section 4. The obtained results are presented and discussed in Section 5, while related work is discussed in Section 6. Final remarks and future work conclude the paper.

2 Motivation and Background

Software maintenance represents one of the most expensive and time consuming activities in the software life cycle (Erlikh 2000; Eastwood 1993; Port 1998), accounting up to 85–90% of the total cost of a software project (Grubb and Takang 2003). To support software engineers in the execution of several kinds of maintenance tasks, clustering approaches have been proposed since they provide a way to look at the properties of the clusters instead of the individual entities within them. This has the advantage to give insights or to raise questions about these entities and their shared properties without paying attention to the entire system or to each individual entity (Romesburg 2004). In the following sections, we introduce software clustering and then we will focus on those approaches that only use lexical information to group software entities. The section concludes with a theoretical background for the application of clustering analysis and the use probabilistic model.

2.1 Software Clustering

Clustering-based approaches require the choice of the features to be used in groupping entities in clusters, and the definition of a similarity measure to compare the entities with respect to the considered features. Finally, the clustering algorithm is chosen to identify groups of entities that are similar with respect to the defined and used measure (Wiggerts 1997).

Software clustering is widely used in software maintenance and evolution (Kuhn et al. 2007; Scanniello et al. 2014; Mahdavi 2005; Maqbool and Babri 2007). Indeed, there is a plethora of approaches and techniques for software clustering that support different tasks ranging from the comprehension of large software systems by recovering their architecture from source code to software refactoring and testing Ducasse and Pollet(2009; Bavota et al. 2013a, 2014b; Scanniello et al. 2013). In the literature, there are two kinds of approaches to group related source code artifacts: structural (Anquetil et al. 1999; Wiggerts 1997) and lexical (Corazza et al. 2010, 2011; Scanniello et al. 2010; Kuhn et al. 2007). In the former case, structural dependencies among source code artifacts (e.g., method invocations) are employed to determine cluster boundaries (Andritsos and Tzerpos 2005; Ducasse and Pollet 2009). In the lexical-based clustering approaches (lexical clustering, from here on), programs/classes are grouped on the basis of the similarity of their lexicon because these classes share common concepts that reflect the domain of the software system (Marcus and Poshyvanyk 2005; Liu et al. 2009). More recently, approaches that combine lexical and structural information have been proposed (Scanniello et al. 2010; Bavota et al. 2013a). For example, Bavota et al. (2013a) approach the package re-modularization problem by adapting and combining class-level coupling metrics and conceptual coupling between classes (Poshyvanyk and Marcus 2006). Class-level coupling metrics provide information about the structural cohesion of packages, while the latter measure captures the lexical information embedded in comments and source code identifiers.

2.2 Lexical Clustering

The use of lexical information (i.e., names of methods, classes, and identifiers, and source code comments) in software clustering is an enough recent research trend in software maintenance and evolution (e.g., Marcus and Poshyvanyk 2005; Maletic and Marcus 2001; Kuhn et al. 2005, 2007). For example, Kuhn et al. (2005, 2007) propose a lexical based clustering

approach asserting that the developers' domain knowledge of source code is mainly embedded in the comments and the identifier names. Indeed, what the code is about can be found in the semantics of lexicon used in the source code. In such a context, Information Retrieval (IR) techniques (Manning et al. 2008) can be applied on the assumption that source code artifacts are semantically related with each other if they share a similar vocabulary (Risi et al. 2012).

A number of IR techniques (e.g., VSM and LSI – Latent Semantic Indexing Deerwester et al. 1990), have been used for clustering source code artifacts (Kuhn et al. 2007; Scanniello et al. 2010). Based on our literature review and the experience gained in lexical clustering for architecture recovery, we can conjecture that supposedly better IR techniques and clustering algorithms do not give better software clustering results (Corazza et al. 2011, 2013; Risi et al. 2012; De Lucia et al. 2009). Accordingly, we postulate that the research has expended all the benefit that can be gained from more elaborate techniques (IR and clustering) and future progress will be achieved by exploring complementary information in the source code. In fact, researchers are following two main directions: (*i*) combining information in software artifacts (e.g., lexical and structural information) (Bavota et al. 2010, 2013a, 2014a, b; Scanniello et al. 2010) or (*ii*) taking into consideration the place where lexemes appear in source code (Corazza et al. 2010, 2011). There are benefits and drawbacks deriving from the use of approaches falling in these two categories. For example, approaches in the former category are less flexible than those in the latter category being them dependent from the programming language used to implement the system under study. However, in the second case a software engineer has to determine relevance weights (or simply weights in the following) of the zones on the basis of his/her knowledge of the software system under study. In case the software engineer's knowledge on a subject system is not adequate, automated approaches should be advisable.

2.3 Probabilistic Models

Optimal values for the relevance weights are difficult to determine, because they are strictly related to many factors concerned with the software under study. To deal with these concerns, the use of machine learning (ML) techniques represents a viable solution (Corazza et al. 2013). ML deals with the definition of algorithms and systems that improve their knowledge or performance on a given task by learning from data (Flach 2012).

The cost to apply ML varies considerably depending on the learning approach these techniques apply: supervised or unsupervised. Supervised approaches require information provided by the experts (e.g., the software architect, in our case) to annotate data for learning (Mitchell 1997). That information is usually very expensive and difficult to obtain (Ducasse and Pollet 2009; Kuhn et al. 2007). On the other hand, unsupervised approaches do not require any manual processing of input data, since the learning approach leverages information automatically obtained from these data (source code lexicon, in our case). Clustering algorithms are typical examples of unsupervised learning techniques.

Whatever the learning approach is, probabilistic models constitute a viable solution to deal with complex tasks that cannot be modeled or are too complex to formalize. These models assume that there is some underlying process that generates the data, according to a well-defined but unknown probability distribution. This distribution has to be hypothesized (Flach 2012) because deriving the type of probability distribution from the data could lead to overfitting (Bishop 2006).

In our case, this distribution represents the probability that a lexeme appears within a class. To define a unique probabilistic model, a mixture of distributions is needed: each

distribution models how lexemes are arranged within a zone. The parameters of the mixture can be determined using various criteria. The most widely adopted criterion is the Maximum Likelihood Estimation (MLE) (Bishop 2006). It maximizes the likelihood,[1] which is the probability that data have been generated by the considered probabilistic model (Bishop 2006).

Expectation-Maximization (EM) is an iterative algorithm that has been proposed to maximize the likelihood function when the probabilistic model involves hidden variables (Dempster et al. 1977) as the relevance of the zones in our case. In particular, that algorithm iterates over assigning a value to hidden variables given the current estimates of the model parameters, re-estimating them from current estimations, until a stationary configuration is reached.

3 The Approach

Our approach leverages the lexical information contained within the source code to group related classes of object oriented software systems. In particular, we consider six zones in each class:

(I) The Class Name (CN) zone contains lexemes appearing in the class declaration, i.e., the name of the class and the names of the superclass and/or the implemented interfaces, if any.
(II) The Attribute Name (AN) zone contains lexemes from the names of attributes and their corresponding types (if not primitive).
(III) The Method Name (MN) zone contains lexemes appearing in the names of methods and in their return types (if not primitive).
(IV) The Parameter Name (PN) zone contains lexemes that are in the parameters of methods. The names of the parameters and their types (if not primitive) are included.
(V) The Comment (Co) zone contains lexemes extracted from all the comments of a class. We remove copyright disclaimers placed at the beginning of source code artifacts as Kuhn et al. (2007) did.
(VI) The Source Code Statement (SCS) zone contains only the lexemes occurring in the body of methods.

To group classes, our approach consists of a pipeline composed by the following steps:

1. **Corpus Creation.** A corpus is created, containing a document for each class.
2. **Corpus Normalization.** The corpus is normalized using a different set of processing steps.
3. **Corpus Indexing.** An inverted index is constructed for each zone, storing a mapping from the lexemes to the documents containing them (Manning et al. 2008). All the documents in the corpus are represented by using VSM (Manning et al. 2008).
4. **Zone Weighting**. Each zone is automatically weighed by means of a probabilistic model, whose parameters are estimated with the MLE approach and the EM algorithm. The weights are used as multipliers for the VSM representation of the documents.
5. **Clustering Software Classes.** Classes are grouped together by means of a clustering algorithm, namely a customized version of k-Medoid (Kaufman and Rousseeuw 1990).

[1] It is equal to the probability of the observed data x given the values of the model parameters θ, i.e., $\mathcal{L}(\theta|x) = P(x|\theta)$.

The steps 1, 2, 3, and 5 are part of typical lexical clustering approaches (e.g., Kuhn et al. 2007; Risi et al. 2012), while the step 4 is a new. In the remainder of this section, we describe how we instantiated these steps.

3.1 Corpus Creation

Since our approach works at class granularity level, each class results in one document in the corpus. A document is built also in correspondence of each inner class. The rationale behind this choice is that a class and its inner class/es are different, and then they should belong to different clusters. The words from a source code class (included the inner class/es) are included in a single document taking into account the zones from which these words are gathered.

3.2 Corpus Normalization

The words in the corpus are tokenized. This tokenization process is performed by separating the words on the basis of blank and punctuation characters. To reduce noise and redundancies, all the tokens extracted from documents are then normalized (Manning et al. 2008). In Fig. 1, we report a sample Java class (i.e., `NullHandle` from jHotDraw 5.1) used as running example to show how some of the steps of our approach work.

1. HTML tags and numbers are removed as they very commonly introduce noise (Manning et al. 2008). As far as the class in Fig. 1 is concerned, the HTML tags of the lines 5–8 are removed.

```
1  /**
2   * A handle that doesn't change the owned figure.
3   * Its only purpose is to show feedback that a figure
4   * is selected.
5   * <hr>
6   * <b>Design Patterns</b>
7   * <img src="images/red-ball-small.gif" width=6 height=6>
8   * <b>NullObject</b><br>
9   * NullObject enables to treat handles that don't do
10  * anything in the same way as other handles.
11  */
12 public class NullHandle extends LocatorHandle {
13     /**
14      * The handle's locator.
15      */
16     protected Locator fLocator;
17
18     public NullHandle(Figure owner, Locator locator) {
19         super(owner, locator);
20     }
21     /**
22      * Draws the NullHandle. NullHandles are drawn as a
23      * red framed rectangle.
24      */
25     public void draw(Graphics g) {
26         Rectangle r = displayBox();
27         g.setColor(Color.black);
28         g.drawRect(r);
29     }
30 }
```

Fig. 1 The `NullHandle` Java class of jHotDraw 5.1

2. The identifiers that are written using standard coding conventions are split. Identifiers are often composed by multiple words using naming conventions. To date, we handle the use of camel case, where capitalized letters are used to divide two or more words in an identifier. For example, the identifier `NullHandle` (i.e., the name of the class shown in Fig. 1) is split in `Null` and `Handle`. This normalization operation is also applied on the identifiers that appear in the source code comments (e.g., line 9).
3. The tokens are lowercased. For example, `Null` and `Handle` are turned into `null` and `handle`.
4. From all the tokens in the six zones, we removed the lexemes contained in a list of most common English terms. A further list containing programming language keywords (i.e., Java keywords in our case) is considered for the tokens in the zones Co and SCS. For example, the terms `a, as, are, the, in, is, its, only, to, do, don, doesn, t, s, r, g, f`, and `that` are removed, along with Java keywords from the class in Fig. 1.
5. The tokens are partitioned into equivalence classes based on their morphological root (or *stem*). The stemming goal is to reduce inflectional forms and other related forms of a word to a common base form (Manning et al. 2008). To this end, we use the Porter's stemmer (Porter 1997). As for the example class in Fig. 1, `handles` and `handle` appearing in the comments (lines 1–11) are reduced to `handl`.

The normalization operations in the first two steps take into account the fact that we are dealing with source code lexicon (Binkley 2007). The remaining three steps represent common operations for the normalization of lexicon in traditional IR-based approaches (Manning et al. 2008), even if the step 4 has been here modified to remove also programming language keywords. These steps are accomplished in the same order as they are shown above. The output of Corpus Normalization is a set of terms.

3.3 Corpus Indexing

The basic idea of an inverted index is to keep a dictionary of terms,[2] where each term is connected to a list that records the references to the documents where this term occurs in (Manning et al. 2008). In our approach, we have also to keep track of the zones from which terms have been gathered. To this end, we have a vocabulary for each defined zone.

VSM is applied to represent the documents in the corpus as vectors. These representations constitute the term-document matrix A, whose entries $a_{t,d,z}$ describe the relevance of the term t with respect to the document d within the zone z. That relevance is computed using the *tf-idf* schema:

$$tf - idftdz = tftdz \cdot idftz = a_{t,d,z} \tag{1}$$

where $tftdz$ - term frequency - is defined as the number of occurrences of the term t in the document d appearing in the given zone z. On the other hand, $idftz$ - inverse document frequency - is computed as follows:

$$\log \frac{N}{dftz}$$

N is the total number of documents in the corpus, while $dftz$ indicates the number of documents in which the term t occurs, within the zone z. The rationale underlying *idf* is

[2] A term is a normalized type that is included in the dictionary. A type is the class of all the tokens containing the same character sequence (Manning et al. 2008).

that if a term appears in almost all the documents, then its discriminative contribution is irrelevant (i.e., the corresponding *idf* value is close to 0). Conversely, the value of the *idf* increases when a term appears in few documents.

The *tf-idf* schema assigns a weight that is:

1. higher when a term occurs many times within a small number of documents, so lending high discriminating power to those documents;
2. lower when a term occurs fewer times in a document, or occurs in many documents;
3. lowest when the term occurs in virtually all the documents.

We use the *tf-idf* schema because it represents a good trade-off between simplicity and effectiveness (Manning et al. 2008).

3.4 Zone Weighting

The weights of the zones are automatically estimated and used as multiplicative factors for the *tf-idf* values. Figures 2 and 3 graphically show the defined workflow to weigh the zones applying MLE and EM, by using the precise style by Reggio et al. (Reggio et al. 2011).

Figure 2 depicts the manipulated objects and the participant in the workflow by means of an UML class diagram. The class diagram explicits the participant (i.e., the EM class stereotyped by `<<worker>>`) in the workflow and six manipulated objects (stereotyped by `<<object>>`). A dependency relationship (visually depicted as a dashed arrow) goes from a class stereotyped by `<<worker>>` into a class stereotyped by `<<object>>` and shows that the participant acts over the objects of the connected classes. Figure 3 summarizes the behavioral view of how our approach works to weigh the zones.

`MixtureModel` in the class diagram (Fig. 2) represents the probabilistic model used in our approach. It is formalized by a mixture of multivariate probabilistic distributions (Bishop 2006), where each of them indicates how terms are distributed in zones.

We consider two types of distributions: Gaussian and Bernoulli. The Gaussian distribution is typically used when no previous information is available on the studied phenomenon (Bishop 2006). It assumes that the *tf-idf* values are normally distributed in each zone. On the other hand, the Bernoulli distribution is widely adopted in text retrieval approaches (McCallum and Nigam 1998). The Bernoulli formulation only considers the presence or the absence of a term in a zone.

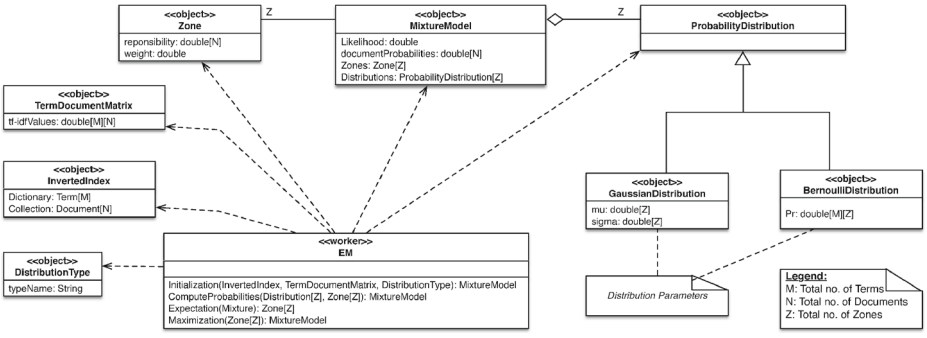

Fig. 2 EM Class Diagram modelled with the precise style (Reggio et al. 2011)

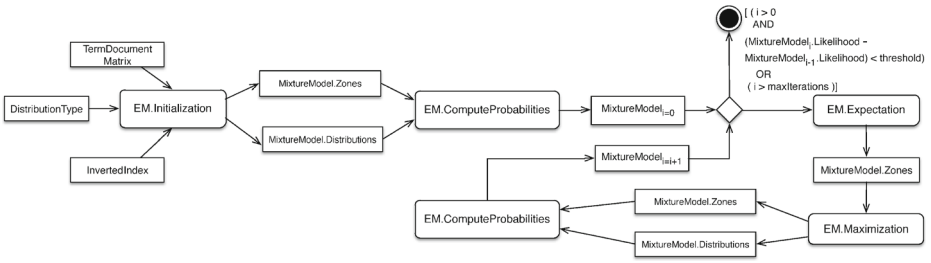

Fig. 3 EM Activity Diagram modelled with the precise style (Reggio et al. 2011)

To initialize the parameters of `MixtureModel`, the EM algorithm (that is `EM.Initialization`) takes three objects as input, namely `TermDocumentMatrix`, the `InvertedIndex`, and `DistributionType` (see Fig. 3). In Table 1, we report how the initialization is performed for the Gaussian and Bernoulli distributions, respectively. For example, for the Gaussian distribution this table shows how μ_z and σ_z^2 are computed together with α_z (i.e., the initial weight of a given zone). Whatever the type of distribution is, `EM.Initialization` produces two objects: `MixtureModel.Distributions` and `MixtureModel.Zones`.

`EM.ComputeProbabilities` is in charge of computing the values for the document probabilities and the likelihood of the model, namely the `MixtureModel` object is updated.

For each document, this probability expresses how terms in the document have been generated by the different distributions associated to the zones. Its value is calculated by multiplying the weights of the zones by the conditional probability of the document and the zones (see Table 1). Document probabilities are then used to compute the likelihood of the mixture. Indeed, we consider the log-likelihood because it avoids possible numerical issues. This choice is almost customary in ML approaches (Bishop 2006).

EM is an iterative algorithm composed of two main steps: expectation (`EM.Expectation`) and maximization (`EM.Maximization`). In the former step, the responsibility (or contribution) of each document to the zones distributions is computed as reported in Table 1. The updated `MixtureModel.Zone` object is returned as the output of `EM.Expectation` (Fig. 3). As far as the maximization step, the objects `MixtureModel.Distributions` and `MixtureModel.Zones` are updated according to the resulting output by `EM.Expectation`. These objects are then used to update the `MixtureModel` by `EM.ComputeProbabilities`.

The loop in the activity diagram (Fig. 3) is iterated until the difference between the likelihood values obtained in two subsequent iterations is less than a given threshold (i.e., the EM algorithm converged to a local-optimal value for the likelihood) or a maximum number of iterations has been performed (see the guard condition of the loop in the activity diagram).

The EM algorithm returns the values of the weights of the zones (see `Zone` in the class diagram of Fig. 2). The larger the weight, the bigger the contribution of the z-th zone to the obtained model is.

The choice of the initial values for the parameters of the EM algorithm might be critical for determining the probabilistic model (Mclachlan and Krishnan 1996). The initialization is straightforward when the probabilistic model is based on Bernoulli distributions, as it is inducted by the formalization of the model (see Table 1). On the other hand, in case

Table 1 EM computation steps for the probabilistic models based on the Gaussian and Bernoulli distribution types

Gaussian	Bernoulli
Initialization	
$\mu_z = \frac{1}{N}\sum_{d=1}^{N}\sum_{t:t\in d} tf-idftdz$	$I(t,d,z) = \begin{cases} 1 & t \in d \text{ within the zone } z \\ 0 & \text{otherwise} \end{cases}$
$\sigma_z^2 = \frac{1}{N}\sum_{d=1}^{N}\sum_{t:t\in d} (tf-idftdz-\mu_z)^2$	$\Pr(t,z) = \frac{1}{N}\sum_{d:t\in d} I(t,d,z)$
$\alpha_z = \frac{\sum_{d=1}^{N}\sum_{t=1}^{M} tftdz}{\sum_{z'=1}^{Z}\sum_{d=1}^{N}\sum_{t=1}^{M} tftdz'}$	$\alpha_z = \frac{\sum_{t=1}^{M}\sum_{d:t\in d} I(t,d,z)}{\sum_{z'=1}^{Z}\sum_{t=1}^{M}\sum_{d:t\in d} I(t,d,z')}$
Compute probabilities	
$\Pr(d\|z) = \prod_{t=1}^{M} \mathcal{G}(\mu_z,\sigma_z)$	$\Pr(d\|z) = \prod_{t:t\in d}\Pr(t,z)\prod_{t:t\notin d}(1-\Pr(t,z))$
$\Pr(d) = \sum_{z=1}^{Z}\alpha_z \Pr(d\|z)$	$\Pr(d) = \sum_{z=1}^{Z}\alpha_z \Pr(d\|z)$
$\log\mathcal{L} = \sum_{i=1}^{N}\log\Pr(d)$	
Expectation	
$r_{d,z} = \frac{\alpha_z \Pr(d\|z)}{\sum_{z'=1}^{Z}\alpha_z \Pr(d\|z)}$	$r_{d,z} = \frac{\alpha_z \Pr(d\|z)}{\sum_{z'=1}^{Z}\alpha_z \Pr(d\|z)}$
Maximization	
$\mu_z = \frac{\sum_{d=1}^{N} r_{d,z}\sum_{t:t\in d} tf-idftdz}{\sum_{d=1}^{N} r_{d,z}}$	$\Pr(t,z) = \frac{\sum_{d:t\in d} r_{d,z}}{\sum_{d=1}^{N} r_{d,z}}$
$\sigma_z^2 = \frac{\sum_{d=1}^{N} r_{d,z}\sum_{t:t\in d} (tf-idftdz-\mu_z)^2}{\sum_{d=1}^{N} r_{d,z}}$	
$\alpha_z = \frac{\sum_{d=1}^{N} r_{d,z}}{\sum_{z'=1}^{Z}\sum_{d=1}^{N} r_{d,z'}}$	$\alpha_z = \frac{\sum_{d=1}^{N} r_{d,z}}{\sum_{z'=1}^{Z}\sum_{d=1}^{N} r_{d,z'}}$

Z =Total no. of zones, N = Total no. of documents, and M = Total no. of terms

of Gaussian distribution based models, different strategies may be applied (Corazza et al. 2010). We experimentally observed that good estimates for the weights could be obtained by considering the rate between the number of tokens in the zones and the total number of tokens when choosing the initial parameters of the model.

3.5 Clustering Software Classes

In VSM, each document is represented as a point in a multi-dimensional Euclidean space. The similarity between pairs of documents is computed using the cosine similarity

Algorithm 1 The k-Medoid Algorithm

Input: k : The total number of clusters to generate.
Input: D : The collection of input documents.
Input: T : Total number of iterations allowed.
Output: P : The array of k clusters of the partition.

```
1:  Function k-Medoids(k, D)
2:      iterCount ← 0
3:      repeat
4:          {m_1, m_2, ..., m_k} ← InitializeMedoids(k, D)
5:          P ← [{m_1}, {m_2}, ..., {m_k}]
6:          repeat
7:              D̂ ← {d_i : d_i ∈ D ∧ d_i ∉ {m_1, m_2, ..., m_k}}
8:              for each: d_i ∈ D̂ do
9:                  for j = 1 to k do
10:                     sim_{i,j} ← CosineSimilarity(d_i, m_j)
11:                 end for
12:                 imax ← argmax_{1≤j≤k}(sim_{i,j})
13:                 P[imax] ← P[imax] ∪ {d_i}
14:             end for
15:             oldMedoids ← {m_1, m_2, ..., m_k}
16:             {m_1, m_2, ..., m_k} ← UpdateMedoids P
17:             iterCount ← iterCount + 1
18:         until (oldMedoids = {m_1, m_2, ..., m_k}) ∨ (iterCount = T)
19:         NoTinyClusters ← CheckSizeOfClusters (C)
20:     until (NoTinyClusters = True) ∨ (iterCount = T)
21:     return P
22: end function
```

(Salton et al. 1975), which is the length normalized version of the inner product between the vectors corresponding to these documents.

To identify groups of classes that are similar with respect to the defined similarity measure, we used the k-Medoid clustering algorithm (Kaufman and Rousseeuw 1990). Indeed, we adapted this algorithm introducing a new halting criterion to reduce the risk of recovering too tiny clusters (i.e., containing very few source code classes). This is recognized as a desirable property in software clustering (Risi et al. 2012; Wu et al. 2005).

The pseudocode of the used k-Medoid clustering algorithm is reported in Algorithm 1.

This algorithm starts by initializing the medoids (Line 4) by randomly choosing a set of k different documents. By default k is equal to 10% of the total number of documents in the corpus (Bittencourt and Guerrero 2009). Then, the algorithm assigns each remaining document to the most similar medoid according to the cosine similarity measure (line 10). Afterwards, the new cluster configuration is computed and the corresponding medoids are updated. The iteration ends when clusters' medoids stop changing (line 18). The whole procedure is repeated until the recovered partition (clustering result) does not contain too tiny clusters or the maximum number of iterations has been performed (Line 20).

4 Experimentation

4.1 Introduction and Problem Statement

The main goal of our experimental investigation is to quantitatively evaluate the benefits deriving from the use of both the zones and the probabilistic model. To this end, the following research questions (RQs) have been defined and investigated:

RQ1. Does the use of zones improve the quality of clustering?
RQ2. Does the use of zones weighed by applying our approach improve the quality of clustering?
RQ3. Is the quality of the clustering affected by the type of the chosen distribution (i.e., Gaussian and Bernoulli)?
RQ4. Is the partition obtained with the probabilistic model close to the best possible partition?

In this section, we present the design of our investigation following the guidelines proposed by Wohlin et al. (2000). For replication purposes, an experimental package, the raw data, and the scripts used to analyze these data are available on the web.[3]

4.2 Definition and Context

Applying the Goal Question Metric (GQM) template (Mashiko and Basili 1997), the goal of our investigation can be defined as follows:

> **Analyze** our six zones **for the purpose of** evaluating their use **with respect to** software clustering applied to architecture recovery **from the point of view** of the researcher and the professional, assessing whether the clustering quality improves when using MLE and EM **in the context of** open source software systems written in Java.

The use of GQM ensured us that the most important aspects of our investigation were defined before its planning and before its execution took place (Wohlin et al. 2000).

The investigation has been conducted on 19 open source Java software systems having different application domains. We chose these systems because they were previously used in empirical investigations conducted in software maintenance and software clustering (e.g., Wu et al. 2005; Romano et al. 2011; Corazza et al. 2011). For each system, we have concentrated on the latest version available on the web when our experimental assessment started on December 15th, 2012.

Information on these systems is shown in Table 2. The first column shows the ID of each system (referenced in Fig. 5), while the second column reports the name of the software system. The analyzed version of each system and the number of classes are reported in the second and third columns, respectively. The number of lines of codes (KLOCs) is shown in the fifth column, while the number of lines of comments is presented in the sixth column.

[3] http://www2.unibas.it/gscanniello/software-clustering/

Table 2 The dataset

ID	System	Version	Classes	KLOCs	KCLOCs	Description
1	Ant	1.8.4	1452	103.56	89.6	A library and command-line tool to define build files for software applications implemented in Java. (ant.apache.org)
2	Lucene	3.6.1	1015	63.2	36.1	A framework that implements IR algorithms. (lucene.apache.org)
3	Tomcat	5.0	1530	163.8	110.5	A Servlet/JSP container for Java web applications. (tomcat.apache.org)
4	Azureus	4.8.1	4785	333.1	97.1	A client for sharing files using the BitTorrent file-sharing protocol. (www.vuze.com)
5	Hibernate	4.1.9	2267	156.0	95.8	An ORM (Object Relational Mapping) library for Java applications. (www.hibernate.org)
6	iText.Net	5.3.4	1201	77.4	50.3	An open source library for creating and manipulating PDF, RTF, and HTML files. (itextpdf.com)
7	jEdit	4.5.2	869	88.4	36.1	A text editor suited to support programming tasks. (www.jedit.org)
8	jFreeChart	0.6.0	89	8.7	7.8	A tool supporting the visualization of bar charts, line charts, scatter plots, histograms, and more. (www.jfree.org/jfreechart)
9	jFTP	1.5.6	469	23.5	4.7	A graphical Java network and file transfer client. (j-ftp.sourceforge.net)
10	jHotDraw	7.4.1	899	73.0	38.3	A GUI framework for technical and structured graphics. (www.jhotdraw.org)
11	jRefactory	2.9.19	1522	110.7	91.3	A GUI application for the refactoring of Java source code projects. (jrefactory.sourceforge.net)
12	jUnit	4.8	547	15.0	4.1	A testing framework for Java programs. (junit.org)

Table 2 (continued)

ID	System	Version	Classes	KLOCs	KCLOCs	Description
13	Liferay Portal	6.1	3961	379.1	137.3	An open source enterprise web platform for building business web-based solutions. (www.liferay.com)
14	Pmd	4.2.5	680	49.6	8.9	A Java source code analyzer able to find unused variables, unnecessary object creation, and so forth. (pmd.sourceforge.net)
15	Synapse	1.2	613	45.7	20.9	An Enterprise Service Bus application providing support for XML, web services and REST applications. (synapse.apache.org)
16	Tiger Envelopes	0.8.9	917	73.4	25.9	An open source personal mail proxy that automatically encrypts and decrypts mail. (tigerenvelopes.sourceforge.net)
17	Velocity	1.6.1	419	35.8	25.0	A framework to build web and non-web applications. (velocity.apache.org)
18	Xalan	2.5.0	915	123.7	128.3	XSLT processor for transforming XML documents into HTML, text, and other XML document types. (xml.apache.org/xalan-j)
19	Xerces	1.4.4	578	71.5	63.6	A collection of components and utilities to parse, validate, and serialize XML documents. (xerces.apache.org)

The seventh column shows a short description of each system and the URL of its official web page within brackets.

4.3 Planning

Our choice of empirical evaluation is based on considering the original allocation of source code files in directories as the authoritative partition. This is not new and is well known in the architecture recovery field (e.g., Risi et al. 2012; Scanniello et al. 2010, Bittencourt and Guerrero 2009; Wu et al. 2005). The rationale is: given the bunch of classes of a software system, if an approach is able to automatically arrange these classes in a partition that resembles the packages proposed by the developers of the systems, then the approach would likely perform well also on other systems (Wu et al. 2005).

4.3.1 Selected Variables

The main factor (i.e., the manipulated variable) on which our study is focused on is *Technique*. It is a nominal variable that assumes as values: VSM (flat vocabulary without zones), ZU (zones unweighed), ZWB (zones weighed by EM assuming Bernoulli distributions), ZWG (zones weighed by EM assuming Gaussian distributions), and TUB (theoretical upper bound). It is worth mentioning that TUB is an estimation of the maximum possible improvement that can be achieved by varying the weights of the six zones.

The clusters produced by a given approach should resemble as much as possible the clusters in the authoritative partition (Wu et al. 2005; Risi et al. 2012; Corazza et al. 2011; Scanniello et al. 2010). To estimate such a resemblance, we computed the MojoFM measure (Wen and Tzerpos 2004) on the clustering results and the authoritative partition. MojoFM is based on the Mojo measure. It computes the minimum number of move and join operations to transform either a source partition (i.e., the automatically identified partition) to the target one (i.e., the authoritative partition) or vice versa (Tzerpos and Holt 1999). In the context of architecture recovery, MojoFM has been introduced to overcome the following two drawbacks of MoJo: (*i*) it does not make the clustering results comparable among different software systems since its value depends on the number of classes in the system under study and (*ii*) we are interested in determining how the automatically defined partition resembles the authoritative one and not vice versa. We compute MojoFM as proposed by Wen and Tzerpos in (2004):

$$\text{MojoFM(A, B)} = 1 - \frac{mno(A, B)}{\max(mno(\forall A, B))} \tag{2}$$

where *mno* indicates the minimum number of move and join operations needed to turn A (the partition obtained by applying our approach) into B (the authoritative partition), while $\max(mno(\forall A, B))$ represents the maximal number of these operations to partition B from every possible partition derived by the elements in A. In our study, we used the implementation of MoJo available at www.cse.yorku.ca/∼bil/downloads.

MojoFM assumes values in between 0 and 1. The larger the value, the most the partition recovered by the approach resembles the authoritative one. When comparing two approaches, the one with the highest authoritative value (i.e., less effort to refine the automatically recovered partition) is considered the best in term of clustering quality.

Since the k-Medoid algorithm may produce slightly different partitions on the same set of classes, we executed this algorithm five times on the classes of each system and each time we compute the MojoFM on the identified partitions. The arithmetic mean of the MojoFM values is our dependent variable. We refer to this variable as the authoritativeness.

4.3.2 Hypotheses Formulation

With respect to the defined research questions and the considered dependent and independent variables, we have formulated the following null hypotheses:

Hn1: There is not a statistically significant difference between the values of authoritativeness obtained with VSM and ZU.

Hn2: There is not a statistically significant difference between the values of authoritativeness obtained with weighed (ZWB and ZWG) and unweighed zones (ZU).

Hn3: There is not a statistically significant difference between the values of authoritativeness obtained with ZWB and ZWG.

The goal of the statistical analysis is to reject these null hypotheses and possibly to accept the alternative ones (e.g., Ha1 = ¬Hn1), which can be easily derived because they admit a positive effect of Technique on authoritativeness. Hn1, Hn2, and Hn3 have been defined to investigate RQ1, RQ2, and RQ3, respectively. As far as RQ4, statistical tests are not adequate (see Press et al. 1992, page 620) because this research question does not involve a performance estimate, but an upper bound which has been estimated as the maximum for the corresponding sample. Therefore, we analyze the differences of the obtained authoritativeness values between ZWG and TUB and between ZWB and TUB.

4.3.3 Design

We use a one factor with more than two treatments experiment design (Wohlin et al. 2000). In particular, we analyze the effect of each technique on the 19 systems we selected and studied.

4.3.4 Preparation and Execution

4.3.5 Authoritative Partition

To obtain authoritative partitions, we exploited the approach proposed by Wu et al. in (2005). This approach can be summarized as follows:

1. create the subsystem hierarchy based on the directory structure (each directory is a subsystem);
2. merge a subsystem with its parent in case it contains a number of source code files that is less than or equal to five;
3. create a cluster for each resulting subsystem.

4.3.6 Theoretical Upper Bound

To correctly judge any improvement in the clustering authoritativeness, it is important to estimate for a given software the maximum possible authoritative value achievable by varying the weights of the six zones.

A possible strategy to perform this estimation consists in randomly assigning the weights and computing authoritativeness values. In case this is repeated a large number of times, the maximum of the obtained authoritativeness values can be considered as an estimate of the highest authoritativeness value. However, it is important to know if this estimation is reliable.

We control the probability distribution of the weights, assuming that they are uniformly distributed in between 0 and 1. However, no hypothesis can be made about authoritativeness values. In fact, we can hypothesize neither that the distribution of authoritativeness values is uniform nor that their probability distribution is symmetric around the mean.

We consider a random sample $S = X_1, X_2, \ldots, X_n$ of values for the random variable X corresponding to clustering quality. n is the number of random assignments (or repetitions) to the weights of the zones. In this scenario, we can apply the Chebyshev theorem (Saw et al. 1984) to the sample maximum X_{MAX}. This theorem allows estimating an Upper Bound (UB) of the probability that there exist authoritativeness values greater than X_{MAX}, provided that considered samples are independently and identically distributed.

The resulting upper bound only depends on the variance of the sample, and on the difference between X_{MAX} and the mean of the sample: the larger the rate λ between the two, the lower the probability that the random variable assumes a greater value (Freund and Wilson 2003).

We compute the mean μ and the variance σ^2 of the sample as:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{3}$$

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n} \mid X_i - \mu \mid^2 \tag{4}$$

If we have $Q^2 = \frac{n+1}{n}\sigma^2$ and $\lambda \geq 1$, we can write

$$\Pr(\mid X - \mu \mid > \lambda Q) \leq \frac{v}{n+1} \tag{5}$$

where v is an integer which depends on n and λ.[4]

In our case, we choose λ such that $\mid X_{\mathrm{MAX}} - \mu \mid = \lambda Q$ and we obtain:

$$\Pr(X > X_{\mathrm{MAX}}) \leq \Pr(\mid X - \mu \mid > X_{\mathrm{MAX}} - \mu) \leq \mathrm{UB} = \frac{v}{n+1} \tag{6}$$

Table 3 reports the values for UB on the studied software systems: we empirically checked that in all cases $\lambda \geq 1$ so that we can apply relation (6). The reliability index of the maximum estimate (see the third column) assumes values in between 0 and 1. When this number is close to zero (e.g., JFTP), the maximum estimate for the authoritativeness is nearly sure. In many cases, we obtained values for the reliability index smaller than 0.3 performing a number of repetitions n equal to 200. The only exception is represented by Xerces, where 300 repetitions were needed.

4.3.7 Analysis Procedure

We use general descriptive statistics (i.e., min, max, median, mean and standard deviation) and box plots. Box plots provide a quick visual representation to summarize the data using five numbers: the median, upper and lower quartiles, minimum and maximum values, and outliers.

To test the defined null hypotheses, we use the Wilcoxon rank-sum test (also known as Mann-Whitney) (Conover 1998). It tests whether samples originate from the same distribution and it is used for comparing two samples that are independent, or not related. We plan to use that non-parametric test because we expect the distribution of data to be non-normal. We verified this assumption by applying the Shapiro-Wilk W test (Shapiro and Wilk 1965). In case this test returns a p-value smaller than the chosen α threshold, the data are considered as not normally distributed.

For all the statistical tests performed, we decided to accept a probability of 5 % (i.e., $\alpha = 0.05$) to commit a Type-I-Error (Wohlin et al. 2000). To test Hn2, we perform the Mann-Whitney test twice (ZWB vs. ZU and ZWG vs. ZU). To reject this hypothesis, we use a conservative approach for the compensation of repeated statistical tests, namely the

[4]The complete formulation is reported in Saw et al. (1984).

Table 3 Theoretical upper bound values

System	Max authoritativeness value	UB value
Ant	0.84	0.12
Lucene	0.89	0.14
Tomcat	0.86	0.12
Azureus	0.87	0.02
Hibernate	0.87	0.02
iText.Net	0.89	0.01
jEdit	0.88	0.03
jFreeChart	0.95	0.03
jFTP	0.87	0.00
jHotDraw	0.96	0.01
jRefactory	0.87	0.07
jUnit	0.87	0.19
Liferay Portal	0.89	0.01
Pmd	0.83	0.22
Synapse	0.83	0.25
TigerEnvelopes	0.86	0.20
Velocity	0.85	0.14
Xalan	0.89	0.03
Xerces (300 iterations)	0.87	0.29

Bonferroni correction. In practice, the hypothesis is rejected by comparing the p-values to a corrected significance level $\alpha_B = \alpha/n_t$, where n_t is the number of tests performed. In our case, the corrected α is α_B and it is equal to 0.025 (i.e., 0.05/2).

Statistical tests analyze the presence of a significant difference between groups, but they do not provide any information about that difference. We used the point-biserial correlation r to measure the magnitude of the effect size of that difference (Kampenes et al. 2006). The magnitude of the effect size is: small ($0 < r \leq 0.193$), medium ($0.193 < r \leq 0.456$), and large ($0.456 < r \leq 0.868$) (Kampenes et al. 2006). We also computed the average percentage improvement[5] as a less robust though a more intuitive qualitative effect size indicator.

The statistical power for each performed test is also computed. The statistical power is the probability that a test will reject a null hypothesis when it is actually false. Researchers assess the power of a test using 0.8 as the standard threshold for adequacy (Kevin Freedman 1999). A value larger than 0.8 indicates a high probability that a statistical test rejects a null hypothesis when it is actually false. The statistical power is computed as 1 minus the Type-II-Error (i.e., β-value). This type of error indicates that the null hypothesis is false, but the statistical test erroneously fails to reject it. In the discussion of the results, the β-value is used only when a statistical test is not able to reject a null hypothesis. The higher the β-value, the lower the probability of erroneously not rejecting a null hypothesis is.

[5]Given two values a and b the means percentage improvement is computed as $\frac{(b-a)}{a}$.

4.3.8 Threats to Validity

To comprehend the strengths and limitations of our empirical investigation, we present and discuss threats that could affect the validity of the achieved results. Despite our efforts to mitigate as many threats to validity as possible, some are unavoidable.

The most important threat to the validity of our results is related to the authoritative partitions we used to assess clustering quality. In fact, the original distribution of the classes in package could affect the authoritative partitions and then the obtained results. To overwhelm any doubts on the validity of our approach, software engineers with experience on the studied systems should be involved to identify authoritative partitions. On open source software systems, this becomes difficult in practice since it is very demanding to find people, who have a deep knowledge on these systems and delighted to be involved in studies like the one presented in this paper. This issue could be possibly overcome involving software industries, where their developed software could be employed. This part of our research might be challenging because software companies are generally reluctant to share both their source code and software engineers and developers that are expert on that code. Even if clustering depends on the task at hand and on human factors (e.g., a person might group together two entities, while another not), the existence of a benchmark should allow dealing with the issues discussed previously. Unfortunately, no benchmarks built on software systems written in Java are available and therefore the building of our authoritative partitions as we did can be considered an acceptable trade-off also taking into account that it is widely undertaken in the context of architecture recovery (e.g., Bittencourt and Guerrero 2009; Wu et al. 2005; Scanniello et al. 2010; Risi et al. 2012). The used approach, originally proposed by Wu et al. (2005), has also the advantage to identify authoritative partitions is fair and repeatable way.

The measure to assess authoritativeness could also affect the results. In our empirical investigation, we opted for a well known and widely used measure (e.g., Bavota et al. 2010), namely MojoFM (Wen and Tzerpos 2004). It is also worth noting, that MojoFM has been applied on the partitions identified by applying each of the methods (i.e., VSM, ZU, ZWB, ZWG, and TUB) considered in our investigation. In this way, the comparison among these methods should be as much fair as possible.

The nondeterminism of the k-Medoid algorithm might also threaten the validity of the results. In other words, clustering results depend on the chosen value for k and on the random initialization of medoids. Regarding k, we chose values proportional to the size of the systems under study. In addition, the number of classes in each cluster may not be well distributed. Although the NoTinyClusters variable (see Algorithm 1) can handle the cases where one cluster has too few classes, it is still possible that one cluster contains the majority of the classes and others contain only a few classes. Possible threats to the validity of the results are also related to the selected number of executions for the k-Medoid clustering algorithm. We used a conservative approach considering the average values of the authoritativeness values. Different configurations for our approach could lead to different results. To be fair, we used always the same configurations in all the instances of the clustering approach (e.g., VSM, ZU, and ZWG).

The software systems on which the approach is applied (i.e., open source) may also affect the validity of the results. It is possible that on commercial software our approach might produce different results. The size of the systems could be an additional threat to validity. As shown in Table 2, the smallest system has 8.7 KLOCs (JFreeChart), while the largest 379.1 (Liferay Portal).

Table 4 Number of terms (lexicon sizes) and their distribution in the zones

System	Terms	CN	AN	FN	PN	Co	SCS
Ant	187,615	1 %	4 %	6 %	4 %	41 %	44 %
Lucene	98,107	1 %	5 %	5 %	5 %	42 %	43 %
Tomcat	306,686	1 %	5 %	6 %	4 %	36 %	49 %
Azureus	570,813	2 %	5 %	8 %	6 %	14 %	66 %
Hibernate	299,879	2 %	5 %	10 %	8 %	20 %	54 %
iText.Net	165,626	1 %	5 %	5 %	5 %	31 %	55 %
jEdit	157,072	1 %	3 %	5 %	4 %	25 %	62 %
jFreeChart	26,639	1 %	2 %	5 %	6 %	31 %	55 %
jFTP	36,429	1 %	7 %	5 %	4 %	16 %	68 %
jHotDraw	127,633	1 %	4 %	7 %	6 %	28 %	54 %
jRefactory	229,178	2 %	2 %	6 %	6 %	27 %	58 %
jUnit	14,122	4 %	3 %	14 %	8 %	26 %	45 %
Liferay Portal	747,225	1 %	16 %	8 %	8 %	11 %	56 %
Pmd	85,828	2 %	4 %	9 %	8 %	14 %	63 %
Synapse	121,967	1 %	4 %	6 %	4 %	19 %	65 %
Tiger Envelopes	156,423	1 %	5 %	6 %	4 %	23 %	62 %
Velocity	54,991	2 %	2 %	6 %	5 %	39 %	46 %
Xalan	180,331	1 %	7 %	5 %	4 %	44 %	40 %
Xerces	160,792	1 %	3 %	5 %	4 %	38 %	50 %

Possible threats are also related to the statistical analyses. We use statistical tests well known for their robustness and sensitiveness (Wohlin et al. 2000).

5 Results and Discussion

In this section, we present the achieved results with respect to the defined research questions. We conclude discussing the implications of these results from the researcher and the practitioner perspectives.

5.1 Descriptive Statistics

In Table 4, we summarize how the terms are distributed among the zones after the Corpus Normalization step. In particular, in this table we report the total number of unique terms[6] present in each of the analyzed software systems, together with their corresponding percentage of distribution among the zones. In all the software, the largest number of terms

[6]Each zone has a different vocabulary, so if the same lexeme appears in two or more zones we considered different the terms in all the zones where the lexeme appears.

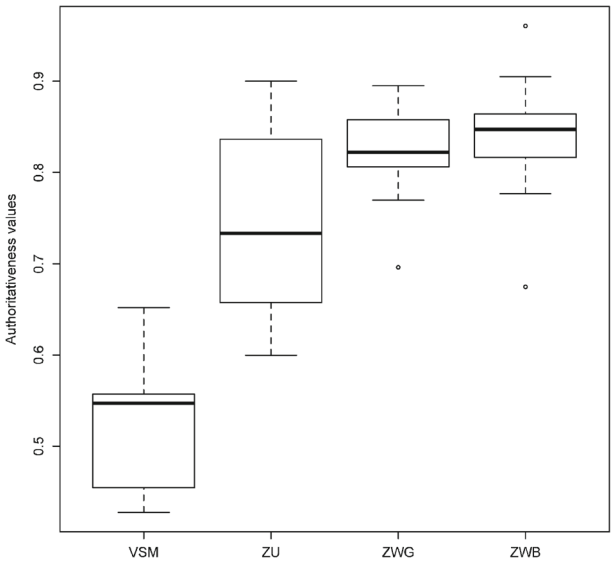

Fig. 4 Box plots of the authoritativeness values

came from SCS (with the only exception of Xalan), while the smallest from CN. Table 4 also shows that the second source for the terms is Co. Finally, the software with the largest number of terms is Liferay Portal (747,225).

By means of box plots, Fig. 4 shows the distributions for the authoritativeness values of VSM, ZU, ZWG, and ZWB. A quick analysis of the results indicates that both ZWG and ZWB outperform VSM and ZU. With regards to ZU, the use of unweighed zones allows improving authoritativeness values with respect to VSM.

Descriptive statistics are reported in Table 5. These statistics confirm that the use of the zones makes the difference in term of authoritativeness of partitions, especially when our probabilistic model is exploited.

5.2 RQ1-Hn1: Zones and Clustering Quality

The Mann-Whitney test[7] allows us to reject Hn1 (p-value < 0.001). Therefore, there is a statistically significant difference between the authoritativeness values obtained with and without considering the zones (VSM vs. ZU), and this difference is in favor of ZU (see Fig. 4 and Table 5). The effect size is large (0.832) and the statistical power is high (1). The observed average percentage improvement is 44%.

According to the data analysis results, we can positively answer RQ1: the introduction of the zones improves the quality of lexical-based clustering in the context of architecture recovery.

[7]The Shapiro-Wilk W test returned 0.049 and 0.063 as the p-values for VSM and ZU, respectively. This confirms our postulation on the distribution of data.

Table 5 Descriptive statistics of authoritativeness values for Technique

Factor	Min	Max	Median	Mean	StD
VSM	0.427	0.652	0.547	0.515	0.065
ZU	0.6	0.9	0.733	0.744	0.1
ZWG	0.696	0.895	0.822	0.826	0.045
ZWB	0.674	0.961	0.847	0.838	0.056

5.3 RQ2-Hn2: Unweighed vs. Weighed

The Mann-Whitney test[8] allows us to reject Hn2. Indeed there is a statistically significant difference between ZU and ZWG, with a p-value of 0.012 (less than α_B). The value for the point-biserial correlation r is 0.366 (i.e., medium), and the statistical power is 0.853. The average percentage improvement is 11 %. As for the difference between ZU and ZWB, the Mann-Whitney test returned 0.004 (less than α_B), the effect size is medium (0.427), and the statistical power is high (0.915). The obtained average percentage improvement is 13 %.

Therefore, we can also positively answer RQ2. Given this finding and the boost in clustering quality shown in Section 5.2, we can claim that the observed improvement is due to the combined use of the chosen zones and the probabilistic model.

5.4 RQ3-Hn3: Gaussian vs. Bernoulli

We did not reject Hn3 because we obtained 0.365 from the Mann-Whitney test as the p-value. The value for the point-biserial correlation r was 0.056 (i.e., small in favor of ZWB). The obtained β-value was 0.89, so indicating that the probability of erroneously non rejecting the null hypothesis Hn3 is high.

The answer to RQ3 is: the quality of the clustering is slightly affected by the type of the distribution chosen. The authoritativeness values obtained with ZWB are slightly better than those obtained with ZWG. The average percentage improvement is about 1%.

5.5 RQ4: Probabilistic Model vs. Upper Bound

Figure 5 shows the authoritativeness values for TUB, ZWG, and ZWB for each software system. This line plot shows that the authoritativeness values obtained with ZWG and ZWB are mostly close to TUB. This finding is further confirmed by the descriptive statistics of the differences between the authoritativeness values of TUB and ZWG and of TUB and ZWB, respectively. These statistics are reported in Table 6. We also graphically show these differences in Fig. 6. A further analysis indicated that the highest differences between TUB and ZWG and between TUB and ZWB were obtained for Liferay Portal (see the outliers on the top of both the boxes in Fig. 6). Among the analyzed software systems, Liferay Portal is the only one conceived to create web applications and it is the largest we analyzed. In addition, we also looked at the lexicons of the studied software to get further indications

[8]The Shapiro-Wilk W test returned 0.063 for ZU and 0.04 for ZWB. On the other hand, this test returned 0.098 for ZWG. Although the results Shapiro-Wilk W test suggest that a parametric test (e.g., unpaired t-test) could be used to verify the presence of a statistically significant difference between the values of authoritativeness obtained with ZWG and ZU, we used the Mann-Whitney test because repeated statistical tests were needed to test Hn2.

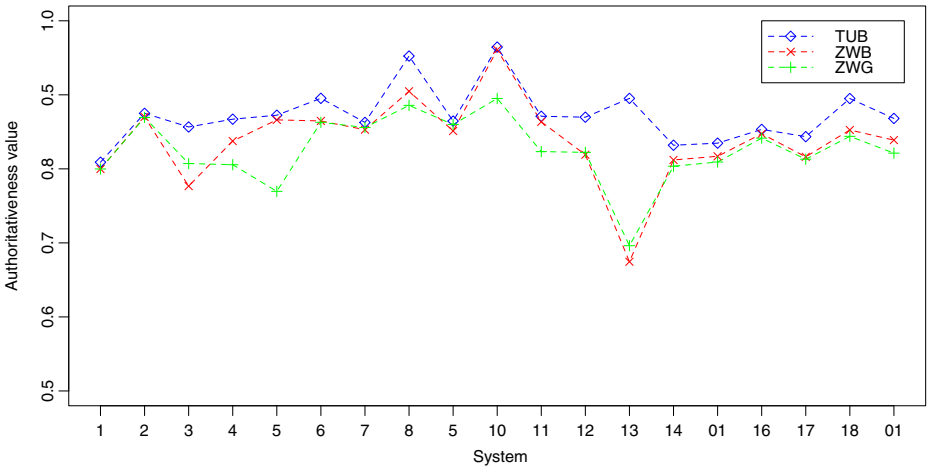

Fig. 5 Authoritativeness values for TUB, ZWB, and ZWG

to explain the obtained results. We noted that Liferay Portal is that with the worst lexicons, namely the names of identifiers (class and method names and variables) are single chars or are composed by English words shortened by applying contraction or abbreviation operations often without using any naming convention. In addition, source code comments very often contained identifier names.

Summarizing the authoritativeness values achieved by applying ZWB on all the software systems are slightly closer to their theoretic upper bounds. This confirms that ZWB slightly outperforms ZWG although this difference is not statistically significant. This is further corroborated by the sum of the differences for ZWG and ZWB that are 0.897 and 0.656, respectively.

5.6 Discussion

We observed that: (*i*) the use of zones improves clustering with respect to the use of a flat vocabulary and (*ii*) the use of weights computed by applying MLE and EM improves the quality of clustering. The observed improvements are statistically significant. Therefore, we can postulate that the terms placed in the considered zones convey information with different levels of relevance, and the use of this information improves the quality of clustering in the context of architecture recovery.

The use of a theoretical proof for the upper bound of the authoritativeness values allowed us to show that our probabilistic approach (both using Gaussian and Bernoulli distribution types) produces partitions that resemble the best possible partition. Therefore, the use of our

Table 6 Descriptive statistics of the differences

Comparison	Min	Max	Median	Mean	StD	Sum
TUB - ZWG	0.005	0.199	0.047	0.047	0.045	0.897
TUB - ZWB	0.004	0.221	0.02	0.035	0.049	0.656

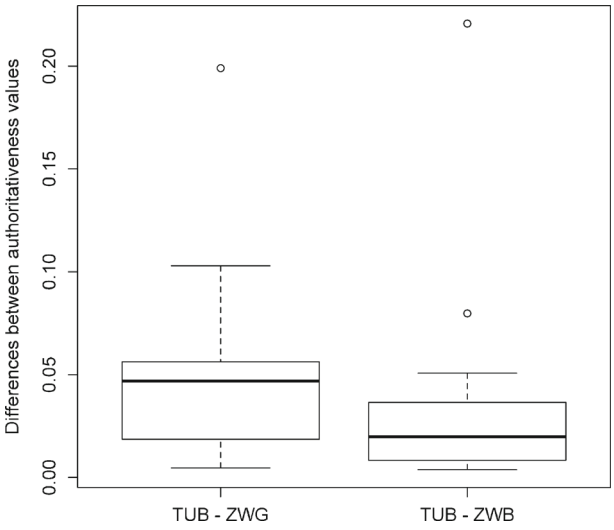

Fig. 6 Box plots of the differences between TUB and ZWB (*left*) and TUB and ZWG (*right*)

solution represents a viable alternative to manually assigning weights to the zones. This is one of the most important results of our research.

The partitions choosing Gaussian and Bernoulli distributions are almost similar as the obtained authoritativeness values show. This result suggests that the boost in the clustering quality is mostly related to both the zones and MLE and EM. That is, the quality of clustering is slightly affected by the type of the assumed distributions of the terms among the zones. Therefore, it is better to chose the model that converge faster, namely the model based on Bernoulli. It could be also possible that different kinds of assumed distributions produce better clustering results. Our findings pose the basis for future investigations.

As for the Gaussian distribution, we experimentally observed that better estimates for the weights of the zones were obtained by using a frequency approach (i.e., the rate between the number of terms in the zones and the total number of terms) to initialize the parameters of the model. The random choice of these parameters led to worse results (Corazza et al. 2011). A possible justification is that a zone that contains more lexical information can be more relevant for building the probabilistic model.

5.6.1 On the Quality of Lexicon

It is not surprising that our approach succeeds and fails with the lexicon quality of source code, since partitions are identified on the basis of identifier names and comments. Despite some threats about the generalizability of the results, we observed that when programmers choose good identifiers and properly comment source code, the quality of partitions improves. The results of our empirical investigation confirm with stronger evidence the findings by Kuhn et al. (2007).

The quality of the lexicon of some analyzed software systems is good (e.g., jHotDraw and jFreeChart), while it is worse in some other cases (e.g., Liferay Portal). In case of a good quality lexicon the differences between theoretical upper bounds and the authoritativeness values identified by our approach are very small (e.g., see Fig. 5). On the other hand, this difference increases when the quality of lexicon decreases. This outcome also holds for

VSM, where its authoritativeness values are still farther from the theoretical upper bound values. That is, our probabilistic model gives a boost in the authoritativeness values even if the quality of lexicon is not so good.

The results of a further analysis suggests that our approach seems to be a little sensitive to the number of lexemes in the source code. This pattern is different from that observed for other clustering approaches based on LSI (Deerwester et al. 1990): (*i*) the larger the number of the lexemes, the better clustering performs and (*ii*) the smaller the number of lexemes, the stronger the effect of the quality of terms on clustering is Kuhn et al. (2007) and Risi et al. (2012).

5.7 Implications

To judge the implications of our empirical assessment, we adopted a perspective-based approach (Basili et al. 1996). We base our discussion on the implications from the *practitioner/consultant* (simply *practitioner* in the following) and *researcher* perspectives (Kitchenham et al. 2008). These implications can be summarized as follows:

- The use of zones (although unweighed) yields an average improvement of clustering authoritativeness of about 44%. The effect of using the zones in architecture recovery is also statistically significant with respect to the use of a flat vocabulary, whose clustering quality is always worse than that with the zones (both weighed and unweighed). This confirms the idea that programmers place different care in choosing terms for different programming language constructs. This finding is relevant from the researcher perspective because it could be interesting to investigate how programmers choose terms (e.g., Scanniello et al. 2014; De Lucia et al. 2011). A deeper understanding on this point could allow the research to better exploit complementary information in source code, thus improving reverse engineering and program comprehension approaches. In other words, our findings go in the direction that future progresses in reverse engineering and program comprehension approaches will be achieved by exploring complementary information in source code.
- Although the approach proposed here is language independent, we performed an empirical assessment on software systems implemented in Java. Due to the magnitude of the benefits deriving from the use of the zones and due to similarities of Java with other object oriented programming languages (e.g., C# and C++), it could be possible that similar results could be obtained with software systems implemented with other object oriented programming languages. This result is practically relevant for the researcher, who could be interested in investigating whether or not there is a correlation between programming language and clustering results.
- Clustering quality seems to be not affected by the type of distributions chosen for EM. From the researcher perspective, this point is interesting in the context of software clustering. This outcome is also of interest for the researcher, who could plan future studies taking into account our outcomes.
- The authoritativeness values of the automatically identified partitions using EM are not far from those of the theoretical upper bound. In addition, reliability index of the maximum estimate is smaller than 0.3 in many cases. These two findings are relevant for the practitioner because the application of our probabilistic model allows him/her to maximize the improvements deriving from the use of the zones.
- Our approach seems to be a little sensitive to the size of the lexicon with respect to the clustering approaches based on LSI. This result is relevant for the practitioner, who

could use our approach without paying attention to the lexicon in the source code. Differently, the researcher could be interested in investigating what is the effect of the IR technique on clustering.
- The studied software systems have been developed in open source projects. The magnitude of the benefits deriving from the use of the zones and EM suggests that similar results could be also obtained on different kinds of software projects. This finding is relevant from both the practitioner and the researcher perspectives.
- The study is focused on desktop applications, development frameworks, tools, and libraries. From the researcher perspective, the effect of using our solutions on different kinds of applications (e.g., business and web based) represents a possible future direction.
- We observed that clustering quality is not affected by the kind of software. This is relevant for the practitioner, who could be interested in using our approach in the maintenance of the software marketed by his/her company.
- The obtained results seem to be independent from the size of the chosen software. Although we are not sure that the achieved results scale to very large software, the magnitude of the benefits stemming from the use of the zone and EM reassures us that the outcomes might be generalized also on this kind of software. This results is relevant for the practitioner.
- The use of our approach does not require deep changes in an interested company. In other words, the developers have not to change their habits in programming. They should only pay more attention to the lexicon to be used. This find can be considered relevant for the practitioner.
- The diffusion of a new technology/method is made easier when empirical evaluations are performed and their results show that such a technology/method solves actual issues (Pfleeger and Menezes 2000). This is why the results of our experimental evaluation could increase the diffusion of lexical clustering in the software industry. This concern is of particular interest for the practitioner.

6 Related Work

The definition of effective solutions for documenting software architectures is a longstanding and relevant research topic in the field of software maintenance and evolution (van Deursen et al. 2004; Koschke 2000; Tonella 2001; Kuhn et al. 2007; Ducasse and Pollet 2009; Maqbool and Babri 2007; Bavota et al. 2013a). In the literature, several terms have been used: modularization, remodularization, reverse architecting, or architecture extraction, reconstruction, mining, recovery, or discovery (Mendonça and Kramer 1996). A complete and extensive survey of the techniques and approaches that only use structural information is presented in Ducasse and Pollet (2009), while approaches that use both lexical and structural information are extensively discussed in Corazza et al. (2013) and Risi et al. (2012). For the sake of brevity, the description of the related literature is limited here to those techniques and approaches based on clustering and that exploit only lexical information in object oriented source code.

Lexical software clustering approaches are based on the idea that artifacts that contain similar lexemes are related. For example, Kuhn et al. (2007) propose an automatic and language independent approach for clustering software entities (e.g., classes or methods) based on LSI. The approach uses a flat vocabulary, whose lexemes are gathered from source code comments and identifier names. Then software entities are clustered according to their

similarity. A correlation matrix is used to identify how the clusters are related to each other. Several are the differences with respect to our work. The most remarkable differences are: the type of empirical evaluation used (qualitative vs. quantitative), the size of the used data set (2 vs. 19), and the IR techniques exploited (LSI vs. VSM). In addition, our work fills in a gap in that work because we explore complementary information in source code investigating the relevance of lexemes according to the zone where these lexemes appear in.

Scanniello et al. (2010) propose an approach that uses LSI to compare source code classes (implemented both in Java and in C++). The k-means clustering algorithm is then exploited to group software entities. Successively, Risi et al. (2012) analyzed the effect of using a variant of LSI on clustering results. Differently, from any other clustering approach, the authors also analyze the efficiency of the approach when performing software clustering. As we did here, the quality of a partition is assessed with respect to its resemblance with the package structure proposed by the original developers. Differently from us, MojoFM is not used to estimate this resemblance. This makes unfair any comparison between the approach by Risi et al. (2012) and ours. It is worth noting that we opted for MojoFM because it is considered better suited to estimate the authoritativeness of clustering (Shtern and Tzerpos 2011) with respect to any other measure based on Mojo (Tzerpos and Holt 1999). The main differences concern the used techniques to perform clustering and the performed empirical assessment. In addition, we explore here whether complementary information in source code might improve the quality of clustering in the context of architecture recovery.

The work by Corazza et al. (2010) is the first attempt to use a probabilistic model to weigh the relevance of lexemes in software clustering. That work has been successively extended by the same authors better characterizing the zones in source code and their relevance (Corazza et al. 2011). With respect to these two approaches, we extended here the probabilistic model introducing a new kind of distribution (i.e., the Bernoulli model) and performed a more exhaustive and rigorous empirical evaluation. With respect to Corazza et al. (2011), we used the K-Medoid algorithm with respect to a hierarchical clustering algorithm because we experimentally observed that authoritative values improved and because the chosen algorithm requires a lower number of tuning variables to be specified. In this paper, we also theoretically prove an upper bound for our approach. Another remarkable difference concerns the discussion of practical implications from both the researcher and professional perspectives. Finally, we delineate a number of possible future directions for lexical-based software clustering. This is one of the most important contributions of the research work presented in this paper.

7 Conclusion and Future Work

In this paper, we have presented an approach to partition object oriented software systems, by grouping classes on the basis of their lexical similarity. Differently from other approaches presented in the literature (e.g., Risi et al. 2012; Kuhn et al. 2005; Maqbool and Babri 2007), our solution separately considered the relevance of the lexical information embedded by programmers in six zones of source code, namely Class Names, Attribute Names, Method Names, Parameter Names Comments and Source Code Statements. The relevance of each zone is automatically weighed using a Maximum Likelihood Estimation, applied on a probabilistic model by means of the Expectation Maximization algorithm.

To assess the validity of our proposal, we have conducted an empirical assessment on 19 Java open source systems. The results indicated that the introduction of both the defined six zones and the probabilistic model significantly improves the clustering quality with respect

to the use of a flat vocabulary. Another remarkable result is that the quality of clustering obtained by exploiting the zones and by applying the probabilistic model is close to a theoretical upper bound we have proved. That is, our approach to weigh the contribution of the different zones allows to reach almost the maximum possible improvement in terms of clustering quality.

A possible future direction for our research could concern the involvement of actual developers in special conceived empirical assessment aimed at evaluating our technique with respect to a baseline, that is VSM without zone. An experimental procedure similar to that used by Bavota et al. (2013b) could be used. In the future, we also plan to apply our approach at granularity levels different from that proposed in this paper (i.e., method with respect to class) opportunely rethinking the zones in which lexemes appear. This new kind of clustering could be compared with respect to previously proposed techniques in the context of concept location and fault prediction (Scanniello and Marcus 2011; Scanniello et al. 2014; Revelle et al. 2011).

References

Ali N, Gueheneuc YG, Antoniol G (2011) Requirements traceability for object oriented systems by partitioning source code. In: Proceedings of working conference on reverse engineering. IEEE Computer Society, pp 45–54

Andritsos P, Tzerpos V (2005) Information-theoretic software clustering. IEEE Trans Softw Eng 31(2):150–165

Anquetil N, Fourrier C, Lethbridge TC (1999) Experiments with clustering as a software remodularization method. In: Proceedings of working conference on reverse engineering. IEEE Computer Society, Washington, pp 235–255

Basili VR, Green S, Laitenberger O, Lanubile F, Shull F, Sørumgård LS, Zelkowitz MV (1996) The empirical investigation of perspective-based reading. Empir Softw Eng 1(2):133–164

Bavota G, De Lucia A, Marcus A, Oliveto R (2010) Software re-modularization based on structural and semantic metrics. In: Proceedings of international working conference on reverse engineering. IEEE Computer Society, pp 195–204

Bavota G, De Lucia A, Marcus A, Oliveto R (2013a) Using structural and semantic measures to improve software modularization. Empir Softw Eng 18(5):901–932

Bavota G, Dit B, Oliveto R, Penta MD, Poshyvanyk D, Lucia AD (2013b) An empirical study on the developers' perception of software coupling. In: Proceedings of international conference on software engineering. IEEE / ACM, pp 692–701

Bavota G, Gethers M, Oliveto R, Poshyvanyk D, De Lucia A (2014a) Improving software modularization via automated analysis of latent topics and dependencies. ACM Trans Softw Eng Methodol 23(1): 4:1–4:33. doi:10.1145/2559945

Bavota G, Oliveto R, Gethers M, Poshyvanyk D, De Lucia A (2014b) Methodbook: Recommending move method refactorings via relational topic models. IEEE Trans Softw Eng 40(7):671–694

Binkley D (2007) Source code analysis: a road map. In: Future of software engineering. IEEE Computer Society, pp 104–119

Bishop C (2006) Pattern recognition and machine learning. Information science and statistics. Springer

Bittencourt RA, Guerrero DDS (2009) Comparison of graph clustering algorithms for recovering software architecture module views. In: Proceedings of the European conference on software maintenance and reengineering. IEEE Computer Society, pp 251–254

Conover WJ (1998) Practical nonparametric statistics, 3rd. Wiley

Corazza A, Di Martino S, Maggio V, Moschitti A, Passerini A, Scanniello G, Silvestri F (2013) Using machine learning and information retrieval techniques to improve software maintainability. In: Eternal systems, communications in computer and information science. Springer, Berlin. In Press

Corazza A, Di Martino S, Maggio V, Scanniello G (2011) Investigating the use of lexical information for software system clustering. In: Proceedings of European conference on software maintenance and reengineering. IEEE Computer Society, pp 35–44

Corazza A, Di Martino S, Scanniello G (2010) A probabilistic based approach towards software system clustering. In: Proceedings of European conference on software maintenance and reengineering. IEEE Computer Society, pp 89–98
De Lucia A, Di Penta M, Oliveto R (2011) Improving source code lexicon via traceability and information retrieval. IEEE Trans Softw Eng 37(2):205–227
De Lucia A, Di Penta M, Oliveto R, Panichella A, Panichella S (2012) Using ir methods for labeling source code artifacts: is it worthwhile? In: Proceedings of international conference on program comprehension. IEEE Computer Society Press, pp 193–202
De Lucia A, Risi M, Scanniello G, Tortora G (2009) An investigation of clustering algorithms in the comprehension of legacy web applications. J Web Eng 8(4):346–370
Deerwester SC, Dumais ST, Landauer TK, Furnas GW, Harshman RA (1990) Indexing by latent semantic analysis. J Am Soc Inf Sci 41(6):391–407
Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J Roy Stat Soc Ser B 39(1):1–38
van Deursen A, Hofmeister C, Koschke R, Moonen L, Riva C (2004) Symphony: view-driven software architecture reconstruction. In: Proceedings of working conference on software architecture, pp 122–134
Ducasse S, Pollet D (2009) Software architecture reconstruction: a process-oriented taxonomy. IEEE Trans Softw Eng 35(4):573–591. doi:10.1109/TSE.2009.19
Eastwood A (1993) Firm fires shots at legacy systems. Comput Canada 19(2):17
Erlikh L (2000) Leveraging legacy system dollars for e-business. IT Professional 2:17–23
Flach P (2012) Machine learning: the art and science of algorithms that make sense of data. Cambridge University Press
Freund RJ, Wilson WJ (2003) Statistical methods, 2nd edn. Academic Press
Grubb P, Takang AA (2003) Software maintenance: concepts and practice, 2nd edn. World Scientific
Jarzabek S (2007) Effective software maintenance and evolution—a reuse-based approach. Auerbach Publ
Kampenes V, Dyba T, Hannay J, Sjoberg I (2006) A systematic review of effect size in software engineering experiments. Inf Softw Technol 49(11–12):1073–1086
Kaufman L, Rousseeuw P (1990) Finding groups in data an introduction to cluster analysis. Wiley Interscience
Kevin Freedman JB (1999) Current concepts review - sample size and statistical power in clinical orthopaedic research. J Bone Joint Surg 81:1454–60
Kitchenham B, Al-Khilidar H, Babar M, Berry M, Cox K, Keung J, Kurniawati F, Staples M, Zhang H, Zhu L (2008) Evaluating guidelines for reporting empirical software engineering studies. Empir Softw Eng 13(1):97–121
Koschke R (2000) Atomic architectural component recovery for program understanding and evolution. Ph.D. thesis, University of Stuttgart
Kuhn A, Ducasse S, Girba T (2005) Enriching reverse engineering with semantic clustering. In: Proceedings of international working conference on reverse engineering. IEEE Computer Society, pp 133–142. doi:10.1109/WCRE.2005.16
Kuhn A, Ducasse S, Gîrba T (2007) Semantic clustering: Identifying topics in source code. Inf Softw Technol 49(3):230–243
Liu Y, Poshyvanyk D, Ferenc R, Gyimóthy T, Chrisochoides N (2009) Modeling class cohesion as mixtures of latent topics. In: Proceedings of international conference on software maintenance. IEEE Computer Society, pp 233–242
Mahdavi K (2005) A clustering genetic algorithm for software modularisation with a multiple hill climbing approach. Ph.D. thesis, Department of Information Systems and Computing, Brunel University
Maletic JI, Marcus A (2001) Supporting program comprehension using semantic and structural information. In: Proceedings of international conference on software engineering. IEEE Computer Society, Washington, pp 103–112
Manning CD, Raghavan P, Schütze H (2008) Introduction to information retrieval. Cambridge University Press, New York
Maqbool O, Babri H (2007) Hierarchical clustering for software architecture recovery. IEEE Trans Software Eng 33(11):759–780
Marcus A, Poshyvanyk D (2005) The conceptual cohesion of classes. In: International conference on software maintenance. IEEE Computer Society, pp 133–142
Mashiko Y, Basili V (1997) Using the GQM paradigm to investigate influential factors for software process improvement. J Syst Softw 36(1):17–32
McCallum A, Nigam K (1998) A comparison of event models for naive bayes text classification. In: Proceedings of workshop on learning for text categorization. AAAI Press, pp 41–48
Mclachlan J, Krishnan T (1996) The EM algorithm and extensions. Wiley Inter-science

Mendonça NC, Kramer J (1996) Requirements for an effective architecture recovery framework. In: Joint proceedings of the second international software architecture workshop and international workshop on multiple perspectives in software development. ACM, pp 101–105. doi:10.1145/243327.243620
Mitchell TM (1997) Machine learning, 1st edn. McGraw-Hill, Inc., New York
Pfleeger SL, Menezes W (2000) Marketing technology to software practitioners. IEEE Softw 17:27–33
Port O (1998) The software trap – automate or else. Bus Week 9(3051):142–154
Porter MF (1997) An algorithm for suffix stripping. Morgan Kaufmann Publishers Inc., San Francisco, pp 313–316
Poshyvanyk D, Marcus A (2006) The conceptual coupling metrics for object-oriented systems. In: Proceedings of international conference on software maintenance. IEEE Computer Society, pp 469–478
Press WH, Teukolsky SA, Vetterling WT, Flannery BP (1992) Numerical recipes in C, the art of scientific computing, 2nd edn. Cambridge University Press
Reggio G, Ricca F, Scanniello G, Di Cerbo F, Dodero G (2011) A precise style for business process modelling: Results from two controlled experiments. In: Proceedings of model driven engineering languages and systems, lecture notes in computer science. Springer, pp 138–152
Revelle M, Gethers M, Poshyvanyk D (2011) Using structural and textual information to capture feature coupling in object-oriented software. Empir Softw Eng 16(6):773–811
Risi M, Scanniello G, Tortora G (2012) Using fold-in and fold-out in the architecture recovery of software systems. Formal Asp Comput 24(3):307–330
Romano S, Scanniello G, Risi M, Gravino C (2011) Clustering and lexical information support for the recovery of design pattern in source code. In: Proceedings of international conference on software maintenance. IEEE Computer Society, pp 500–503
Romesburg H (2004) Cluster analysis for researchers. Lulu Press. http://books.google.it/books?id=ZuIPv7OKm10C
Salton G, Wong A, Yang CS (1975) A vector space model for automatic indexing. Commun ACM 18(11):613–620. doi:10.1145/361219.361220
Saw JG, Yang MCK, Mo TC (1984) Chebyshev inequality with estimated mean and variance. Am Stat 38(2):130–132
Scanniello G, D'Amico A, D'Amico C, D'Amico T (2010) Using the Kleinberg algorithm and Vector Space Model for software system clustering. In: Proceedings of international conference on program comprehension. IEEE Computer Society, pp 180–189
Scanniello G, Gravino C, Marcus A, Menzies T (2013) Class level fault prediction using software clustering. In: Proceedings of international conference on automated software engineering. IEEE / ACM, pp 640–645
Scanniello G, Marcus A (2011) Clustering support for static concept location in source code. In: Proceedings of international conference on program comprehension. IEEE Computer Society, pp 1–10
Scanniello G, Marcus A, Pascale D (2014) Link analysis algorithms for static concept location: an empirical assessment. Empir Softw Eng 1–55. doi:10.1007/s10664-014-9327-7
Scanniello G, Risi M, Tortora G (2010) Architecture recovery using latent semantic indexing and k-means: an empirical evaluation. In: Proceedings of international conference on software engineering and formal methods. IEEE Computer Society, pp 103–112
Shapiro S, Wilk M (1965) An analysis of variance test for normality. Biometrika 52(3–4):591–611
Shtern M, Tzerpos V (2011) Evaluating software clustering using multiple simulated authoritative decompositions. In: Proceedings of international conference on software maintenance. IEEE Computer Society, pp 353–361
Tonella P (2001) Concept analysis for module restructuring. IEEE Trans Softw Eng 27(4):351–363. doi:10.1109/32.917524
Tzerpos V, Holt RC (1999) Mojo: A distance metric for software clusterings. In: Proceedings of the working conference of reverse engineering, pp 187–193
Wen Z, Tzerpos V (2004) An effectiveness measure for software clustering algorithms. In: Proceedings of international conference on program comprehension. IEEE Computer Society, pp 194–203
Wiggerts TA (1997) Using clustering algorithms in legacy systems remodularization. In: Proceedings of working conference on reverse engineering. IEEE Computer Society, Washington, pp 33–43
Wohlin C, Runeson P, Höst M, Ohlsson M, Regnell B, Wesslén A (2000) Experimentation in software engineering - an introduction. Kluwer
Wu J, Hassan AE, Holt RC (2005) Comparison of clustering algorithms in the context of software evolution. In: Proceedings of international conference on software maintenance. IEEE Computer Society, pp 525–535

Anna Corazza is assistant professor at the Department of Electrical Engineering and Information Technologies at the University of Naples Federico II in Italy. She obtained the Laurea Degree in Electronic Engineering and the PhD degree at the University of Padua. From 1990 to 2000, she worked as researcher at FBK in Trento and afterwards at the University of Milan. Her research interests focus on statistical approaches to natural language processing, bioinformatics, software engineering and information retrieval.

Sergio Di Martino received the PhD in Computer Science from the University of Salerno (Italy) in 2005. Since 2007 he is Assistant Professor at University "Federico II" in Naples. He has served as consultant for many Research Centres, especially in the automotive domain. His research interests focus on knowledge discovery and visualization from complex datasets, such as software repositories. He has published more than 80 papers on these topics in international journals, books, and conference proceedings.

Valerio Maggio received his Master degree in Computer Science from the University of Naples, "Federico II", in 2009. He also received the Ph.D. in Computational Science and Informatics from the University of Naples (Italy), in 2013. He served as a Research Fellow in the Dept. of Electrical Engineering and Information Technologies at the University of Naples "Federico II" for eight months. He then joined the Dept. of Information and Electrical and Applied Mathematics at the University of Salerno, in 2014, where he is currently employed as a Post Doc Researcher. His research interests focus on empirical software engineering, software maintenance, information retrieval, and machine learning. He is a member of the IEEE Computer Society.

Giuseppe Scanniello received his Laurea and Ph.D. degrees, both in Computer Science, from the University of Salerno, Italy, in 2001 and 2003, respectively. In 2006, he joined the Department of Mathematics and Computer Science, University of Basilicata, Potenza, Italy, where he is currently an assistant professor and leads the Software Engineering group. Recently, he got the national qualification as an Associate professor for the scientific disciplinary sectors 01/B1 (Computer Science) and 09/H1 (Information Processing Systems). His research interests include requirements engineering, empirical software engineering, reverse engineering, reengineering, software visualization, workflow automation, migration, wrapping, integration, e-learning, global software engineering, cooperative supports for software engineering, and visual languages. He has published more than 130 papers in international journals, books, and proceedings of refereed conferences. He serves on the organizing and program committees of several major international conferences in the field of software engineering. He is a member of the IEEE Computer Society.