

Navigating Source Code with Words

Dawn Lawrie, Dave Binkley
Loyola University Maryland
Baltimore, MD, USA
{lawrie,binkley}@cs.loyola.edu

Abstract—The hierarchical method of organizing information has proven beneficial in learning in part because it maps well onto the human brain’s memory. Exploiting this organizational strategy may help engineers cope with large software systems. In fact such an strategy is already present in source code and is manifested in the class hierarchies of objected-oriented programs. However, an engineer faced with fixing a bug or any similar need to locate the implementation of a particular feature in the code is less interested in the syntactic organization of the code and more interested in its conceptual organization. Therefore, a conceptual hierarchy would bring clear benefit. Fortunately, such a view can be extracted automatically the source code.

The hierarchy generating tool HierIT performs this task using an information-theoretic approach to identify “content-bearing” words and associate them hierarchically. The resulting hierarchy enables an engineer to better understand the concepts contained in a software system. To study their value, an experiment was conducted to quantitatively and qualitatively investigate the value that hierarchies bring. The quantitative evaluation first considers the *Expected Mutual Information Measure* (EMIM) between the set of topic words and natural language extracted from the source code. It then considers the *Best Case Tree Walk* (BCTW), which captures how “expensive” it is to find interesting documents. Finally, the hierarchies are considered qualitatively by investigating their perceived usefulness in a case study involving three engineers.

I. INTRODUCTION

The hierarchical method of organizing information has proven beneficial in learning in part because it maps well onto the human brain’s memory [2]. This organizational system can be observed in many human endeavors including the Dewey Decimal System for organizing books [3], that used by Amazon.com, and the typical file system on a computer that organizes files using directories with subdirectories. Hierarchical organization can also be found in software. Object-oriented software being an obvious example. Employing an organizational strategy that maps well onto human thinking helps software engineers cope with the complexities of large systems.

Within software engineering, existing hierarchical organizations are dominated by the structural aspects of software. For example, the behavior of an instance of class `BufferedInputStream` can be understood in part by knowing that `BufferedInputStream` is a subclass of `FilterInputStream`, which, in turn is a subclass of `InputStream`. Unfortunately, such a syntax focused view is not always the most useful. For example,

engineers, faced with a bug report or having some similar need to locate the implementation of a particular feature in the code, are less interested in the syntactic organization of the code and more interested in its *conceptual organization*. Because such an organization is almost never made explicit in the code nor is it maintained through links to the design documentation, obtaining a conceptual hierarchical view of a program requires mining it from the source code.

Modern search engines have moved away from generating hierarchies because they can get mostly right answers to typical web queries with sub-second response time. However, for harder problems than a simple web search, generating a hierarchy is worth a few seconds computational expense. Alas, manually extracting such information can be expensive and error prone, making tool support an obvious preference. Borrowing from natural language techniques, it is possible to automatically generate a hierarchy. This paper introduces HierIT, a tool for source code hierarchy generation. To generate a hierarchy, HierIT relies on two key concepts: *topicality* and *predictiveness*. The *topicality* of a word indicates how *content-bearing* the word is in the source code. Such words would describe the domain of the program. For example in banking software, topical words include deposit and withdraw. *Predictiveness* measures the number of subtopics a given word is likely to have. Account is likely predictive in this banking software, where many topical words occur in the context of the word account.

The key question addressed in this paper is “can HierIT produce effective hierarchies of software?” In other words, are the automatically constructed hierarchies HierIT produces from source code meaningful and of value? The value HierIT brings is empirically investigated both quantitatively and qualitatively. The quantitative evaluation has two aspects. First, using the *Expected Mutual Information Measure* (EMIM) [4] between the set of topic words and natural language extracted from the source code, and second, using the *Best Case Tree Walk* (BCTW) [1] to estimate the cost of using the hierarchy to support accessing a user-selected set of interesting documents. EMIM compares the distribution of words found in the hierarchy to the distribution of all words, while BCTW summarizes the cost of navigating the hierarchy to find interesting documents. Finally, after the two quantitative assessments, HierIT’s hierarchies are considered qualitatively by investigating their perceived usefulness.

The remainder of the paper first presents the technique for automatically creating a hierarchy in Section II. This is followed by an example hierarchy generated by HierIT.

Next, in Section IV the quantitative evaluation techniques are described in detail. Section V presents the results of the quantitative and qualitative evaluations. Related work is considered in Section VI, which is followed by future work. The paper concludes in Section VIII.

II. BUILDING SOURCE CODE HIERARCHIES

The key assumption behind hierarchy construction is that for a given set of documents (e.g., a set of web pages or source code files), there is an optimal list of words that describes the topics contained in the documents. Furthermore, this list is ordered with the first word capturing the most dominant topic (the one that maximizes *topicality* and *predictiveness*) and the last word capturing the least dominant topic. The challenge in hierarchy construction is to automatically identify and order an appropriate list of words.

The underlying hypothesis for automatically constructing hierarchies is that words chosen to be included in the hierarchy are chosen based on two important qualities. The first is *topicality*, which indicates whether a word is content-bearing in a document set. It is believed that such words are more likely to be interpreted as a topic of the document set because by definition they communicate the meaning of the text. The second quality is *predictiveness*, which measures the expected number of subtopics of a given word. It is hypothesized that the more subtopics a word has, the more likely the word is to describe a high-level topic of the document set.

Combining these two concepts leads to a model for finding the best topic words that is based on the following *Term Selection Formula*:

$$T^* = \arg \max_{T \subseteq W} \sum_{i=1}^{|T|} \mathcal{P}(\text{Topical}(t_i), \text{Predictive}(t_i) | t_1 \dots t_{i-1}) \quad (1)$$

where W is the set of all possible ordered subsets of the vocabulary and T^* is the resulting ordered set of optimal topic words. In more detail, Equation 1 identifies the set of topic words that maximize the joint probability of *topicality* and *predictiveness*, where the probability for a given topic word, t_i , is dependent on topic words that appear before it in the ordering.

In more detail, *Topical* is a binary random variable and is true when a word is content bearing. A topical word is not necessarily the most frequent word in the document set. In fact it may only be mentioned occasionally; however, the information conveyed to the reader by a content-bearing word is crucial to understanding the document. For example, endangered mammal appearing in a document about humpback whales would associate the species with an endangered species topic, making endangered *topical*.

The second part of the joint probability in Equation 1 is *predictive*, which can be thought of as a precondition for the occurrence of other words. For example, sports is good predictor of team, league, etc. This quality takes into account the fact that subtopics occur in the presence of the main topic. Previous work on topic hierarchies [5], [1] has shown that this is an important aspect of topic words, because predictive words are frequently used to discuss a topic at

different levels of generality. Specifically, predictive words are those which occur with a distinct set of vocabulary and without which it would be highly unlikely that other words would occur. For example, in a retrieved set about *endangered mammals*, “endangered” is likely to be a predictive word because an endangered animal is likely to be identified as endangered when it is described. In such a document set, the word “species” and the specific animal mentioned is unlikely to occur without first finding the word “endangered.” Therefore, general words related to a topic are often identified as predictive words.

Combining these two random variables, produces a set of words that will maximize a user’s understanding of the information contained in the documents. In order to compute the joint probability of the two random variables, the two probabilities are assumed to be independent; thus knowing that a word is or is not *topical* does not lead to knowing whether or not it is *predictive*. This independence assumption is a standard assumption in Information Retrieval with regard to words.

Using the independence assumption, joint probability can be computed as follows:

$$\mathcal{P}(\text{Topical}(t_i), \text{Predictive}(t_i) | t_1 \dots t_{i-1}) = \mathcal{P}(\text{Topical}(t_i) | t_1 \dots t_{i-1}) \mathcal{P}(\text{Predictive}(t_i) | t_1 \dots t_{i-1}).$$

Thus, methods for estimating *topicality* and *predictiveness* can be developed independently. In the remainder of this section, Section II-A describes the estimation techniques for *topicality*, while *predictiveness* is considered in Section II-B.

A. Estimating Topicality

When creating hierarchies, it is necessary to estimate the probability that a particular word is in fact a topic word. Cronen-Townsend and Croft have shown in their work with the *Clarity Measure* that words with high contributions to the Kullback-Leibler divergence score are likely to be about the topic of the document, while words that are not ranked highly are less likely to be about the topic [6]. Thus, KL divergence can be used to estimate *topicality*.

In this context, KL divergence is a measure of relative entropy between the language model of the document set used to create the hierarchy and the language model of some more general text, for example, a collection of source code. Each word’s contribution to KL divergence is calculated using the formula

$$\text{KL contribution}(w) = \mathcal{P}_{\text{hier}}(w) \log_2 \frac{\mathcal{P}_{\text{hier}}(w)}{\mathcal{P}_{\text{GT}}(w)}, \quad (2)$$

where $\mathcal{P}_{\text{hier}}(w)$ is the probability of a word in the document set and $\mathcal{P}_{\text{GT}}(w)$ is the probability of a word in general text. A particular word w has a contribution of zero when $\mathcal{P}_{\text{hier}}(w) = \mathcal{P}_{\text{GT}}(w)$. It has a positive contribution when $\mathcal{P}_{\text{hier}}(w) > \mathcal{P}_{\text{GT}}(w)$, and a negative contribution when $\mathcal{P}_{\text{hier}}(w) < \mathcal{P}_{\text{GT}}(w)$. Most topical words are those where $\mathcal{P}_{\text{hier}}(w) \gg \mathcal{P}_{\text{GT}}(w)$ because such words occur much more frequently in the document set than would be expected given the general model.

Calculating a KL contribution requires estimating $\mathcal{P}_{\text{hier}}(w)$ and $\mathcal{P}_{\text{GT}}(w)$. The most straightforward approach

is to use the maximum likelihood estimate in the unigram language model for the hierarchy where

$$\mathcal{P}_{hier}(w) = \frac{\text{number of occurrences of } w}{\text{total number of words}}. \quad (3)$$

The probability of a word in the general text can be estimated in a similar way by using the frequency of words in a suitably large collection.

The choice of the general text collection impacts the quality of the hierarchies and is one of the parameters of hierarchy creation that deserves future investigation. For the purposes of this initial study, the software project itself is used as the source of the general text probabilities. Future work will consider the impact of other possibilities such as a collection of projects. Motivation for such study comes from building natural language hierarchies where this choice impacts the quality of the hierarchies. For instance, if the general text is a collection of news documents and the hierarchy is over web documents, common web phrases like “Click on me” will appear to be topical because they have a low probability of occurring in the news domain, when in fact they are not topical.

Given the variation in the vocabularies used in different software projects, the “click on me” problem is a real hazard if the general text is drawn from a non-representative collection of projects. One obvious choice is to draw both the document set and the general text from the same program, as is done herein.

An alternative approach for calculating the KL contribution biases the probability of a word to a given query. This can be useful when the document set is a retrieved set and the user is interested in topics related to the query. This alternative approach focuses the hierarchy on topics related to the query. Given a query Q , the probability of w is estimated using the formulation in Cronen-Townsend and Croft [6]:

$$\mathcal{P}_{hier_Q}(w) = \mathcal{P}(w|Q) = \sum_{D \in hier_Q} \mathcal{P}(w|D)\mathcal{P}(D|Q), \quad (4)$$

where $hier_Q$ is the set of documents retrieved for query Q and

$$\mathcal{P}(D|Q) = \prod_{q \in Q} \mathcal{P}(q|D).$$

Here, q is a word in query Q and D is a document from the set of documents $hier_Q$. This method demotes terms such as the phrase “Click on me” because they are irrelevant to the query. This new formulation is an alternative to Equation 3, which is used in the computation of Equation 2.

Regardless of whether query biasing is used to calculate the KL contribution or not, the probability that a word w_i is *topical* in the *Term Selection Formula* is given by the following equation:

$$\mathcal{P}(\text{Topical}(t_i)|t_1 \dots t_{i-1}) = \frac{\text{KL contribution}(t_i) - \text{minKL}}{\text{maxKL} - \text{minKL}} \quad (5)$$

where minKL is the smallest KL contribution value observed and maxKL is the largest KL divergence value observed. This method of estimation ignores $t_1 \dots t_{i-1}$ of Equation 1, which means that the quality of *topicality* is independent of the prior topic words selected.

B. Using Relative Entropy to Identify Predictive Words

As with *topicality*, a relative entropy approach is considered to estimate *predictiveness*. Since the quality of *predictiveness* exists when a word has many subtopics, relative entropy is used to identify predictive words by considering each word and its subtopics in the document set. The language models needed for *predictiveness* are first developed and then used to build an approximation of relative entropy.

The first step when using relative entropy to identify predictive words involves transforming the document set into a collection of *topic language models*. Given the definition of predictiveness, a subtopic is defined as a *key point that is discussed within the context of the topic*. In order to identify the context of a topic, it is hypothesized that the words occurring near each other in the text have a strong dependence on one another. Considering word a , which occurs at position i , and word b , which occurs at position j , the likelihood of dependence between them decreases as the distance $|i - j|$ increases. This hypothesis is loosely based on Wittgenstein’s *Use Theory of Meaning*, which states that the meaning of a word is defined by the circumstances of its use [7]. In light of this theory, some of the surrounding words can be expected to be generalities of the topic, while others are subtopics of the topic. If a almost always occurs in the presence of b , but b occurs without a then b can be taken as a subtopic of a . For example, in a document on endangered whales and rhinos, “endangered” is a super topic while “whales” and “rhinos” are both sub topics.

Thus a word w is a potential subtopic of topic word t if the language model assigns $\mathcal{P}(t|w)$ a value close to one. In general, this probability describes how dependent an occurrence of w is on an occurrence of topic word t . Although this probability looks similar to the probabilities in a bigram language model, it is not the same because w does not have to occur in the position preceding t .

The topic language models are constructed based on $\text{segments}(w, k)$ – the collection of segments of text of length $2k + 1$ with w at their center. For topic t , those elements of $\text{segments}(w, k)$ that are of interest are those that include t . The intent here is that k captures the maximum distance between words where associations are probable. Given the reliance on the proximity of w and t , these models are referred to as *co-occurrence* language models. Formally, a co-occurrence model captures the following probability

$$\mathcal{P}(t|w) \equiv \mathcal{P}(x_j = t | \exists x_j \text{ s.t. } i - k \leq j \leq i + k \text{ and } x_i = w)$$

The resulting language models differ from more traditional language models in two ways: (1) it is not generative and (2) it is not a probability mass function, so it cannot be used in the KL divergence calculation.

When estimating a co-occurrence model, each individual probability, $\mathcal{P}(t|w)$, is calculated by dividing number of segments $\text{segments}(w, k)$ that contain t by the total number of segments that contain w :

$$\mathcal{P}(t|w) = \frac{|\{t \in S \text{ s.t. } S \in \text{segments}(w, k)\}|}{|\text{segments}(w, k)|}.$$

This means that $\mathcal{P}(t|w) = 1$ when t co-occurs with w in every segment of w (i.e., the subtopic w always co-occurs with the topic word t).

Having defined the language models needed to determine *predictiveness*, it is now possible to approximate the relative entropy used in its definition. A co-occurrence model is used to determine which words are the best topics. In the co-occurrence model, the assumption is that the likelihood that a word is a subtopic is proportionate to the number of times the word occurs near the topic. To find how topic words compare to one another, the probabilities are summed over the entire vocabulary, W :

$$\mathcal{P}(\text{Predictive}(t)) = \frac{1}{|W|} \sum_{w \in W} P(t|w). \quad (6)$$

Identifying topics is accomplished by interpreting the co-occurrence model as a bipartite graph where each word in the vocabulary is represented by two vertices. One vertex represents the word as a topic, and the other vertex represents the word as a subtopic. Edges are created between topic vertices and subtopic vertices when $\mathcal{P}(t_i|w_j) > 0$ and assigned the weight $\mathcal{P}(t_i|w_j)$. The resulting graph enables the application of the Dominating Set Problem (DSP) algorithm, to effectively find the main topic word of the document set. Given that this problem is NP-hard, a greedy approximation is used.

Simply identifying and removing successive main topic words would allow words that are really subtopics of a particularly dominant topic to crowd out true topic words. To avoid this problem the *coverage* of a topic word (in terms of its likely subtopics) is removed along with each topic word when identifying subsequent topics. It is hypothesized that when the entire vocabulary has been covered, then all the main topics have been identified.

The coverage of topic word t , $\text{coverage}(t)$, is an over approximation of t 's potential subtopics. Removing the vertices for covered words prevents subtopics of popular topics from appearing as topics themselves. The set $\text{coverage}(t)$ is defined as the set of words w where $\mathcal{P}(t|w) > \text{threshold}$. The threshold is generally set as the mean of probabilities $\mathcal{P}(t|w)$.

The greedy approximation is based on Equation 6. For each topic vertex, the incident edge weights are summed to compute $\mathcal{P}(\text{Predictive}(t))$. The vertex maximizing Equation 2 is selected as the first topic. Then, the selected vertex and those vertices representing words it covers are removed from the graph.

Before choosing the second topic, the topic vertex weights must be recalculated. To sum over the edge weights of the remaining subtopics Equation 6 is modified as follows:

$$\mathcal{P}(\text{Predictive}(t_i)|t_1 \dots t_{i-1}) = \frac{1}{|W_i|} \sum_{w \in W_i} P(t_i|w), \quad (7)$$

where $W_1 = W$, $W_{i+1} = W_i - S_i$, and $S_i = \{t_i\} \cup \text{coverage}(t_i)$. Each successive topic word is then chosen greedily until all vertices have been removed (i.e., W_i becomes empty). At this point the summarization of the complete document set has been attained by ensuring that each

individual subtopic vertex is covered by at least one topic vertex.

After the top-level topics are selected based on their *topicality* and *predictiveness*, subtopics can be identified. To do so the same process is used but with language models produced using the text surrounding each topic word. Specifically, for topic word t , the set of segments $\text{segments}(t, k)$ are treated like documents as input to the algorithm. For each level the value of k is typically reduced to focus the relationship between t and w . The process can be repeated for as many levels as desired.

III. A FIRST LOOK

A source code hierarchy presents the topics and subtopics of a program in a way that is easily browsable by a user. The intent is that an engineer can rapidly get a sense of the vocabulary used in the selected code. It is important to note that the engineer is not expected to spend significant effort or time on understanding uninteresting or challenging-to-interpret word combinations. In essence the question being addressed is “does there exist a path to interesting documents?”

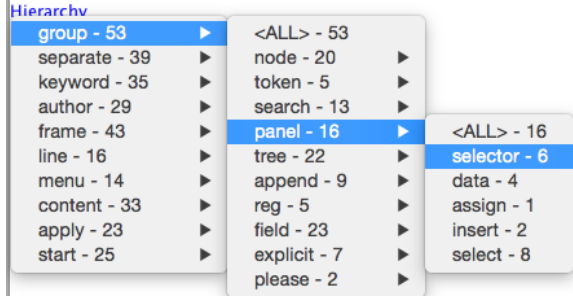
In the current tool, the hierarchy is displayed using standard popup menus as shown in Figures 1 and 2. Each menu item is labeled with a topic word and the number of documents that are associated with a particular topic. While considered in greater depth at the end of the paper, the query shown in Figure 8 boils down to a problem when *using the context selector from the groups panel to add keywords*. It is interesting how quickly words from the query jump out on the screen shot shown in Figure 1, in which the topic *group > panel > selector*, is currently selected.

The hierarchal presentation can focus the engineer on a manageable number of interesting documents. By construction the topic words in the path from the root to a particular topic used in the hierarchy all appear in the document, making the clusters monothetic. If an item is selected, then links to the documents summarized by the hierarchy can be displayed so that an engineer can navigate to documents of interest, such as those shown in the lower half of the two figures.

The first step to generating a hierarchy is to identify the set of documents that should be used as the basis for the hierarchy. Figures 1 and 2 display a hierarchy that was based on the top two hundred methods retrieved for a query related to JabRef2.6's Bug ID 1436014, which is shown in Figure 8. Considering both Figures 1 and 2, there are six highlighted topic words, five of which appear in the query. The word *dialog* is the exception. Given the bug description, this might be a reasonable exploration path because dialogs may have to do with user interactions.

To illustrate the potential benefits that hierarchy brings, consider the methods related to the topic *group > panel > selector*, which are labeled (1) to (6) in the bottom half of Figure 1. These methods are ordered in the same relative ordering that the search engine produced. However, rather than appearing as (1) through (6) as is shown in the figure, the search engine positioned the methods at ranks 25, 40, 47, 100, 179, and 180 respectively. Such lowly ranked methods are

unlikely to be considered by an engineer. A second advantage comes from enabling an engineer can gain useful information by studying the documents clustered together in a topic by the hierarchy. For example, examining the documents in Figure 2 reveals that the topic *content* > *selector* > *dialog* is present in two parts of the source code: the *GroupDialog* and the *ContentSelector* related classes.



- (1) [net.sf.jabref.JabRefFrame.fillMenu\(\)](#)
- (2) [net.sf.jabref.groups.GroupSelector.AddSubgroupAction.actionPerformed\(ActionEvent\)](#)
- (3) [net.sf.jabref.groups.GroupSelector.AddGroupAction.actionPerformed\(ActionEvent\)](#)
- (4) [net.sf.jabref.groups.GroupsTree.drop\(DropTargetDropEvent\)](#)
- (5) [net.sf.jabref.collab.GroupChange.makeChange\(BasePanel, NamedCompound\)](#)
- (6) [net.sf.jabref.groups.GroupSelector.RemoveGroupAndSubgroupsAction.actionPerformed\(ActionEvent\)](#)

Figure 1. Displays the hierarchy when selecting the *group* > *panel* > *selector* topic and the functions associated with that topic.

IV. QUANTITATIVE EVALUATION TECHNIQUES

This section describes two views used to evaluate the “goodness” of a hierarchy. The first focuses on the words found in the hierarchy. In essence it asks how good a job these words do at summarizing the vocabulary of the document collection. The second view looks at the best-case scenario for a hierarchy user who has to reach a given set of documents. This view asks how helpful the structural organization found in the hierarchy is in reaching a given set of documents.

A. Word Evaluation

The hierarchy is intended to be a summary of a document collection; thus, it is important to understand how well the words contained in a hierarchy summarize the text of the document collection from which it was built. This first evaluation criteria considers this question by calculating how well the words of the hierarchy predict the vocabulary of the document collection using the Expected Mutual Information Measure (EMIM) [4]. This measure provides a means of comparing the quality of the words that make up the hierarchies. EMIM does this by measuring the extent to which the distributions of the topic words and the non-stopwords of the vocabulary deviate from stochastic independence. It is defined as follows:

$$\text{EMIM}(T, V) = \sum_{t \in T, v \in V} P(t, v) \log \frac{P(t, v)}{P(t)P(v)},$$



- (1) [net.sf.jabref.groups.GroupDialog.GroupDialog\(JabRefFrame, BasePanel, AbstractGroup\)](#)
- (2) [net.sf.jabref.groups.AutoGroupDialog.AutoGroupDialog\(JabRefFrame, BasePanel, GroupSelector, GroupTreeNode, String, String, String\)](#)
- (3) [net.sf.jabref.ContentSelectorDialog.ContentSelectorDialog\(JabRefFrame, boolean, MetaData, String\)](#)
- (4) [net.sf.jabref.ContentSelectorDialog2.ContentSelectorDialog2\(Dialog, JabRefFrame, BasePanel, boolean, MetaData, String\)](#)
- (5) [net.sf.jabref.ContentSelectorDialog2.ContentSelectorDialog2\(Frame, JabRefFrame, BasePanel, boolean, MetaData, String\)](#)
- (6) [net.sf.jabref.ContentSelectorDialog.ContentSelectorDialog\(JabRefFrame, boolean, MetaData\)](#)
- (7) [net.sf.jabref.FieldContentSelector.FieldContentSelector\(JabRefFrame, BasePanel, Dialog, FieldEditor, MetaData, AbstractAction, boolean\)](#)
- (8) [net.sf.jabref.ContentSelectorDialog_fieldSelector_actionAdapter.ContentSelectorDialog_fieldSelector_actionAdapter\(ContentSelectorDialog\)](#)
- (9) [net.sf.jabref.ContentSelectorDialog_Close_actionAdapter.ContentSelectorDialog_Close_actionAdapter\(ContentSelectorDialog\)](#)

Figure 2. Displays the hierarchy when selecting the *content* > *selector* > *dialog* topic and the functions associated with that topic.

where T is the set of topic terms and V is the set of document non-stopwords. In order to calculate the joint probability, the Lavrenko and Croft[8] formulation is used:

$$P(t, v) = \sum_{d \in D} P(d)P(t|d)P(v|d),$$

where D is the set of documents and $P(d)$ is a uniform distribution, and $P(t|d)$ and $P(v|d)$ are estimated with the maximum likelihood estimation as is done for Equations 3. The greater the dependence between the two distributions, the better the hierarchy is a summary of the text.

B. Structural Evaluation

The second evaluation considers the complexity of navigating to a set of relevant documents of interest to a user. It assumes that the user’s goal is to read all of the “interesting” documents, while avoiding reading “uninteresting”

documents. The evaluation determines the upper bound on the efficiency gain that can be realized by using a hierarchy. Unlike a list of documents, which provides a single ordering of the documents, there may be several different paths within a hierarchy that lead a user to the relevant documents.

The evaluation is optimistic. It finds the most efficient way, using the hierarchy, that a user could read all relevant documents. This is done using *Best Case Tree Walk* (BCTW) [1], which was inspired by Kruskal's Minimum Spanning Tree algorithm. In that algorithm the cheapest edges are added to the tree first. Likewise, in the BCTW algorithm the cheapest hierarchy nodes are added first. A hierarchy node appears as a menu item in Figure 1. It consists of a topic word, a number of documents, a parent node, and a set of child nodes. When calculating the cost of utilizing the hierarchy, it is assumed that once a topic word's hierarchy node is selected, all documents in that node are read by the user. According to this evaluation, the best hierarchies are those with a very high concentration of relevant documents, so that a minimal number of non-relevant documents (documents not chosen by the user as interesting) are encountered.

BCTW measures how well the hierarchy supports efficient access to the user's chosen set of relevant documents. It estimates the time it takes to find all relevant documents by calculating the total number of hierarchy menus that must be traversed and the number of documents that must be read from the set of selected hierarchy nodes S . The algorithm aims to find an optimum path through the hierarchy by traveling to nodes that hold the greatest concentration of relevant documents. Given the evaluation knows where the relevant documents are located, the algorithm can quickly hone in on the parts of the hierarchy containing relevant documents. The algorithm selects the set of nodes that leads to the minimum location cost.

One of the challenges presented by the hierarchy is that the same document can be encountered in multiple locations in the hierarchy, so some combinations of nodes that access all relevant documents will be of higher cost than others. In theory, all combinations of nodes would need to be compared since documents can appear in multiple nodes and nodes can require different overlapping paths to be explored. In practice both of these costs have little impact on the cost of adding a new node to the selected set, so the number of documents at a node is a good approximation for the cost of adding the node. As an approximation, the algorithm presented will nearly always lead to the set with the minimal cost.

Figure 3 shows the algorithm for selecting the nodes that need to be traversed. The algorithm accumulates the selected hierarchy nodes in S . It first breaks the hierarchy into individual hierarchy nodes, which correspond to particular topics. For example, *group > search* represents a hierarchy node in Figure 1. Next, all the hierarchy nodes that contain only irrelevant documents are removed. The remaining nodes, set F , are then ordered by the total number of documents referenced in each node. For instance the hierarchy node for *group > search* in Figure 1 contains thirteen documents. Nodes are then considered in order from fewest documents to most documents.

```

BCTW(Hierarchy h, Document Set D)
  S ← ∅ // selected nodes
  N ← getHierarchyNodes(h)
  F ← NodesWithRelevantDocs(h, D)
  OF ← sortByNumDocsContainedInNode(F)
  foreach n ∈ OF
    if (containsUnreadRelevantDoc(n, S, D))
      R ← redundantNodes(n, S, D)
      S ← n ∪ S - R
    else if (containsMultipleRelevantDoc(n, D))
      R ← redundantNodes(n, S, D)
      T ← n ∪ S - R
      if (Cost(T) < Cost(S))
        S ← T
  return Cost(S)

```

Figure 3. The *Best Case Tree Walk* Algorithm finds the nodes in the hierarchy that minimizes that cost of access all user identified interesting documents. set of user defined "interesting" documents D

The main body of the algorithm first determines if the node n under consideration for selection adds any new relevant documents to the selected set S . If the node adds new relevant documents, it is selected. If this node also provides access to a document that can be accessed by another node, then the previously selected node may be redundant. Redundant nodes are those whose relevant documents are a subset of the relevant documents in n and the other nodes. Thus the new set of selected nodes, S , includes n and the previously selected nodes minus any redundant nodes in R .

A node n may also be added if by adding the node, the cost of accessing relevant documents will be lowered. This will only occur when a node contains multiple relevant documents because multiple nodes will become redundant, making the amortized cost of adding the larger node acceptable. This, in essence, increases the proportion of relevant to irrelevant documents. In this circumstance, a temporary set T is created that consists of n and the previously selected nodes minus any redundant nodes in R . When the cost of the new set T is less than cost of the selected set, S , S is replaced by T .

Although this algorithm leads to a succinct analysis of the hierarchy, it is worth noting that it contains three simplifying assumptions. First, all documents are regarded as equal size and thus document-length variability is ignored. Similarly, all menus are treated as equally complex to process despite the variability in their length. Finally, when computing the cost, documents and menus are treated the same (i.e., the time and effort required to read a document is regarded as being the same as that to read a menu). This most likely overestimates the effort required to read a menu. In most instances, the number of menus examined across different types of hierarchies is very similar. Differences are observed in the number of documents a user must read, so this assumption has a very minor effect of the results.

A hierarchy is assigned a score using the **Cost** function over the selected hierarchy nodes. A lower score denotes a superior hierarchy. Different hierarchies can be compared on the basis of their score.

V. RESULTS

This section empirically investigates source-code hierarchies by addressing the following two research questions.

- RQ1 What is the quantitative impact of *query-focusing* and *program* on the words and structure of the hierarchies?
- RQ2 How well do HierIT hierarchies match an engineer’s intuition for interesting topic words?

A. RQ1

This section compares several different hierarchies both in terms of the words they include (using EMIM) and their structure (using BCTW). The hierarchies were constructed from the top 200 documents retrieved for 39 JabRef2.6 modification requests and 45 Eclipse3.0 modification requests using a retrieval configuration that has performed well in other studies [9]. These two programs were selected to represent a small program (JabRef with 117K lines of code and 4,741 methods) and a large program (Eclipse with 1.84M lines of code and 96,309 methods). For each modification request, four different hierarchies were generated. One variant of the hierarchies was unbiased, using Equation 3 in the computation of *topicality*. These documents were retrieved using a query based on both the summary and description in the modification request. The other three variants all involved query bias as defined in Section II-A. For these variants, the retrieval and the bias were done using the same query. The variants come from using the summary as the query, the description as the query, and then as was done in the unbiased version, both the summary and the description.

Apart from the document set and bias, all hierarchies used the same other parameter settings. In particular all hierarchies are three levels (as displayed in Figures 1, where three menus are shown representing each of the three levels). The settings for k for determining *predictiveness* are $k=50$ for the top-level topic words, $k=25$ for mid-level topic words such as *panel* in Figure 1, and $k=10$ for determining low-level topic words such as *insert* in Figure 1.

With 84 queries between the two programs and four variants, there would be 336 different hierarchies. However, some of the queries failed to produce results because no documents were actually scored by the search engine. In this case, many search engines return the documents in the order they were indexed; however, in this experiment such queries were excluded, as these would reflect documents from the classes that were at the beginning of the alphabet rather than related to the query. This resulted in the absence of 44 hierarchies, so the results are reported over 292 hierarchies.

First, considering the words, a statistical model was constructed with EMIM as the response variable and Program and query bias as the explanatory variables. Using an ANOVA analysis with Tukey’s Honest Significant difference to find statistical differences and Bonferroni to account for multiple comparisons, both program and query bias were found to have a significant impact on EMIM. Figure 4 plots the results where higher scores denote better summaries. Where the data fails the ANOVA’s normality assumption, the results were confirmed using the non-parametric Kruskal-Wallis test.

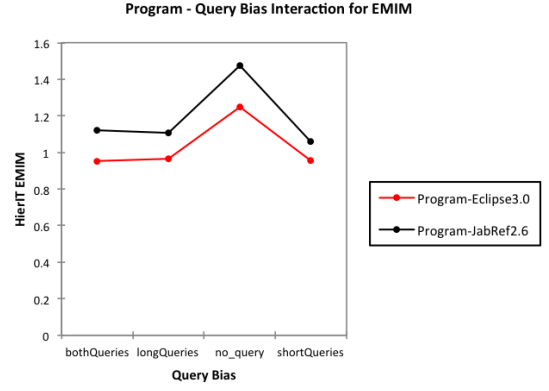


Figure 4. The interaction between program and query bias based on EMIM

In terms of query bias, *no_bias* creates a better summary than the three biased hierarchies. The p -value for comparing *no_bias* with each of the other three variants is < 0.0001 , highly significant. This is a very encouraging result. Introducing bias, should impact the ability of the hierarchy to fairly summarize the document set since it is favoring terminology from the query, rather than drawing more uniformly from the distribution of document set words. Thus, this provides evidence that the hierarchy is a fair summary of the document set.

Secondly, there is no real difference between any of the types of query biasing. Since the main difference between the query types is the number of words in the query, this means that query length has little to no impact on the bias of the hierarchy. A few words are sufficient to change the terminology of hierarchy.

As also shown in Figure 4, for the second explanatory variable, program, query bias has no significant impact. The JabRef hierarchies are statistically better summaries of the underlying document set than Eclipse hierarchies. There are many possible explanations for this result. First looking at the total number of nodes in the hierarchies shows that on average Eclipse hierarchies are bigger: there are an average of 269.6 nodes in an Eclipse hierarchy compared to JabRef’s average of 241.4. This may be because the diversity in retrieved methods is greater for Eclipse, as it is a larger program. However, the picture is complex. It is likely that if the Eclipse documents were about more diverse topics, then the highest level would show that Eclipse hierarchies were more likely to hit the maximum size; however, the data does not support this observation. Figure 5 shows that Eclipse is more likely to have a small number of high-level topics, while JabRef has a large number of high-level topics. With fewer high-level topics, but more nodes, it appears that Eclipse high-level topics have more subtopics than JabRef, so the artificial limit of ten menu items is impacting the ability to fully describe the subtopics.

Turning to the structure analysis, the question is how well the algorithm minimizes the cost to access a set of interesting or relevant documents. Although hierarchies can be useful to many software engineering tasks, this initial investigation uses as a convenience set, the modified methods as identified by



Figure 5. Histogram of the number of hierarchies with a particular size menu by Program

the SEMERU data set [10] as the “interesting” documents in the hierarchy.

Given the standard alternative to a hierarchy when doing retrieval is to use a ranked list, this investigation first compares the cost of accessing all relevant documents using the hierarchy to the cost of accessing all the relevant documents using the ranked list. To mirror the construction of the hierarchies, the ranked lists are also truncated to the top 200 documents. Therefore, neither approach will find relevant documents ranked worse than 200.

In the comparison the hierarchies provide lower cost access to the relevant documents than a ranked list. HierIT has an average access cost of 13.8, while the ranked list’s average cost is 79.8. Both a t-test and the Mann-Whitney report the difference in the mean is highly statistically significant $p < 0.0001$.

Turning to an analysis of the hierarchies themselves, BCTW takes the role of the response variable and the explanatory variables are program and query bias. Overall no statistical differences were found. In a model that only considers query bias, weak statistical significance was found ($p = 0.072$). This model indicates that the non-biased hierarchies also have lower cost, while a bias using the summaries produces higher costs. As can be seen in Figure 6, the No Bias hierarchies have the lowest score. In addition the Summary queries for Eclipse appear to be particularly poor at biasing the hierarchy in a way that provides easy access to relevant documents. Given the weak significance, more data is necessary to determine if these trends truly exist.

Considering the two evaluations together, there is evidence that hierarchies without bias lead to the best source code hierarchies both in terms of their ability to summarize and to provide access to “interesting” documents, which is significantly better than using a ranked list.

B. RQ2

The second research question, *How well do HierIT hierarchies match an engineer’s intuition for interesting topic words?* takes a more qualitative look at the hierarchies. The experiment used five example queries from JabRef and asked three software engineers to label the non-stoplist words as interesting or non-interesting. The five example queries were

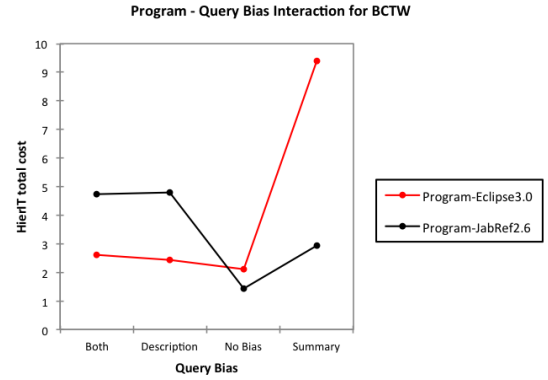


Figure 6. The interaction between program and query bias based on BCTW

the long descriptions taken from five bug reports in JabRef’s issue tracking system. (The id numbers of the selected bug reports are included in Figure 9.) Two of the five descriptions are shown verbatim (in *italics*) at the top of Figures 7 and 8.

For each description, two sets of data were generated. First, each description was shown, along with a list of the non-stoplist words extracted from the description. The engineers were asked to cross out those words that they felt would not be helpful in searching for relevant source code. Second, each description was also used as a query. From the returned ranked list, the top 200 relevant documents were used in each case to construct a hierarchy. The engineer’s selections can be compared with the words found in the hierarchy to assess the quality of the hierarchy. To simplify the comparison only the top level of the hierarchy is considered.

In addition to the two descriptions, Figures 7 and 8 show the non-stoplist words from each description. Those words shown in bold occur in the top-level topic list of the corresponding hierarchy. Next to each word are the three engineers’ assessment where a ‘Y’ indicates the engineer thought the word had value, and an ‘N’ means that they crossed it out during the selection.

The data for all five selected descriptions is presented in Figure 9 where the first five rows summarize the five example queries. Each row shows the Bug Id, the number of non-stoplist words extracted from the description and then the four entries of the 2x2 contingency table for the query. For reference, the last two rows summarizes the examples shown Figures 7 and 8.

The penultimate row of the table shows the totals over all five queries. First, of the query terms not chosen by HierIT to be part of the top-level hierarchy, a little less than half the time the engineers were actually interested in them. For the other half, the engineers agreed that the words were uninteresting. In contrast over twice as many of those words HierIT used in the top level of the hierarchy were selected by an engineer. This can be seen in the last row, which shows the subject selected yes/no percentages separately for words used by HierIT and words not used by HierIT. The contingency table for the totals is shown graphically in Figure 10.

The long description for JabRef Bug ID 1285977 reads

It is not possible to properly sort a numeric field (such as years), when all “numbers” do not have the same number of digit (actually, this is an alphabetical ordering, and JabRef ignores numeric fields).

The words below are the non-stop words accompanied by subject assessment

Word	Subject		
	1	2	3
sort	Y	Y	Y
field	N	Y	N
actually	N	N	N
number	Y	Y	Y
ordering	Y	Y	Y
numeric	Y	Y	Y
alphabetical	N	Y	Y
years	Y	Y	N
digit	N	Y	Y
JabRef	N	N	Y
ignores	N	N	Y
possible	N	N	N
properly	N	N	N

Figure 7. JabRef Long Description 1285977. Bold words are in the hierarchy. A ‘Y’ labels an engineer-selected word.

The long description for JabRef Bug ID 1436014 reads

When adding a keyword via the content selector, no comma is added before the keyword. I have set “When adding/removing keywords separate them by:” to “,” in the Preferences > Groups panel. But keywords still get separated by a space only. This applies to JabRef 2.0.1

The words below are the non-stop words accompanied by subject assessment

Word	Subject		
	1	2	3
Groups	Y	Y	N
separate	Y	N	Y
keyword	Y	Y	Y
content	N	Y	Y
applies	N	N	N
Preferences	Y	Y	N
selector	Y	Y	Y
comma	Y	N	Y
panel	Y	Y	N
JabRef	N	N	Y
set	Y	N	N
space	Y	N	N
removing	N	N	N

Figure 8. JabRef Long Description 1436014. Bold words are in the hierarchy. A ‘Y’ labels an engineer-selected word.

Issue Id	Words	Selected by HierIT			
		No		Yes	
		Subject Selected No	Subject Selected Yes	Subject Selected No	Subject Selected Yes
1540646	29	51	21	2	13
1538769	19	14	31	5	7
1503956	8	5	7	2	10
1285977	13	13	11	5	10
1436014	13	12	12	6	9
Totals:	82	95	82	20	49
Percent:		54%	46%	29%	71%

Figure 9. Contingency analysis for each query studied.

Statistically, a Chi-squared test applied to the 2x2 contingency table determines if the user selection is deferentially distributed over the HierIT selection. The resulting p -value of 0.0005 indicates a statistically significant difference; thus HierIT chosen words are more valued by the engineers.

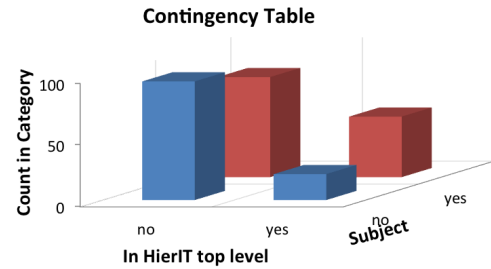


Figure 10. Graph of the contingency table for the totals.

VI. RELATED WORK

There are two bodies of work that are related to this in the areas of natural language processing for automatically building hierarchies and also work in software engineering. Within natural language processing community, researchers have worked on the task of automatically constructing hierarchies for the last two decades [5], [11], [12], [13], [14].

Within the software engineering domain, there has been some interest in finding topics, mainly with the topic modeling technique called Latent Dirichlet Allocation (LDA) [15]. One of the disadvantages of LDA is that it is difficult to describe the topic to a user because it is a mixture of many words. The monothetic topics presented herein are much easier for a user to grasp, given that a single word or phrase can be used to describe the topic.

Other projects also attempt to identify keywords within software. For example, the work of Ohba and Gondow, develops a technique to identify keywords which could be used to summarize source code [16]. In this work a modified version of the standard term weighting scheme term frequency-inverse document frequency called ckTF/IDF is used to identify concept keywords. These keywords came from identifiers. Although the authors do not use co-occurrence information in there definition as is done in HierIT, they do observe the importance of co-occurrence information to their Identifier Exploratory Framework, which allows a user to browse the identifiers found in the source code.

Finally, Abebe et al.’s recent approach to supporting concept location develops source code ontologies [17]. There is a

similarity between the ontologies and hierarchies making the study of ontologies a possible place to look for improvements to HierIT's hierarchies. Another related idea is found in the work of De Lucia et al. which generates labels for source code entities [18]. There are commonalities between the words of a level in a hierarchy and those in a label. Thus the process for extracting labels might be leveraged to better extract topics from a source code collection.

VII. FUTURE WORK

Two main directions for future work are considered. One is to create better source code hierarchies and the other is to investigate their usability. There are several parameters that need to be set when building the hierarchy. Exploring these parameter settings is one avenue of future work. When examining the parameters, two in particular may need special attention given the change in the genre of the documents from natural language text to source code. The first is the source of the general text used to calculate the *topicality* of a word. Currently, the project itself is used as the general text. The way in which the general text is selected has been found to have a big impact on the resulting hierarchies. Therefore, it is worth exploring alternatives to simply using the project, which may be too small. One alternative is to use mixture models that combine the program and some larger collection of programs.

Another parameter that is important to *predictiveness* is the window size, which is used when determining words that co-occur. In natural language word counts have been found to work as well as more natural features of the writing such as paragraphs and sentences. In addition these features can be non-trivial to recognize since, for example, a period is not always a sentence ending period. In source code, scope and statements are trivial for a compiler to recognize, and they may be more meaningful when identifying words that are dependent on each other. Thus, future work will move away from fixed-sized windows in favor of source code specific boundaries. While considering the window size, one might also investigate whether some windows are more important, such as those the beginning of a file or those proceeding method definitions, and integrate such information into the algorithm.

Finally, HierIT is a tool intended to be used by engineers. Therefore, it is important to understand the hierarchies strengths and limitations. To do this, a larger user study will be designed to ascertain HierIT's usability.

VIII. CONCLUSION

This paper provides an encouraging first look at the potential that a hierarchical topic analysis brings to source code. HierIT proved itself capable of automatically creating hierarchies from source code that both effectively summarized the language found in the source code and also organized the information to speed access to documents of interest. Most encouraging of the empirical analysis is how well the hierarchy's top-level words matched those selected by the engineers who took part in the case study. Looked at collectively, this evidence portends a bright future for hierarchical analysis of source code.

ACKNOWLEDGMENT

The authors would like to thank Razieh Behjati, Thomas Rolfsnes, and Stefano Di Alesio for their help in conducting this experiment.

REFERENCES

- [1] D. Lawrie, W. Croft, and A. Rosenberg, "Finding topic words for hierarchical summarization," in *Proceedings on the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, 2001, pp. 349–357.
- [2] F. Reif, *Applying Cognitive Science to Education: Thinking and Learning in Scientific and Other Complex Domains*. The MIT Press, 2008.
- [3] OCLC Forest Press, "Introduction to the dewey decimal classification," http://www.oclc.org/dewey/about/about_the_ddc.htm, 2002.
- [4] C. van Rijsbergen, *Information retrieval*, 2nd ed. London: Butterworths, 1979.
- [5] M. Sanderson and W. B. Croft, "Deriving concept hierarchies from text," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and Development in Information Retrieval*, 1999, pp. 206–213.
- [6] S. Cronen-Townsend and W. Croft, "Quantifying query ambiguity," in *Proceedings of HLT*, 2002, pp. 94–98.
- [7] C. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.
- [8] V. Lavrenko and W. Croft, "Relevance-based language models," in *Proceedings on the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, 2001, pp. 365–379.
- [9] D. Binkley, D. Lawrie, C. Uehlinger, and D. Heinz, "Enabling improved ir-based feature location," *Journal of Systems and Software*, vol. 101, 2015.
- [10] B. Dit, M. Revelle, M. Gethers, and D. Poshvanyk, "Feature location in source code: A taxonomy and survey," *Journal of Software Maintenance and Evolution*, vol. 23, no. 7, 2011.
- [11] P. Anick and S. Tipirneni, "The paraphrase search assistant: Terminological feedback for iterative information seeking," in *Proceedings on the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, M. Hearst, F. Gey, and R. Tong, Eds., 1999, pp. 153–159.
- [12] H. Yang, "Browsing hierarchy construction by minimum evolution," *ACM Trans. Inf. Syst.*, vol. 33, no. 3, pp. 13:1–13:33, Mar. 2015. [Online]. Available: <http://doi.acm.org/10.1145/2714574>
- [13] M. Hall, S. Fernando, P. Clough, A. Soroa, E. Agirre, and M. Stevenson, "Evaluating hierarchical organisation structures for exploring digital libraries," *Information Retrieval*, vol. 17, no. 4, pp. 351–379, 2014. [Online]. Available: <http://dx.doi.org/10.1007/s10791-014-9242-y>
- [14] J. C. S. Soderland and G. B. Mausam, "Hierarchical summarization: Scaling up multi-document summarization," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014, pp. 902–912.
- [15] S. W. Thomas, "Mining software repositories using topic models," in *Software Engineering (ICSE), 2011 33rd International Conference on*. IEEE, 2011, pp. 1138–1139.
- [16] M. Ohba and K. Gondow, "Toward mining "concept keywords" from identifiers in large software projects," in *Proceedings of the Proceedings of the Second International Workshop on Mining Software Repositories*, St Louis, MO, May 2005.
- [17] S. L. Abebe, A. Alicante, A. Corazza, and P. Tonella, "Supporting concept location through identifier parsing and ontology extraction," *Journal of Systems and Software*, vol. 86, no. 11, pp. 2919–2938, 2013.
- [18] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Labeling source code with information retrieval methods: an empirical study," *Empirical Software Engineering*, vol. 19, no. 5, pp. 1383–1420, 2014.