HESA: The Construction and Evaluation of Hierarchical Software Feature Repository

Yue Yu, Huaimin Wang, Gang Yin, Xiang Li, Cheng Yang
National Key Laboratory for Parallel and Distributed Processing
School of Computer Science, National University of Defense Technology
Changsha, China
yuyue_whu@foxmail.com, whm_w@163.com, {jack.nudt, shockleylee}@gmail.com

***Abstract*—Nowadays, the demand for software resources on different granularity is becoming prominent in software engineering field. However, a large quantity of heterogeneous software resources have not been organized in a reasonable and efficient way. Software features, a kind of important knowledge for software reuse, are ideal materials to characterize software resources. Our preliminary study shows that the effectiveness of feature-related tasks, such as software resource retrieval and feature location, will be greatly improved, if a multi-grained feature repository is available. In this paper, we construct a *Hierarchical rEpository of Software feAture* (HESA), in which a novel hierarchical clustering approach is proposed. For a given domain, we first aggregate a large number of feature descriptions from multiple online software repositories. Then we cluster these descriptions into a flexible hierarchy by mining their hidden semantic structures. Finally, we implement an online search engine on HESA and conduct a user study to evaluate our approach quantitatively. The results show that HESA can organize software features in a more reasonable way compared to the classic and the state-of-the-art approaches.**

***Keywords*—Software reuse; Mining Software repository; Feature-ontology; Clustering;**

I. Introduction

Software reuse is widely recognized as an effective way to increase the quality and productivity of software [1]. With the development of software industry, the degree of software reuse is deeper than previous years and the demand for resources on different granularity becomes more prominent. For example, when developing a new large software system, we may reuse some API calls from the third party to accomplish the core functions and the mature open source software as the basic framework. Additionally, some code fragments or components can be reused to meet other additional demands. The reusable resources are multi-grained, consisting of API calls (the finest level of granularity), code fragments, components (higher than API calls) and software systems (much higher than others).

However, considering the large-scale, heterogeneous and multi-grained software resources, it is a great challenge for stakeholders to retrieve the suitable one. With the evolution of open source ecosystems, more than 1.5 million open source software projects are now hosted in open source communities [2]. Reusable resources [3] are manifold, including code bases, execution traces, historical code changes, mailing lists, bug databases and so on. All of these valuable resources have not been reorganized in a reasonable and efficient way to assist in the activities of software development.

As a kind of attributes which capture and identify commonalities and differences in a software domain, software feature [4] [5] is an ideal material to characterize the software resources. Constructing a feature repository of a flexible structure can make a great contribution to multi-grained reuse.

However, classic feature analysis techniques, such as Feature Oriented Domain Analysis (FODA) [6] and Domain Analysis and Reuse Environment (DARE) [7], are heavily relied on the experience of domain experts and plenty of market survey data. Hence, the feature analysis is a labor-intensive and error-prone process.

In recent years, more and more stakeholders develop, maintain and share their software products on the Internet. In order to promote their products to users, project managers write some market-oriented summaries, release notes and feature descriptions on the profile pages via natural language. The large number of online software profiles can be treated as a kind of repository containing a wealth of information about domain-specific features. Although researchers propose several automatic methods to mine features from the web repository [8] [9] [10], the problems have not completely be solved, specifically in organizing features as flexible granularity.

In this paper, we are trying to address the above problems by proposing a novel approach to construct a *Hierarchical rEpository of Software feAture* (HESA). First of all, we extract a massive number of feature descriptions from online software profiles and mine their hidden semantic structure by probabilistic topic model. Then, we present an *improved Agglomerative Hierarchical Clustering* (iAHC) algorithm, seamlessly integrated with the topic model, to build the feature-ontology of HESA. Finally, we implement an online search engine[1] for HESA to help retrieve features in a multi-grained manner, which can support multiple reuse requirements. By conducting a user study, we demonstrate the effectiveness of our system with quantitative evaluations comparing to the classic and the state-of-the-art approaches.

The rest of this paper is or organized as follows. Section II introduces the overview of our work. Section III describes how to construct HESA in detail. Experiments and analysis can be found in Section IV. Finally, we present some related

[1]http://influx.trustie.net

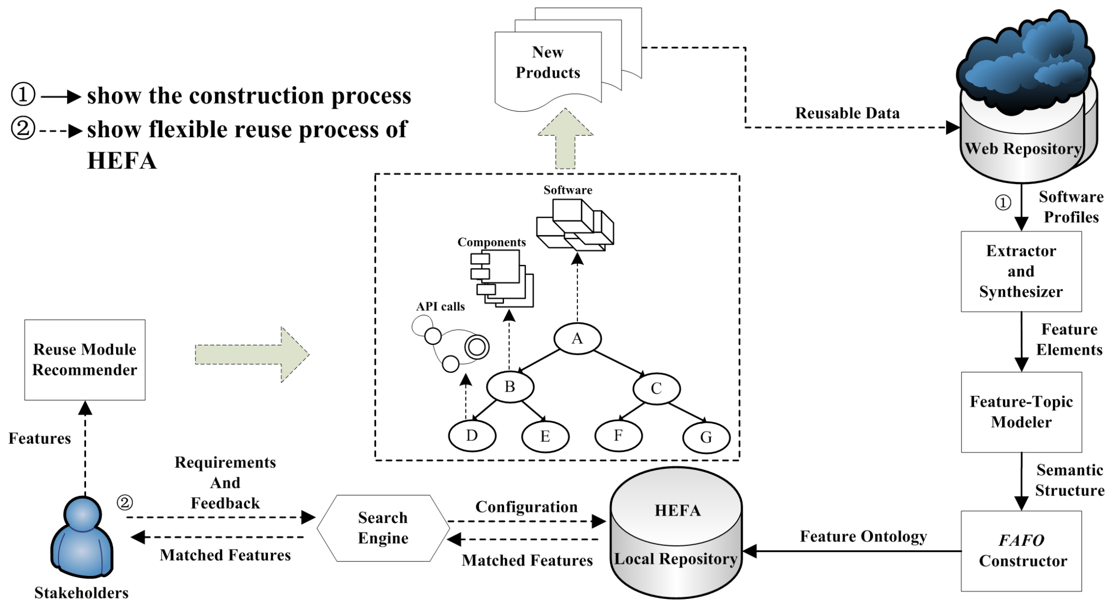


Figure 1. Overview of the construction and use of HESA

work in Section V and draw our conclusions in Section VI.

II. Approach Overview

First of all, we give the definitions of some concepts used in this paper.

Feature element: Feature element is a kind of raw descriptions which can indicate a functional characteristic or concept of the software product.

Feature: Feature is an identifier of the cluster about feature elements, where the cluster is an intermediate output of *improved Agglomerative Hierarchical Clustering* (iAHC).

Feature-ontology: Feature-ontology is a kind of hierarchical structure induced from feature elements by iAHC.

HESA: The assembly of all feature-ontologies of the different categories is the *Hierarchical rEpository of Software feAture*.

The objective of this paper is to build a hierarchical structure of feature as a flexible granularity. In the top layers, the features in coarse granularity may be mapped to the corresponding software resources such as mature applications, design patterns, and superclasses. In the bottom layers, the features can be mapped to some related code fragments, API calls and subclasses.

Before describing the specific details of the underlying algorithms, an architectural overview of approach will be provided as below. There are actually two processes concerning the application of our method, i.e., the construction process and the use process of HESA by stakeholders. In this paper, we only focus on the construction process owing to space limitations.

As depicted in Figure 1, the construction process consists of three primary modules and the input is software profiles data collected and updated continuously by a web crawler. The first module is called the *Extractor and Synthesizer*. We use the Extractor component to extract feature elements. Then, after running preprocessing tasks, the Synthesizer component will automatically located these feature elements, into a unified category. Especially, the word "domain" will be replaced by "category" for they are sharing the same meaning in the rest of this paper.

The second module, the *Feature-Topic Modeler*, is responsible for mining the semantic structure hidden in feature elements. We will merge the synonymic feature elements in terms of their semantic structure in the next step.

The last module, the *FAFO (Flexible grAnularity Feature-Ontology) Constructor*, is a critical part of the construction process. In this module, we present a novel algorithm iAHC to construct the feature-ontology and more details can be found in Section III. The major functionalities of this module are listed as below:

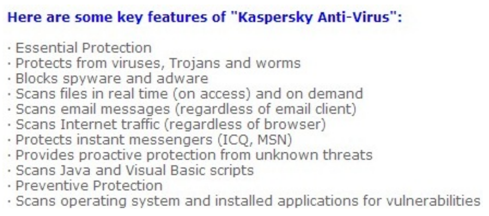
(1) The synonymic feature elements are merged based on the semantic structure outputted by the Feature-Topic Modeler;

(2) For each cluster, a significant group of feature elements is selected as the medoid used to generate feature;

(3) A feature-ontology is learned and features can be retrieved in terms of flexible granularity.

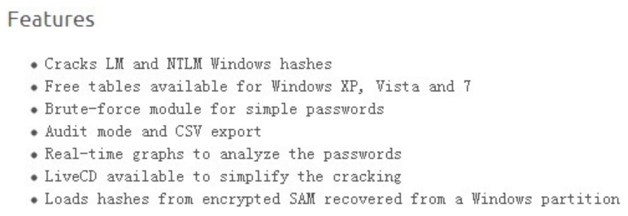
After all the raw data under our category are disposed, the construction process of HESA is finished.

The HESA can perfectly support the multi-grained resource reuse. For example, when a company plans to enter a new domain such as Antivirus, the stakeholders want to know a few general features about this domain, e.g. *"Anti-rootkit"*, *"Heuristic scanning"*, and *"File backup"*. Inputting requirements and use the search engine of HESA, the matched features will be returned. Based on the feature in coarse granularity, they can find some mutual software systems. Furthermore, to know this field more clearly and locate some reusable code fragments or packages, some fine granularity


Here are some key features of "Kaspersky Anti-Virus":

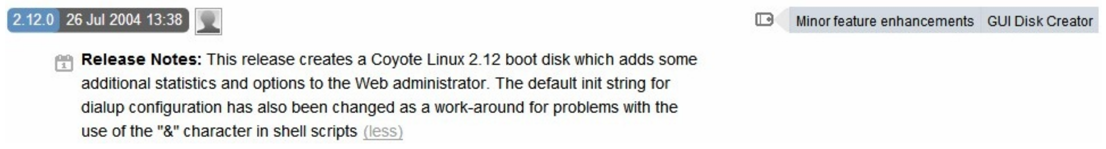
· Essential Protection
· Protects from viruses, Trojans and worms
· Blocks spyware and adware
· Scans files in real time (on access) and on demand
· Scans email messages (regardless of email client)
· Scans Internet traffic (regardless of browser)
· Protects instant messengers (ICQ, MSN)
· Provides proactive protection from unknown threats
· Scans Java and Visual Basic scripts
· Preventive Protection
· Scans operating system and installed applications for vulnerabilities

(a) Bullet-point lists of features in *Softpedia.com*


Features

- Cracks LM and NTLM Windows hashes
- Free tables available for Windows XP, Vista and 7
- Brute-force module for simple passwords
- Audit mode and CSV export
- Real-time graphs to analyze the passwords
- LiveCD available to simplify the cracking
- Loads hashes from encrypted SAM recovered from a Windows partition

(b) Bullet-point lists of features in *Sourceforge.com*


2.12.0 26 Jul 2004 13:38

Minor feature enhancements GUI Disk Creator

Release Notes: This release creates a Coyote Linux 2.12 boot disk which adds some additional statistics and options to the Web administrator. The default init string for dialup configuration has also been changed as a work-around for problems with the use of the "&" character in shell scripts (less)

(c) Release notes in *Freecode.com*

Figure 2. Examples of feature elements in the software pages

features can be retrieved from HESA, e.g., *"Automatic detection of downloaded files and Lock Autorun.inf, virus cannot execute."*

III. Construction of HESA

A. Feature Elements in Software profiles

The online software profiles contains a wealth of information about domain-specific features. In this paper, all the feature elements are extracted from software profiles in Softpedia.com[2], Freecode.com[3] and Sourceforge.com[4]. As depicted in Figure 2(a), there is a bullet-point list of some key features about the software resource in Softpedia.com and Sourceforge.com. Another type of raw descriptions being used is the Release Notes in Freecode.com. A product has many release versions about bug fixes, performance optimizations and feature enhancements. As depicted in Figure 2(b), we extracted feature elements from the release notes about feature enhancement, which contain some related tags or key words, such as "add", "support for" and "new feature".

To allocate different feature elements, which extracted from three different websites, into a unified domain category, the two categories of Softpedia.com and Sourceforge.com was combined into a new one. Then, all software and their feature elements are automatically classified into the unified category according to softwares tags or descriptions using the method of paper [2].

B. Feature Element Analysis

Because different people describe the functions in terms of their personal understanding in an open environment, feature elements are unstructured and disordered. To illustrate these problem clearly, the feature elements in Antivirus category are used as examples in this paper.

[2]http://www.softpedia.com
[3]http://freecode.com
[4]http://sourceforge.net

Hybrid Semantic-level: The problem of hybrid semantic-level is that different feature elements describe a common theme in different semantic level, such as the following descriptions:

(1) *"Email Scanner enhanced email protection"*;

(2) *"Email scanning for Microsoft Outlook, Outlook Express, Mozilla Thunderbird, Windows Live Mail, Windows Mail, and other POP3/IMAP mail clients, ensuring your email is free of viruses and other threats"*;

(3) *"Blocks spam mails, phishing attack mails, junk mails and porn mails before they reach your inbox"*;

The first sentence describes the theme of email protection in a general level. However, the last two sentences present more details including what type of mail clients would be supported and what kind of message would be filtered.

According to sampling statistics of our datasets, there are 25.7% feature elements in a relative high semantic-level, 33.9% feature elements in a relative specific semantic-level, and 40.4% in the intermediate-level.

On one hand, the massive number of feature elements in different semantic-level are good materials for the construction of flexible granularity ontology. On the other hand, it is a great challenge for the traditional methods to cluster and reorganize feature elements.

Synonymic Feature Element: The problem of synonymic feature element happens when two features are used to describe some common or very similar functional attributes. Some feature elements are almost the same with each other, such as the four feature elements below:

(1) *"Kills the core of AdPower and not only symptoms"*;
(2) *"Kills the core of BANCOS.D and not only symptoms"*;
(3) *"Kills the core of Dyfuca and not only symptoms"*;
(4) *"Kills the core of eBot and not only symptoms"*;

The difference between these feature elements is the name of the malicious code, such as "AdPower", "BANCOS.D" and "Dyfuca". However, all of them present a common functional attribute about the ability of killing various popular viruses.

Another typical problem is that each pair only shares few core words, such as the following:

(1) *"Ability to update that does not require downloading full package"*;

(2) *"Incremental database updates and often to include information about latest threats"*;

(3) *"Incremental updating system minimizes the size of regular update files"*;

These three feature elements present the common attribute about incremental updating, but only the word "***update***" is shared by the two sentences. According to the statistics, there are 33.7% of synonymic feature elements in Antivirus category, 28.9% in File-manger category, and 41.6% in Audio-Player category. Thus, feature elements should be merged together by an effective method.

***Latent Semantic Structures**:* According to our observation, one feature element may relate to several specific topics. Take five feature elements of *Mozilla Firefox*[5] and three topics of "*Browse with Security*", "*Protect your Privacy*" and "*Easy to Use*" as an example. Figure 3 illustrates the relationship among these feature elements and topics. Each feature element connects with one or two topics and this connection can be exposed by the key words or phrases. For example, the feature element "*Control the level of scrutiny you'd like Firefox to give a site with a variety of customized settings*" is related to the topics of "*Browse with Security*" and "*Protect your Privacy*". The phrases "*Control the level of*" and "*scrutiny*" reflects that it is possible to associate with the topic of security, and "*you'd like*" and "*customized settings*" reflects the relevancy with the topic about user experience. The relationship between topic and feature element is a kind of latent semantic structures which is useful for the clustering of feature element and the construction of feature-ontology.

C. Feature-Topic Model

***Problem Formalization**:* In a specific category, such as Antivirus, all the feature elements in the corpus can be represented as $\mathbf{F}_m = \{f_1, f_2, \ldots, f_i, \ldots, f_m\}$, where f_i denotes the ith feature elements in the corpus. Assuming that K latent topics $\mathbf{T}_k = \{t_1, t_2, \ldots, t_j, \ldots, t_k\}$ are implicit in the feature elements, where t_j denotes the jth topic. Although a feature element can be bound up with several topics, it may put more emphasis on some topics than the others. The topic degree within feature element f_i can be represented as a K-dimensional vector $\boldsymbol{v}_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,j}, \ldots, p_{i,k})$, where $p_{i,j}$ is a topic weight describing the extent to which the topic t_j appears in feature element f_i. When $p_{i,j} = 0$, f_i is irrelevant to t_j. Thus, the $\boldsymbol{v}_i, i \in [1, m]$, represented by $\mathbf{V}_m$, can be used to indicate the semantic structure implied in feature elements. If the $\mathbf{V}_m$ can be obtained, the thematic similarity measure would be induced for each pair of feature elements and the synonymic feature elements would be merged together. Because topic models answer what themes or topics a document relates to and quantify how strong such relations are, it is a effective way to learn $\mathbf{V}_m$.

[5]http://www.mozilla.org/en-US/firefox/security

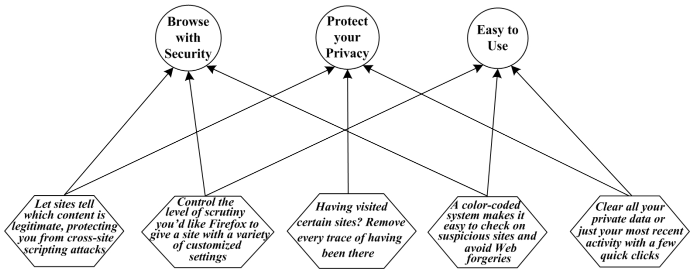

Browse with Security
Protect your Privacy
Easy to Use
Let sites tell which content is legitimate, protecting you from cross-site scripting attacks
Control the level of scrutiny you'd like Firefox to give a site with a variety of customized settings
Having visited certain sites? Remove every trace of having been there
A color-coded system makes it easy to check on suspicious sites and avoid Web forgeries
Clear all your private data or just your most recent activity with a few quick clicks

Figure 3. Feature elements mapping to topics

***Topic Modeling Technique**:* A topic model provides a means to automatically analysis the semantic structures within unstructured and unlabeled documents. In this paper, we choose Latent Dirichlet Allocation (LDA) [11] because it has been shown to be more effective for a variety of software engineering purposes [12] [13] than other topic models like LSI. In LDA, each word w_i in a document d is generated by sampling a topic z from *document-topic* distribution, and then sampling a word from *topic-word* distribution. More formally, a latent topic $z = j$ is modeled as an unlabeled *topic-word* distribution $\phi^{(j)} = P(w|z = j)$, which was drawn from a dirichlet prior distribution $\mathbf{Dirichlet}(\beta)$. The number of topics K is specified beforehand to adjust the granularity. Each document d is a mixture of topics $\theta^{(d)} = P(z)$ with a dirichlet prior distribution $\mathbf{Dirichlet}(\alpha)$. The generative process of each word in d is an essentially draw from the joint distribution: $P(w_i) = \sum_{i=1}^{K} P(w_i|z_i = j)P(z_i = j)$. Given the observed documents, Gibbs Sampling algorithm [14] is widely used for posterior inference. Finally, the *word-topic* $\boldsymbol{\phi}$ and topic-document $\boldsymbol{\theta}$ distribution can be approximated.

However, document is a generalized concept which can be any textual resource. In this paper, a feature element f_i can be viewed as a document which is preprocessed by removing commonly occurring words and then by stemming the remaining words to their root form. According to category, we apply LDA to process the documents using the MALLET tool [15] which is an implementation of the Gibbs sampling algorithm. Then, the *topic-feature* distribution $\mathbf{V}_m$ can be trained, which is the same as $\boldsymbol{\theta}$.

D. iAHC : improved Agglomerative Hierarchical Clustering

To support multi-grained reuse environment, the semantic similar feature elements should be merged and reorganized as a flexible hierarchical structure defined as feature-ontology. In this paper, we present an iAHC algorithm (**Algorithm 1**) integrated with the LDA.

Initially, every feature elements is a distinct cluster. Line 4-7 finds the closest two clusters c_i and c_j in the current cluster set M, and merge them into a new cluster c and update M. The proximity used to measure the distance between every two clusters, defined as below:

$$proximity(c_i, c_j) = \frac{\sum_{\substack{fi \in c_i \\ fj \in c_j}} similarity(fi, fj)}{|c_i| \times |c_j|} \quad (1)$$

Algorithm 1 improved Agglomerative Hierarchical Clustering

```
Require:
    $\mathbf{F}_m = \{f_1, f_2, \dots, f_i, \dots, f_m\}$;
    feature-topic distribution $\mathbf{V}_m$;
Ensure:
    The construction of feature-ongtology;
 1: $M \leftarrow D$
 2: $featureSet \leftarrow \emptyset$
 3: repeat
 4:     $\langle c_i, c_j \rangle = \mathbf{findTwoClosestClusters}(M)$
 5:     merge $c_i$ and $c_j$ as $c$
 6:     delete $c_i$ and $c_j$ from $M$
 7:     add $c$ to $M$
 8:     $centroid = \mathbf{calculateCentroid}(c)$
 9:     for $c_i \in c$ do
10:         $value_s = \mathbf{Similarity}(c_i, centroid)$
11:         $degree_t = \mathbf{calculateTopicDegree}(c_i)$
12:         $score_m = \kappa \times value_s + \lambda \times degree_t$
13:         add $score_m$ to $MedoidScore$
14:     end for
15:     $medoid_C = \mathbf{findMaximumScores}(MedoidScore)$
16:     $score_F = \mathbf{Similarity}(medoid_C)$
17:     $feature_C = \mathbf{mergeMedoid}(medoid_C, score_F)$
18:     $\mathbf{saveFeaturetoHESA}(M, feature_C)$
19: until $|M| = 1$
```

Where $c_i, c_j \subseteq \mathbf{F}_m$, $similarity(f_i, f_j)$ used to calculate the divergence between any two data point. Based on LDA, the divergence can be understood as the thematic space coordinate distance between the two feature elements. There are several ways to calculate the divergence between any two *feature-topic* distributions, such as *Jenson-Shannon* divergence, *cosine similarity* and *KL* divergence. Taking cosine similarity as an example, the Equation is shown as below:

$$similarity(f_i, f_j) = \frac{\boldsymbol{v}_i \cdot \boldsymbol{v}_j}{||\boldsymbol{v}_i||\ ||\boldsymbol{v}_j||} = \frac{\sum_{r=1}^{k} p_{ir} \times p_{jr}}{\sqrt{\sum_{r=1}^{k} p_{ir}^2} \times \sqrt{\sum_{r=1}^{k} p_{jr}^2}} \tag{2}$$

Where k is the topic number and p is the probability value of $\boldsymbol{v}$.

Line 8-14 pick out a set of feature elements from the new cluster, defined as *medoid*, which can be used to represent the theme of c. Two metrics, *similarity value* and *topic degree*, are used to determine the medoid. Firstly, to get the $value_s$, we calculate the similarity between $c_i \in c$ and the *centroid* of c through Equation 2, where the vector $\boldsymbol{v}_{\bar{c}}$ of centroid is calculated by $\boldsymbol{v}_{\bar{c}} = \frac{\sum_{i=1}^{|c|} \boldsymbol{v}_i}{|c|}$. Then, the Equation 3 is used to calculate the $degree_t$ based on the following two important observations of feature-topic distribution V_m in our datasets.

$$degree_t = x_{max} + \frac{1}{e^{\sqrt{\frac{\sum_{r=1}^{\hat{k}} (x_{max} - p_{ir})^2}{\hat{k}}}}} \tag{3}$$

Where x_{max} is the maximum value of $\boldsymbol{v}_i$, and $\hat{k}$ is the frequency when $\boldsymbol{v}_i$ not equal to zero, and p_{ir} is any value that not equals to zero in $\boldsymbol{v}_i$.

Observation 1 The most probable topic reflects the most prominent theme that the document (feature element) is about.

Observation 2 The more widely and evenly distributed its topics are, the higher-level the document (feature element) is.

In brief, Equation 3 can ensure the feature element in the coarsest granularity have the highest score $degree_t \in (0, 2]$, where $x_{max} \in (0, 1]$ can reflect the emphasis topic and the formula $\frac{1}{e^{\sqrt{\sum_{r=1}^{\hat{k}} (x_{max} - w_{ir})^2}}} \in (0, 1]$ can reflect the semantic generality.

The $score_m$ is used to measure the *medoid* calculated as the Equation of line 12, where κ and λ is the empirical coefficients.

Finally, the *medoid* with the highest $score_m$ would be selected. Measuring the similarity for the each pair of elements in $medoid_C$ (line 15), the $feature_C$ (line 17) can be formed by merging distinguished feature elements whose similarity score below a threshold (set to 0.38). Each iteration in the repeat clause saves the M and $feature_C$ to HESA. On the termination of the algorithm, a feature-ontology (Figure 4) for the category is constructed.

E. The Retrieval Method of HESA

Figure 4 depicts an example of the construction process and result with 6 data nodes using the iAHC algorithm. Each cluster consists of several nodes and the top node (the red color one in Figure 4) is the feature of the cluster. The concept *layer* is defined as below:

(1) The layer 0 consists of the bottom nodes which are the original feature elements;

(2) The layer i consists of the feature node of cluster i and all nodes in layer $i - 1$ except those being merged in cluster i.

For example, the layer 3 consists of the features of cluster 3, cluster 2 and cluster 1, because all the nodes in different clusters.

The most important advantage of the feature-ontology is that the nodes in a layer are the most representative features under a given similarity threshold. If the stakeholder needs a generalized feature of the category, the feature in the top layer can be selected. Assuming that the category of Figure 4 can be covered by three features, the feature nodes of cluster 3, cluster 2 and cluster 1 would be retrieved step by step. From top down, the semantic granularity is finer and finer accompanying with the increasing number of features, which can satisfy the requirements of multi-grained reuse environments.

Algorithm 2 is a flexible method to retrieve features in terms of quantity, which demonstrates the advantages of the feature-ontology. The input is the quantity of feature you need for a specific domain. Line 1-6 show the process of finding the suitable layer. Then, all the nodes in the same layer are selected out in the repeat clause (line 9-20). An online search engine has been implemented based on it in this paper.

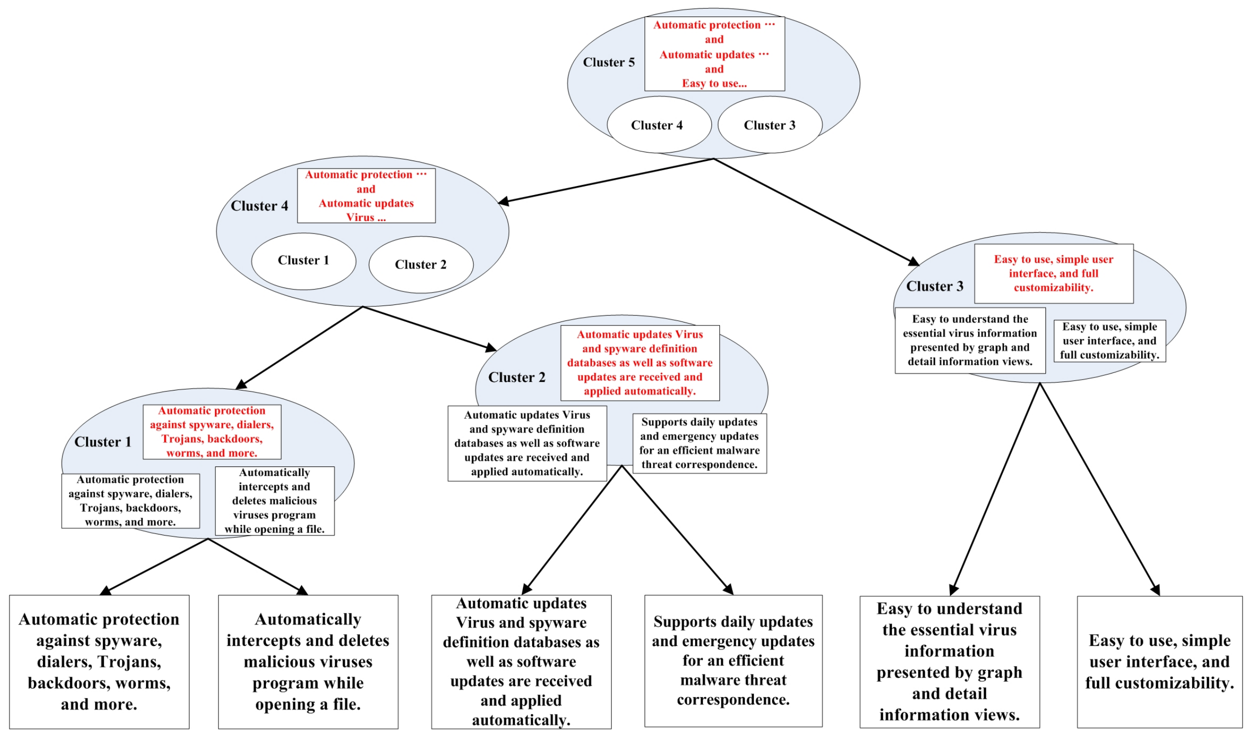

Cluster 5
Automatic protection ··· and Automatic updates ··· and Easy to use...
Cluster 4
Cluster 3
Cluster 4
Automatic protection ··· and Automatic updates Virus ...
Cluster 1
Cluster 2
Cluster 3
Easy to use, simple user interface, and full customizability.
Easy to understand the essential virus information presented by graph and detail information views.
Easy to use, simple user interface, and full customizability.
Cluster 2
Automatic updates Virus and spyware definition databases as well as software updates are received and applied automatically.
Automatic updates Virus and spyware definition databases as well as software updates are received and applied automatically.
Supports daily updates and emergency updates for an efficient malware threat correspondence.
Cluster 1
Automatic protection against spyware, dialers, Trojans, backdoors, worms, and more.
Automatic protection against spyware, dialers, Trojans, backdoors, worms, and more.
Automatically intercepts and deletes malicious viruses program while opening a file.
Automatic protection against spyware, dialers, Trojans, backdoors, worms, and more.
Automatically intercepts and deletes malicious viruses program while opening a file.
Automatic updates Virus and spyware definition databases as well as software updates are received and applied automatically.
Supports daily updates and emergency updates for an efficient malware threat correspondence.
Easy to understand the essential virus information presented by graph and detail information views.
Easy to use, simple user interface, and full customizability.

Figure 4. An example of the feature-ontology

IV. Empirical Evaluation

In this section, we present our dataset and experiment setting, research questions and answers, and describe some threats to validity.

A. Dataset and Experimental Setting

Dataset: We have collected 187,711, 432,004 and 45,021 projects profiles from Softpedia.com, Sourceforge.com and Freecode.com respectively. Compared with that of the other two communities, the quantity of projects from Freecode.com is relatively small. Thus, we just adopt projects in Softpedia.com and Sourceforge.com. The feature elements have been classified into 385 categories and we randomly choose the data of 3 unique categories to evaluate our method including Antivirus, Audio-Player and File-manger. Furthermore, the feature elements are preprocessed by removing commonly occurring words and then by stemming the remaining words to their root form. To ensure the quality of data, we omit the preprocessed feature elements with less than 6 words. Table 1 presents the details about our dataset.

Parameter setting: As shown in Table 1, for LDA, the number of topics K was empirically set as different value, and the hyper-parameters α and β were set with $\alpha = 50/K$ and $\beta = 0.01$ respectively, and the iteration of Gibbs Sampling was set as 1000. In addition, the coefficients κ and λ of **Algorithm 1** were set with $\kappa = 0.7$ and $\lambda = 0.3$.

B. Research Questions

To demonstrate the effectiveness of the approach in this paper, we are interested in the following research questions:

Table I
PREPROCESSED EXPERIMENT DATASETS

Category	#Softpedia	#Sourceforge	#Total	#Topic
Antivirus	2919	1105	4024	50
Audio-Player	3714	1283	4997	60
File-Manager	2270	970	3240	40

RQ1 How the resultant feature-ontology looks like?

RQ2 Does the iAHC algorithm achieve better clustering results than the simple but classical method and the state-of-the-art approach?

RQ3 How accurate is the feature-ontology? Is the structure reasonable?

C. Cross-Validation Design of the User Study

The cross-validation limits potential threats to validity such as fatigue, bias towards tasks, and bias due to unrelated factor. We randomly divided the 45 students from computer school of NUDT into three groups to evaluate RQ2 and RQ3. The 2 questions and the 3 categories of dataset can be composed to 6 tasks. Each group randomly picks up 2 of them and finishes in one day, and then we summarize the result.

***RQ1: Feature-ontology**:* Figure 4 shows an example feature-ontology of the Antivirus category which is a very reasonable structure. The features (red color) in different layer can be mapped to resources on different granularities. In addition, the feature is relatively representative for each cluster.

***RQ2: Clustering Results**:* We choose the K-Medoids (tf-idf), a classic and widely used clustering algorithm, and

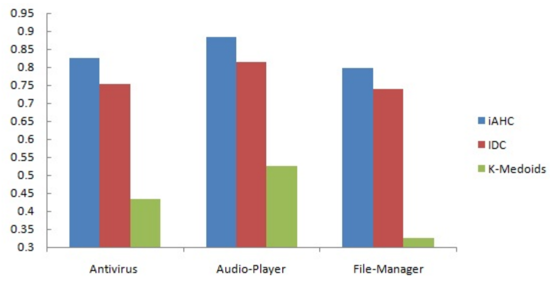
Algorithm 2 a flexible granularity retrieval method

```
Require:
    $k_f$ the quantity of feature you need;
    $T$ a hierarchical structure consisting of $n$ nodes;
Ensure:
    $featureSet$ a set of features;
 1: $layer \leftarrow n - k$
 2: if $layer = 0$ then
 3:     return the nodes of $T[0]$
 4: end if
 5: $i \leftarrow 0$
 6: $featureSet[i] \leftarrow$ the node of $T[layer]$
 7: $layer \leftarrow layer - 1$
 8: $i \leftarrow i + 1$
 9: repeat
10:     if $T[layer]$ is a subtype of $featureSet[i]$ then
11:         $layer \leftarrow layer - 1$
12:     else
13:         $featureSet[i] \leftarrow$ the node of $T[layer]$
14:         $i \leftarrow i + 1$
15:         $layer \leftarrow layer - 1$
16:         if $layer = 0$ then
17:             return the rest nodes of $T$
18:         end if
19:     end if
20: until $i = k_f$
21: return $featureSet$
```

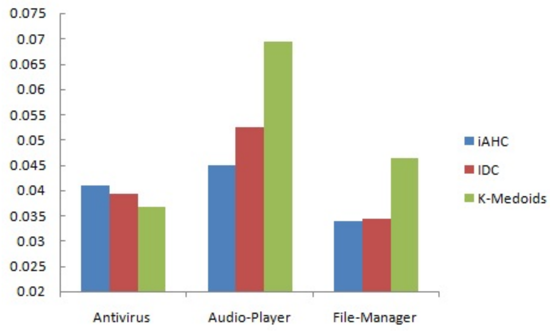
the Incremental Diffusive Clustering (IDC), the state-of-the-art technique proposed in paper [8], as the baseline. Especially, the IDC use the feature descriptions from Softpedia.com which is the same as our dataset. We also use the modified version of Cans metric [8] to compute the ideal number of clusters. Then, we retrieve the corresponding number of clusters from HESA for comparison. Precision is a percent of the reasonable feature elements in a cluster. Figure 5 shows the average value and standard deviation of the judgments given by different groups. We can see that our approach achieves the highest precision in all three categories and relatively low deviations. The precisions and deviations are comparatively stable across different categories, which shows the probability that our approach is more generalizable in different domains. We plan to conduct more experiments to study this issue in the future.

RQ3: Accuracy of the feature-ontology: According to the three categories, participants randomly choose 15 clusters in different layers from HESA using the online search engine respectively. Each participant is randomly assigned 3 layers and asked to provide a 3-point Likert score for each cluster to indicate whether they agree if the feature is the most representative of all terms. Score 3 means very reasonable, Score 2 means reasonable but also have better one, Score 1 means unreasonable.

Table 2 shows that 35.03% features are reasonable, 49.57% partially reasonable and only 15.40% unreasonable. The mean of Likert score is 2.20, which means that the feature selected



(a) the average value



(b) the standard deviation

Figure 5. The clustering results for each category

Table II
EVALUATION OF FEATURE-ONTOLOGY QUALITY

Category	Score-3	Score-2	Score-1	Likert
Antivirus	33.3%	50.0%	16.7%	2.17
Audio-Player	39.1%	46.3%	14.6%	2.25
File-Manager	32.7%	52.4%	14.9%	2.18
Average	35.03%	49.57%	15.4%	2.20

out by our approach is reasonably meaningful.

D. Threats to validity

First, the participants manually judge the clustering results and their ratings could be influenced by fatigue, prior knowledge and the other external factors. These threats were minimized by randomly distributing participants to the various groups and dividing the tasks into multiple parts. Second, due to our limited datasets, parameters used in our approach, the evaluation is not comprehensive enough.

In the feature, with the help of our online search engine, we plan to adopt the idea of crowdsourcing and upload a lot of tasks about use study on the Internet with a low price, such as 5 cent. Therefore, a comprehensive and reliable evaluation result can be obtained.

V. Related Work

Recently, mining software repository has been brought into focus and many outstanding studies have emerged that use this data to support various aspects of software development [16]. However, fewer previous works have been done for mining software feature and especially construction of feature-ontology (defined in this paper), to the best of our knowledge.

In this section, we review some previous works about feature analysis and ontology learning.

In feature analysis area, most approaches involve either the manual or automated extraction of feature related descriptions from software engineering requirements and then use the clustering algorithm to identify associations and common domain entities [7] [17] [18]. Niu et al. [19] propose an on-demand clustering frame-work that provided semi-automatic support for analyzing functional requirements in a product line. Mathieu Acher et al. [9] introduced a semi-automated method for easing the transition from product descriptions expressed in a tabular format to feature models. A decision support platform is proposed in paper [20] to build the feature model by employing natural language processing techniques, external ontology (such as WordNet), and MediaWiki system. However, the quantity of the existing documents is so limited that the brilliance of data mining techniques cannot be fully exploited. To address this limitation, paper [8] and [10] proposed the Incremental Diffusive Clustering to discover features from a large number of software profiles in Softpedia.com. Based on the features, a recommendations system is build by using association rule mining and the k-Nearest-Neighbor machine learning strategy. Compared with these studies, the clustering algorithm presented in this paper is more effective by mining the semantic structures from feature elements and especially focus on the construction of feature-ontology.

Ontology learning (also called ontology extraction) from text aims at extracting ontological concepts and relation from plain text or Web pages. Paper [21] developed an ontology learning framework using hierarchical cluster and association rule for ontology extraction, merging, and management. Several researches have attempted to induce an ontology-like taxonomy from tags. Jie Tang et al. [22] proposed a generative probabilistic model to mine the semantic structure between tags and their annotated documents, and then create an ontology based on it. Xiang Li et al. [23] enhance an agglomerative hierarchical clustering framework by integrating it with a topic model to capture thematic correlations among tags. Based on tens of thousands of software projects and their tags, Shaowei Wang et al. [24] propose a similarity metric to infer semantically related terms, and build a taxonomy that could further describe the relationships among these terms. In this paper, to support multi-grained reuse, emphases of the feature-ontologys construction is on the measure of similarity and granularity instead of generality.

VI. Conclusion and Future Work

The continuing growth of open source ecosystems creates ongoing opportunities for mining reusable knowledge. In this paper, we have explored the idea of mining large scale repositories and constructed the HESA to support software reuse. In the future, we plan to improve the performance of our method and aggregate richer features from software repositories. In addition, we will design several representative applications based on HESA, such as software resource recommendation system, to support the reuse of multi-grained resources.

VII. Acknowledgement

This research is supported by the National High Technology Research and Development Program of China (Grant No. 2012AA011201) and the Postgraduate Innovation Fund of University of Defense Technology (Grant No.S120602).

References

[1] W. B. Frakes and K. Kang, "Software reuse research: Status and future," *IEEE Trans. Softw. Eng.*, vol. 31, no. 7, pp. 529–536, Jul. 2005. [Online]. Available: http://dx.doi.org/10.1109/TSE.2005.85

[2] T. Wang, G. Yin, X. Li, and H. Wang, "Labeled topic detection of open source software from mining mass textual project profiles." in *Software Mining*, 2012, pp. 17–24.

[3] A. E. Hassan and T. Xie, "Mining software engineering data." in *ICSE (2)*, 2010, pp. 503–504.

[4] S. Apel and C. Kastner, "An overview of feature-oriented software development." 2009, pp. 49–84.

[5] K. Lee, K. C. Kang, and J. Lee, "Concepts and guidelines of feature modeling for product line software engineering." in *ICSR*, 2002, pp. 62–77.

[6] K.C.Kang, S.G.Cohen, J.A.Hess, W.E.Novak, and A.S.Peterson, "Feature-oriented domain analysis (foda) feasibility study. technical report." 1990.

[7] W. B. Frakes, R. P. Dłaz, and C. J. Fox, "Dare: Domain analysis and reuse environment." 1998, pp. 125–141.

[8] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, and M. Mirakhorli, "On-demand feature recommendations derived from mining public product descriptions." in *ICSE*, 2011, pp. 181–190.

[9] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire, "On extracting feature models from product descriptions." in *VaMoS*, 2012, pp. 45–54.

[10] C. McMillan, N. Hariri, D. Poshyvanyk, J. Cleland-Huang, and B. Mobasher, "Recommending source code for use in rapid software prototypes." in *ICSE*, 2012, pp. 848–858.

[11] D. Blei, A. Ng, and M. Jordan, "Latent Dirichlet Allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003. [Online]. Available: http://www.cs.princeton.edu/ blei/lda-c/

[12] K. Tian, M. Revelle, and D. Poshyvanyk, "Using latent dirichlet allocation for automatic categorization of software." in *MSR*, 2009, pp. 163–166.

[13] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling." in *ICSE (1)*, 2010, pp. 95–104.

[14] T. Griffiths, "Gibbs sampling in the generative model of Latent Dirichlet Allocation," Stanford University, Tech. Rep., 2002. [Online]. Available: www-psych.stanford.edu/ gruffydd/cogsci02/lda.ps

[15] A. K. McCallum, "Mallet: A machine learning for language toolkit," 2002, http://mallet.cs.umass.edu.

[16] A. E. Hassan, "The road ahead for mining software repositories." 2008.

[17] S. Park, M. Kim, and V. Sugumaran, "A scenario, goal and feature-oriented domain analysis approach for developing software product lines." 2004, pp. 296–308.

[18] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler, "An exploratory study of information retrieval techniques in domain analysis." in *SPLC*, 2008, pp. 67–76.

[19] N. Niu and S. M. Easterbrook, "On-demand cluster analysis for product line functional requirements." in *SPLC*, 2008, pp. 87–96.

[20] E. Bagheri, F. Ensan, and D. Gasevic, "Decision support for the software product line domain engineering lifecycle." 2012, pp. 335–377.

[21] A. Maedche and S. Staab, "Learning ontologies for the semantic web." in *SemWeb*, 2001.

[22] J. Tang, H. fung Leung, Q. Luo, D. Chen, and J. Gong, "Towards ontology learning from folksonomies." in *IJCAI*, 2009, pp. 2089–2094.

[23] X. Li, H. Wang, G. Yin, T. Wang, C. Yang, Y. Yu, and D. Tang, "Inducing taxonomy from tags: An agglomerative hierarchical clustering framework," in *Advanced Data Mining and Applications*. Springer Berlin Heidelberg, 2012, vol. 7713, pp. 64–77.

[24] S. Wang, D. Lo, and L. Jiang, "Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging." in *ICSM*, 2012, pp. 604–607.