

Recovering Concepts from Source Code with Automated Concept Identification

Maurice M. Carey
Dept. of Computer Science & Engineering
Arizona State University - Tempe
Box 878809, Tempe, AZ 85287-8809
mmcarey@asu.edu

Gerald C. Gannod*†
Dept. of Computer Science and Systems Analysis
Miami University
Oxford, OH 45056
gannodg@muohio.edu

Abstract

The complexity of the systems that software engineers build has continuously grown since the inception of the field. What has not changed is the engineers' mental capacity to operate on about seven distinct pieces of information at a time. Improvements like the widespread use of UML have led to more abstract software design activities, however the same cannot be said for reverse engineering activities. The well known concept assignment problem is still being solved at the line-by-line level of analyzing source code. The introduction of abstraction to the problem will allow the engineer to move farther away from the details of the system, increasing his ability to see the role that domain level concepts play in the system. In this paper we present a technique that facilitates filtering of classes from existing systems at the source level based on their relationship to the core concepts in the domain. This approach can simplify the process of reverse engineering and design recovery, as well as other activities that require a mapping to domain level concepts.

1. Introduction

The complexity of the systems that software engineers build has continuously grown since the inception of the field. What has not changed is the engineers' ability to deal with a limited number of datum at any given time, psychologists tell us that the average person has working memory capacity to operate on about seven distinct pieces of information at a time [1]. While improvements in the field like the widespread use of UML has led to more abstract software design activities, the same cannot be said for reverse engineering activities. The well known *concept assignment problem* is still being solved at the *line-by-line* level of analyzing source code. The introduction of abstraction to the

problem would allow the engineer to move farther away from the details allowing his own limited resources to capture a broader picture of the system, increasing his ability to see the role that domain level concepts play in the system.

In this paper we present a technique that facilitates filtering of classes from existing systems at the source level based on their relationship to the core concepts in the domain. This allows a software engineer to work with a smaller subset of the system. The technique presented involves collecting object-oriented metric data from existing systems, which are then used in machine learning methods to create classifiers of the systems. The classifiers are then used to identify classes that are likely related to domain level concepts. As will be seen, we have gathered some interesting results that indicate this technique may ease the effort required to identify concepts in existing systems. This approach can simplify the process of reverse engineering and design recovery, as well as other activities that require a mapping to domain level concepts.

The remainder of this paper is organized as follows. Section 2 describes the context for our approach including information on the concept assignment problem, object-oriented metrics, and machine learning. Section 3 presents the process used to collect data from an existing system, as well as training a support vector machine. Section 4 shows an example of the filtering application of the approach to the Panda software. Section 5 evaluates the effectiveness of our approach by presenting the results obtained from analyzing example software systems. Section 6 discusses some areas of related work. Finally, Section 7 concludes and suggests future investigations.

2. Background

This paper makes use of ideas related to several areas of research. The concept assignment problem is similar to our goal of filtering the class diagram. Metrics are used as a way of assigning quantifiable attributes to a class. Finally, machine learning is used to automate the process of classification.

*This author supported by National Science Foundation CAREER grant No. CCR-0133956.

†Contact Author.

2.1 Concept Assignment Problem

Biggerstaff et al. [2] describes the concept assignment problem as recognizing concepts in software systems and building a model or human level understanding of the concepts. We see the concept assignment problem as a two part process. The first step is to identify concepts in the software system. The second step is to build a model of the concepts. This paper describes a new method of *identifying* concepts that are represented by classes, and using the identification to produce an abstraction of a recovered class diagram. Specifically, we describe an instance of the concept assignment problem dealing only with object-oriented classes. Object-oriented design suggests that we ignore the details of the implementation of a class, so we believe that analyzing object-oriented software at the class level is a valid approach to solving the concept assignment problem.

2.2 Object-Oriented Metrics

A metric is defined as a standard of measurement [3]. In this paper, we use metrics as quantifiable attributes of classes. We will be more interested in what a set of metrics might say about the class than what each individual metric implies. Since our focus is on attributes of classes we will be most interested in class level metrics.

Several object-oriented metrics are used in our approach. These metrics are primarily designed to capture information about the size and complexity of software at the class level. Figure 1 provides a brief summary of the metrics used in our work. These metrics were chosen because they were available to be analyzed. We have not optimized the metrics used, nor do we believe that is an appropriate step to take at this time because we do not want to over optimize the metrics used to the data set collected. The best way to view the role of the metrics collected here is as incomplete indicators of the concepts, each metric adds more information to the decision making process.

2.3 Machine Learning

This work makes use of two different machine learning algorithms developed by the machine learning community. The first, Support Vector Machines (SVM) are the primary algorithm we use to classify results. The second, k-nearest neighbors (KNN) is used to validate results obtained with our primary algorithm. We use both as a black-box that takes inputs that are n length vectors of real numbers and outputs classification decisions.

2.3.1 Support Vector Machines

Support Vector Machines (SVM) were first introduced by Boser et al. [4]. SVM are an example of a learning methodology known as supervised learning [5]. A learning methodology is an approach to creating programs that calculate answers by analyzing given examples while supervised learning uses examples consisting of input-output

Number of Attributes (NOF) The total number of instance and static attributes in the class.

Number of Static Attributes (NSF) The number of class or static attributes in the class.

Number of Methods (NOM) The total number of instance and static methods in a class.

Number of Overridden Methods (NORM) The total number of methods in the class that override methods from an ancestor class.

Number of Static Methods (NSM) The number of static or class methods in the class. Henderson-Sellers refers to this as Number of Class Methods (NCM).

Number of Parameters (PAR) The number of parameters in a method. In this paper we use the average over the class.

Number of Subclasses (NSC) The total number of direct subclasses of this class, a.k.a Number of Children.

Method Lines of Code (MLOC) The number of lines of code contained in the body of all methods in the class.

Lack of Cohesion of Methods (LCOM) LCOM can be calculated as in (1) where $\mu(A_j)$ is the number of methods that access the attribute A_j , a is the number of attributes, and m is the number of methods. LCOM is a measure of the cohesiveness of a class where smaller values indicate more cohesive classes.

Nested Block Depth (NBD) The depth of nested blocks of code.

McCabe Cyclomatic Complexity (VG) The maximum number of independent circuits through the directed acyclic graph of a method. In this paper we use the average over the class.

Weighted Methods per Class (WMC) The sum of McCabe Cyclomatic Complexity for the n methods in the class i , calculated by the formula given in (2).

Depth of Inheritance Tree (DIT) The depth of the class in the inheritance hierarchy.

Specialization Index (SIX) Defined as $\frac{NORM-DIT}{NOM}$. Designed so that higher values indicate classes that are more specialized.

$$LCOM = \frac{\left(\frac{1}{a} \sum_{j=1}^a \mu(A_j)\right) - m}{1 - m} \quad (1)$$

$$WMC = s_i = \sum_{j=1}^n V_{ij}(G) \quad (2)$$

Figure 1. Definition of Metrics Used

pairs. The goal in a learning problem is to find a function that maps the given inputs to the desired outputs.

SVM generate linear functions as their hypothesis. The hypothesis are then applied to attributes that have been transformed to a high dimensional feature space. The transformation of attributes to a feature space is carried out by the application of kernels to the example datum. An example of a kernel is the radial basis function kernel shown in Equation (3) [6], which intuitively describes a radius of

influence, where $i = 1, \dots, \ell, j = 1, \dots, \ell$, and σ is a parameter to the kernel that scales the radius of the support vectors influence.

$$K(\vec{x}_i, \vec{x}_j) = \exp\left(-\frac{\|\vec{x}_i - \vec{x}_j\|^2}{2\sigma^2}\right). \quad (3)$$

SVM used as binary classifiers must solve the optimization problem in Equation (4) [5]. Geometrically this translates into finding a linear separating hyperplane in the higher dimensional space. The hyperplane is optimized for maximal margin and defined by the *weight vector* \vec{w} and *bias* b . In actual practice the dual representation is maximized allowing optimization to take place in the kernel space with slack terms introduced along with a penalty parameter C . In this context, $\vec{x}_i \in \mathbb{R}^n$ are the n -dimensional training vectors, $y_i \in \{1, -1\}$ are the classification labels for the training vectors, and ℓ is the number of vectors. The set of classification labels $\{1, -1\}$ correspond to classes that are concepts (1), and those that are implementation details (-1).

$$\begin{aligned} \min_{\vec{w}, b} \quad & \langle \vec{w} \cdot \vec{w} \rangle \\ \text{subject to} \quad & y_i (\langle \vec{w} \cdot \vec{x}_i \rangle + b) \geq 1, \\ \text{where} \quad & i = 1, \dots, \ell \end{aligned} \quad (4)$$

In our data set, $n = 14$ so the vector \vec{x}_i has 14 dimensions, each related to a metric. Examples of some sample vectors from the Panda data set are shown in Table 1. The metrics and their ordering are as in Table 1.

Table 1. Panda data set examples.

i	\vec{x}_i	y_i
1	$\langle 0, 0, 3, 31, 11, 0, 0, 1.75, 4, 0.444, 2.75, 0.5, 0, 1 \rangle$	-1
2	$\langle 4, 0, 2, 331, 92, 0, 2, 1.806, 32, 0.5, 2.556, 0.917, 0, 5 \rangle$	1
3	$\langle 0, 0, 8, 51, 16, 0, 0, 1.071, 14, 0.862, 1.143, 0.929, 0, 1 \rangle$	1
4	$\langle 1, 0, 2, 24, 10, 0, 2, 1.667, 2, 0.5, 3.333, 0, 0, 1 \rangle$	-1

2.3.2 k-Nearest Neighbors.

k-Nearest Neighbors (KNN) [7] is the simplest instance-based machine learning method. The algorithm is based on all the training instances being points in an n -dimensional space \mathbb{R}^n . Instances are classified by calculating the Euclidian distance to all points in the training set. The Euclidian distance, shown in Equation 5, is calculated from $a_r(\vec{x})$ (e.g., the r th attribute of the learning instance vector \vec{x}).

$$d(\vec{x}_i, \vec{x}_j) = \sqrt{\sum_{r=1}^n (a_r(\vec{x}_i) - a_r(\vec{x}_j))^2} \quad (5)$$

The class of the k nearest points is then used as a simple majority rules vote, the class of the majority is assigned as the class of the point in question. An alternative is to weigh the class of the k nearest points based on the multiplicative inverse of their distance from the point in question.

3. Approach

This section describes our overall approach and presents our analysis of the implementation. Then we discuss the processes used to classify the systems, collect metrics, and generate the results.

3.1 Overview

The primary objective of our research is to study Hypothesis 1.

Hypothesis 1 *The vectors consisting of metrics measuring size and complexity of software components are strong indicators of classes that represent concepts within a domain.*

In other words, in a given domain the interesting concepts will be realized in software by classes or components whose size and complexity, measured by the metrics presented in Section 2.2, will be *recognizably* different from uninteresting classes that do not represent core concepts of the domain.

Figure 2 shows an overview of the process used in our approach. The *Data Collection and Classification* step is used to gather information about training data as well as data on the subject system. The result (*Metric and Classification Data* in Figure 2) is then analyzed in the *Data Analysis* step. The result of the *Data Analysis* step is a set of identified concepts that can be fed into any of a number of applications that use the concept identification to perform recovery of high level abstractions.

One of the intentions of our work is to support reverse engineering and design recovery by facilitating recovery of abstract models from as-built models. This approach builds a classifier that operates on a individual software systems, though future work may expand the scope to operate on multiple systems.

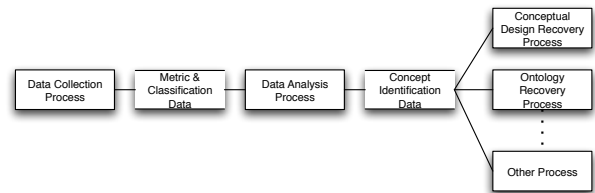


Figure 2. Approach Overview

3.2 Data Collection

The data collection process consists of collecting metrics data, transforming the data into a usable form by way of XML transformation, integrating manual classification data, and exporting the combined data to a format compatible with our analysis software.

The collection of metrics data begins with the import of the existing Java-based software system into the Eclipse platform and using the Eclipse plug-in Metrics [8] to generate the required metrics data. The metrics data is then

exported into an XML file for further processing.

An Excel XML spreadsheet file is then created from the XML-based metrics file using an XSL transform. This allows the data to be viewed or further manipulated in an easy to use environment. The spreadsheet file is integrated with the data collected from the manual classification step, resulting in a spreadsheet that contains all metric and classification data. Though we prefer the use of Matlab in our project for the classification and analysis of data it would also be possible to perform some analysis using a spreadsheet program. The final spreadsheet is then saved to a simple comma separated value (CSV) file. The CSV file is suitable for import into Matlab for analysis.

3.3 Manual Classification Step

The *manual classification* step is used to classify each component (class) from the system into one of two categories: *concept* or *other*. The purpose of this step is to provide a means to train and validate the classifier. The *concept* category is made up of components or classes that are representative of concepts from the knowledge domain of the system. The *other* category consists of everything that does not fit into the concept category. For example, consider a parser component. For a compiler this would be part of the concept category, since a parser is a core concept in this domain. In a word processor, this would likely not be classified to the concept category, since a parser would only support the primary or core concepts in the knowledge domain.

The approach to classification used in this paper involves examining the naming, relationships with other classes or concepts, and role within the architecture for the candidate class. This information must be evaluated with regard to the domain knowledge in order to classify the candidate class. It should be noted that there are no rules about how to classify a component. The decisions are based on the experience of the analyst, and their expertise in the domain. It is assumed that different analysts will produce slightly varying results. For our examples, we assumed any resulting classification that can be validated by a group of experts would be roughly equivalent to any classification we could arrive at or that the resulting error would not have a significant impact on the approach.

3.4 Machine Classification

The machine classification process consists of creating a SVM classifier from the data set then running the SVM classifier on the data set.

The data set is randomly broken up into two sets known as the *training set* and the *test set*. Both sets are of approximately equal size and consist of the metric data as well as the classification for all classes in the set.

The SVM classifier is created using the training set by an SVM algorithm implemented in Matlab. The metric data, in

essence, is the domain of the classification function and the classification data is the range. The result is a set of support vectors that will be used to perform classification runs. The SVM algorithm takes a parameter, which in the case of the radial basis function is conceptually related to a radius of influence for a given support vector, that is selected by a process known as *cross-validation*. Cross-validation allows us to make an intelligent parameter selection by partitioning the training set into n partitions, this technique is designed to reduce the risk of overfitting. We then sample over a range of parameter options giving a set P of potential parameters. For each parameter in P we train n different classifiers by removing 1 of the n partitions from the training set for each classifier. We then evaluate the accuracy of the classifier on the remaining partition for each of the n different classifiers and compute an average accuracy for the classifiers with the given parameter value. The result is an optimal selection of the parameter from P based on how well the classifiers created with that value generalize to the n test partitions in addition to the accuracy they achieve over the partitioned training set. Once the parameter selection is finished we train a classifier with the given value over the complete training set.

The SVM classifier is used to analyze the metric component of the test data set, this step generates a predicted classification result. The predicted or observed classification is compared with the manual or expected classification derived manually from the class.

4. Example

In this section, we illustrate one of the potential uses of our approach. Specifically, we show how the concept identification technique can be used as part of a reverse engineering activity.

Panda [9] is a natural deduction proof assistant written in Java and developed at Arizona State University. The tool is used primarily to support education of natural deduction as a proof technique. We had access to the original class and statechart diagrams that were used for development of Panda and thus used that information as a sanity check during identification of concepts. Panda's size is 84 classes and about 9 thousand lines of code (KLOC).

The architecture of Panda is best described as a GUI based model view controller application. The model consists of classes implementing logical constructs. The view consists of classes implementing Java GUI elements. The controller consists of classes implementing the use cases that can be applied in a proof.

The basic process that we are using in this example is the following. First, we generate a class diagram for the system. Figure 3 shows a diagram for the Panda package in the Panda system. In addition, we generate the relevant metrics using the original source code for Panda. Second,

we use the SVM to identify those classes that are concepts in the system. Third and finally, we create an abstraction of the original class diagram that utilizes only those classes that were identified as concepts.

As shown in Figure 3, the Panda package has 45 classes. When we apply the SVM to the Panda data, 24 classes are filtered. When compared to a manual classification we performed, the automated classification produced 1 false positive and 4 false negatives. A more detailed evaluation of the approach is described in the next section.

Figure 4 shows the resulting class diagram using the information gathered using the SVM. As shown in the diagram, the resulting class structure is a more abstract representation of the original system. Specifically, it focuses more on domain concepts rather than details of the implementation. In this case, the remaining classes are related to logical formulas and the commands that operate upon them (e.g., the applicable inference rules). Filtered classes consisted of user interface classes and other implementation details.

5. Evaluation

Two software systems were used to evaluate our approach. In this section we first describe the systems analyzed and the statistical methods used to evaluate our results. Then we present the results of several experiments on data sets collected from two case study systems.

5.1 Description of Example Systems

Panda, as described earlier, is a natural deduction proof assistant. In the context of the concept identification work, Panda is used as a small system for illustration purposes.

Scarab [10] is a web-based defect tracking system based on the Turbine [11] and Torque [12] frameworks. Scarab was designed to be a replacement for defect tracking systems like Bugzilla [13]. The size of Scarab is 585 classes and 128 KLOC.

Turbine is a model view controller framework for web-based applications. Torque is an object-relational mapper for Java classes that allows objects to be persisted to the database. Scarab therefore has the architecture of a MVC web application whose model is persisted to a database.

5.2 Statistical Analysis Method

In order to evaluate the accuracy of the classifier, we performed a statistical analysis of the classification results. We calculated the *sample error* from the test set using Equation (6) [7] where n is the number of samples in the test set S , f is the function specified by our manual classification mapping the data point x to one of the two classes, h is the *hypothesis* generated by the learning algorithm, and the quantity $\delta(f(x), h(x))$ is defined in Equation (7). When the hypothesis disagrees with the manual classification, the sample error increases and $\delta(f(x), h(x))$ will be 1 for any

instance x where the predicted classification $h(x)$ does not match the expected classification $f(x)$.

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x)) \quad (6)$$

$$\delta(f(x), h(x)) = \begin{cases} 1, & \text{if } f(x) \neq h(x) \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

We can estimate the *true error*, the error of the whole population, from the sample error using Equation (8) if we can meet the criteria in Figure 5. Here z_N is chosen based on the confidence level of the estimate.

$$error_{\mathcal{D}}(h) = error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}} \quad (8)$$

-
1. $n \geq 30$,
 2. the hypothesis commits r errors over the n samples,
 3. the sample, or test set, S contains n data points which are drawn independently of each other,
 4. the sample is chosen independently of the hypothesis, and
 5. the sample is chosen according to the probability distribution \mathcal{D} of the population.

Figure 5. Criteria for calculation of *true error*

Criteria 1 thru 4 of Figure 5 are met by the methods outlined in Section 3.4 and the size of the data set. Criteria 5 is more interesting since the dependency on the probability distribution requires that we test for a significant difference in the distributions of the test samples and the distribution of the population.

In order to confirm Criteria 5 we can perform the Kolmogorov-Smirnov hypothesis test on the population and sample distributions to test for significant difference. We formulate our hypothesis for this test in Hypothesis 2.

Hypothesis 2 *There is no significant difference in the distribution of the population and the sample data.*

The p -value of the Kolmogorov-Smirnov hypothesis test is the statistical significance, and α is the probability of rejecting the null hypothesis when it is in fact true. We will reject the null hypothesis if the p -value is less than α . Table 2 shows the results for each of the necessary test sets with an α value of 0.05 representing a confidence level of 95% for the test.

Using a t -test we have a toolset in place to detect situations where, for example, the accuracy is good, but there is no significant difference between the expected value (or

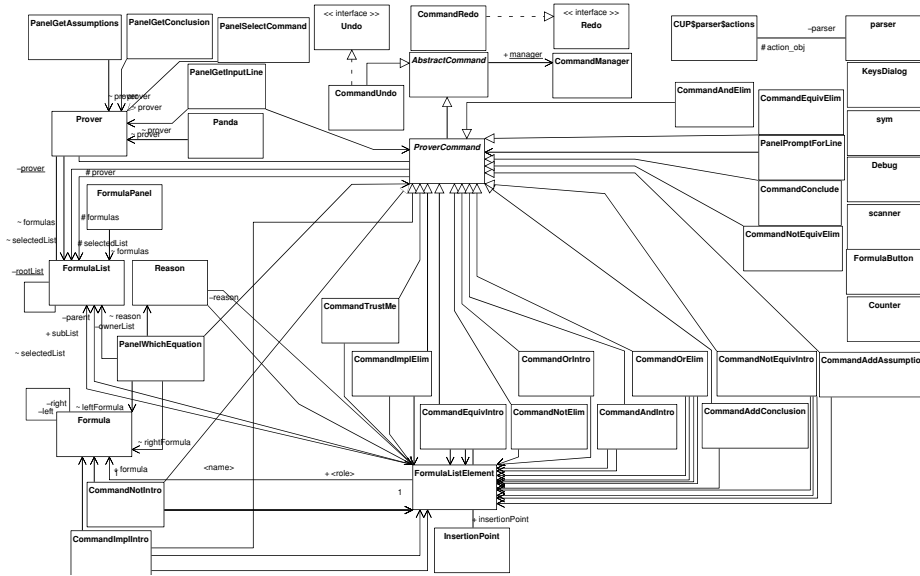


Figure 3. Panda class diagram

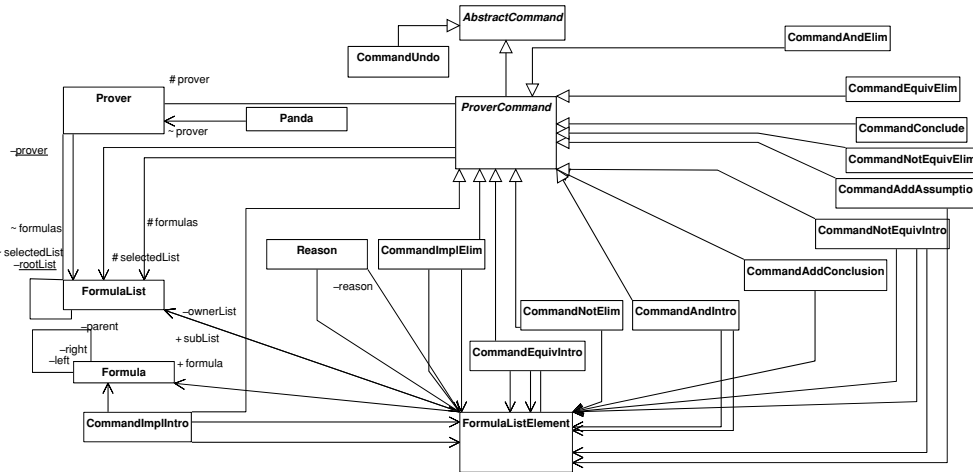


Figure 4. Panda class diagram after filtering

Table 2. Kolmogorov-Smirnov test $\alpha = 0.05$.

Data set	p -value	Reject Hypothesis 2
1 (Panda)	0.9922	No
2 (Scarab)	1.0000	No

mean) of the sample distribution and the measured accuracy. This could happen, as an example, in the case where the distribution of the expected results is 80% negative. The accuracy of the observed results could be 80% just by always guessing negative. The t -test will help to discover these situations by testing Hypothesis 3.

Hypothesis 3 *There is no significant difference in the mean of the sample data and the accuracy measured on that sam-*

ple data.

Rejecting Hypothesis 3 is a strong indicator that the accuracy is significantly different than the distribution of the data. Intuitively, a significant difference in the accuracy and the mean indicates that the classifier has learned something other than the expected value of the data set. As such, in Table 3 which shows the 95% confidence interval on the difference between population mean and sample accuracy, rejection of Hypothesis 3 is desirable.

5.3 Test Sets

Table 2 shows test sets 1 and 2 are randomly sampled from the Panda and Scarab data sets and are shown to be representative of the population they are sampled from by the acceptance of Hypothesis 2. Table 4 shows a mapping

Table 3. Results of Hypothesis 3 t -test.

Experiment	p -value	Reject Hypothesis 3
1 (Panda)	0.0198	Yes
2 (Scarab)	7.6288×10^{-11}	Yes
3 (Scarab KNN)	0.0753	No

of the test sets to the experiments they are used in.

Table 4. Test data mapping to experiments.

Experiment	Test set
1 (Panda)	1
2 (Scarab)	2
3 (Scarab KNN)	2

5.4 Experiments

This section shows the results of the seven experiments we have conducted thus far using the approach outlined in Section 3. The results are summarized in Table 5, which shows the sample accuracy along with predicted bounds of the *true accuracy* for N samples.

5.4.1 Experiment 1: Panda.

For this experiment we used the data set collected from Panda using the methods outlined in Sections 3.2 and Sections 3.3. The process of classifying Panda took approximately four hours but given our previous experience with the software this may not be an indicator of the time required for an engineer looking at the system for the first time. We followed the process outlined in Section 3.4 to split the data into training and test sets, and used the training set to generate the SVM classifier used in this experiment. The purpose of this experiment was to attempt to identify evidence for the hypothesis using a fairly small system. We wanted to see if it was worth while to commit to the process for a larger system. The data set for Panda is under 100 elements and while a larger data set would better capture any generalizable properties of the hypothesis, this data set was simple to obtain and encouraged further analysis.

We used a SVM classifier based on a radial basis function kernel as in Equation (3) of Section 2.3.1 with the slack term of $C = 10$. Results are shown in Table 5. Accuracy was 80.43% for this sample. The t -test analysis (as indicated in Table 3) showed that the accuracy of the test versus the expected value of the results collected from the software were significantly different at the 95% confidence level, in other words the probability that the accuracy is different than the mean of the distribution is over 95%. These results were encouraging for a data set of this size, prompting us to proceed with the data collection effort for Scarab.

5.4.2 Experiment 2: Scarab.

For this experiment we used the data set collected from Scarab. Classification of Scarab required significantly more

time than classifying Panda with over 40 hours of effort expended. This may have been a function of less familiarity with the system as well as the larger size. The process outlined in Section 3.4 was used to split the data into training and test sets, generate an SVM classifier for this experiment, and collect results. The purpose of this experiment was to validate the approach for a much larger data set than that used in the Panda experiment. Scarab is also a program that is more representative of systems in actual use.

The SVM classifier used was based on a radial basis function kernel with parameter of $C = 10$. Results are shown in Table 5. Accuracy for this sample set was 81.39%, and the t -test analysis showed that the accuracy of the test versus the expected value of the results were significantly different at the 95% confidence level. These are solid results that show quite a bit of promise for future research as it indicates that a fairly high degree of accuracy can be obtained using a moderate sized training set.

5.4.3 Experiment 3: Scarab Revisited.

For this experiment we wanted to repeat the Scarab experiment with a different machine learning algorithm for the classification. We chose the KNN classifier in order to form a simple baseline for comparison to the SVM results collected previously. The purpose of this experiment was to eliminate the chance that there is something special about the classification algorithm based on SVM.

In this experiment we used a value of $k = 3$. Results are shown in Table 5. The only parameter to the k -nearest neighbor algorithm is k . The sample accuracy is 76.92% which comes close to the accuracy obtained using SVM on the same data set in Section 5.4.2. The t -test analysis does not show a significant difference in the accuracy of the test versus the expected value as shown in Table 3. However, this result provides support for our hypothesis given that with 92% confidence we can show significant difference, this represents a 1 in 12.5 chance that the hypothesis is incorrectly rejected for this test. Given that we are over 99.999% confident in the results for experiment 2 and this test is only designed to confirm those results with a different machine learning algorithm it seems acceptable to view this test as a successful validation of those results even at only 90% confidence.

5.4.4 Discussion.

Table 3 gives us an indicator of how well the accuracy is measured. A small p -value is an indicator that there is a significant difference between the accuracy and the mean. Imagine a coin that is unfairly biased towards heads such that the expected value of heads is 80%. If we were to play a game where the coin is flipped repeatedly and the player must try to guess the result, the player would eventually discover the bias and begin guessing heads every round of the game. This learned strategy would lead to a measured accu-

Table 5. 99% confidence interval on accuracy of classification. (LB = lowerbound, UB = upperbound)

Experiment	Sample Accuracy	N	True Acc. LB	True Acc. UB
1 (Panda)	80.43%	46	68.97%	91.90%
2 (Scarab)	81.39%	403	77.59%	85.19%
3 (Scarab KNN)	76.92%	403	71.51%	82.34%

racy of the player at about 80%, in other words there would be no significant difference between the expected value of the experiment and the accuracy of the guess. In our results we believe that the machine is really making good predictions, because there *is* a statistically measurable difference between the expected value of the data set and the classification guess of the machine for all of the SVM predictions. So the results show support for Hypothesis 1.

These results lead to the idea that a SVM classifier trained within an application will work well on that application. This result has practical applications since a software engineer trying to identify concepts within a system with a large number of classes could start by classifying parts of the system. The approach described could be used to predict results for the remainder of the system. As the software engineer accepts or corrects the predictions they become more accurate. So the approach can be used as an interactive concept identification assistant that becomes more accurate as the system is classified. Those familiar with SPAM filters such as SpamAssassin [14] will recognize this concept, the more you train it the less time you spend dealing with SPAM. In presentation of this work to peers many practicing engineers appreciate the potential time savings. Their work with systems that have a large number of classes shows the immediate practical benefits to an approach that trains a computer to do a repetitive classification task.

5.5 Threats to Validity

The threats to validity include internal and external factors. The internal factors include errors in the statistical analysis, and overfitting of the data set. The external factors include manual classification errors, and inaccurate or poor definition of the concepts in the domain. We will describe each in further detail below.

The primary internal threat to validity that should be addressed is the difficulty in deciding what constitutes a *good result*. The difficulty stems from the distribution of classifications where there are more negatives than positives. In other words there are fewer classes representing concepts than implementation. It has been shown in Section 5.2 that the accuracy of the prediction can not be used in isolation, but must be considered along with the *t*-test analysis of the distributions' expected value verses the accuracy. This gives a more complete picture of the result showing some indication as to how significant the accuracy is. The assumption here was that results that had better than average accuracy

along with a significant difference in mean were good results that support our hypothesis.

Overfitting is a threat to validity that effects any machine learning approach, but is only a secondary threat to the validity of our approach. Overfitting means that results will not generalize well to other data sets, since the machine has learned a pattern for a specific data set. In other words if the SVM is overfitted to the data then we could not expect good results from a different data set classified with the same machine. At this point we are not that concerned about generalization to other data sets as we have a practical application of the approach in the classification of a single large system. In future work the potential of overfitting will play a larger role in the threat to internal validity of our approach if it is expected that our approach will produce general results.

The primary external threat to validity is misclassification during the manual classification process. There are two possible scenarios that result in the introduction of inaccurate analysis into the system and they are either a logical error or a typographic error on the part of the engineer. We have worked with enough software engineers to know that each has a different opinion on any given subject, these differing opinions would appear in the system as logical errors. However, this assumes that one of the engineers is more correct than the other within some externally specified system of objective truth that likely does not exist, and this ignores the possibility of having more than one correct representation of the system. We must concede that any accurate automated classification is only as accurate as the engineer who trained it, but this really is not problematic as long as the engineer is consistent and in fact is representative of the results of a manual classification. The typographic errors that could have been introduced into our data set were minimized by careful data entry, along with rechecking each result against the class. Though we believe we have minimized the errors as much as possible we have no way to measure this using the current process, and as is the case with many machine learning applications we can not predict the effect of a classification error on the resulting classifier.

A secondary external threat to validity is the design of the software itself. If poorly designed software systems are introduced it is difficult to predict the results. As an example imagine a single concept being represented by two classes. There may be instances where the lack of cohesion makes sense but it may not be possible to decide where.

6. Related Work

Biggerstaff et al. [2] describe the assignment of human oriented concepts to implementation oriented concepts. This process is a matching between preexisting notions about the system or existing domain knowledge to textual representations of the system implementation. While the definition of the concept assignment problem given fits well with the filtering application we present here there are a few key differences. The approach used by Biggerstaff et al. uses source code where we use metrics based on the source code. We use a machine learning approach that does not involve declaratively creating an expert system. They make a strong case for the methods used to recognize concepts, using at times a filtering of the source code. Our approach could be added to the toolset mentioned here as an additional filtering technique, though in time we will automate more of the process to the point where interaction with the user is very limited.

Svetinovic et al. [15] discuss concept identification from a forward engineering perspective illustrated in several case studies. The claims of misuse or misunderstanding of object-oriented domain analysis are worth noting since the automated identification of concepts requires that those concepts are represented within the implementation, implying they were designed into the software. They identify many of the concerns we have raised in our discussion of external validity. Primarily, that there is not one single agreed upon way of designating what is and is not a concept in a software system, according to the paper this is a fundamental difficulty with object-oriented domain analysis.

Merlo et al. [16] describe a design recovery process which uses a machine learning approach to link text from source code to concepts identified by a domain analysis. This is essentially a partial automation of a portion of the work performed by Biggerstaff et al. The machine learning algorithm used is based on neural networks. The approach differs from ours in that a change in domain requires a new neural net to be trained. That is not necessarily the case with our SVMs as they can be applied to programs from very different domains but currently without significant accuracy.

Hsi et al. [17] approach recovery of ontology by manual construction of an *interface map*. The interface map is a depth first traversal of all elements of the software's interface. The approach then uses the interface map to generate a semantic network. The semantic network is then used as the basis for calculating graph theoretic measures that are used as indicators of *core concepts* of the ontology. The approach differs from our work in that it is based on construction and analysis of a graph. The evaluation does not include any comparison to a *control group* in order to express the accuracy of the approach. Each of the varying methods introduced to recover the ontology produce differing ontologies, and no method of reconciling differing

results is presented. Advantages of our approach include the use of a control group in the form of the manual classification results to show the accuracy of the approach, and we only produce one ontology based on the metrics that we use. This paper presents an interesting set of metrics that may add to our approach in future work.

Software architecture reconstruction is a recent development that represents a reverse engineering process designed to recover architecture specific information. As Favre [18] points out, software architecture is not well defined. Software architecture can only be defined in terms of the audience to which it is presented. An experienced software engineer will recognize the overloaded use of the term to have slightly differing meanings based on who is the target audience. Obviously, the project manager, technical lead, upper management, customers, fellow engineers with similar experience, the newest addition to the team who finished his degree only a few months ago, as well as other stakeholders all have a different perspective on the software architecture. van Deursen et al. [19] seem to ignore this ambiguity but the approach of view-driven software architecture reconstruction called Symphony is introduced. This paper essentially codifies best practices that have been observed by the authors in the actual practice of software architecture reconstruction. Placed in the context of software architecture reconstruction and in the vocabulary of Symphony our work would be a type of mapping from a source view to a target view, the target view being a static graph of the core concept classes described by a UML class diagram.

Marcus et al. [20] describe a method of using latent semantic indexing to map natural language concepts to the locations of those concepts in source code. Zaidman et al. [21] show that webmining techniques can be applied to source code to uncover important classes, similar in nature to our core concept classes.

7. Conclusion and Future Work

The results strongly indicate that there is a relationship between the given metrics and the identification of concepts. The relationship that is captured by the SVM classifier produces above average results over a single software system. However, a larger data set is needed to comprehensively verify these results. Additionally, we have made no attempt to optimize the metrics that were selected for this set of experiments. In fact we initially simply selected all metrics available from the output of the metrics software used thinking that we would have to optimize the feature vector in order to obtain acceptable results. Obviously this is an area of future research since optimal results may allow a stronger formulation of our hypothesis, particularly in regards to what type of metrics are the best indicators.

The example filtering shown in Section 4 effectively demonstrates one of the practical applications of the clas-

sification technique presented here. The class diagrams depicted in Figure 3 and Figure 4 show a visual demonstration of the effectiveness of the filtering process. Figure 4 is simply easier to understand due to the smaller number of classes. The effect is even more obvious and useful on systems with many more classes.

Work on this project is ongoing. We currently have plans to extend our data sets significantly. This involves classifying several different systems from ideally orthogonal domains. We are also investigating other possible applications of this technique. Plans are underway to develop an Eclipse plugin to allow this technique to integrate with the Eclipse [22] environment, providing online classification suggestions to the developer. This represents an exciting practical application of the results even lacking generalization to more than one system, which is another area of future interest. We are also interested in increasing the accuracy of the classifier. This involves developing a deeper understanding of the relationship that the classifier is detecting. We will be investigating the use of different kernels as well as performing more analysis of the features to determine what metrics are the best indicators. We will explore the learning curve of the classifier, how quickly can it be trained to acceptable accuracy. Additionally we are interested in training optimizations, some training sets are better than others. Is it possible to select them prior to manual classification.

References

- [1] George A. Miller. The magical number seven, plus or minus two. *The Psychological Review*, 63:81–97, 1956.
- [2] Ted J. Biggerstaff, Bharat G. Mitbander, and Dallas E. Webster. Program understanding and the concept assignment problem. *Communications of the ACM*, 37(5):72–82, 1994.
- [3] Brian Henderson-Sellers. *Object-Oriented Metrics: Measures of Complexity*. Object-oriented series. Prentice Hall PTR, 1996.
- [4] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, New York, NY, USA, 1992. ACM Press.
- [5] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.
- [6] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 2002.
- [7] Tom M. Mitchell. *Machine Learning*. WCB/McGraw-Hill, 1997.
- [8] Frank Sauer. Metrics 1.3.6. [Online] Available <http://metrics.sourceforge.net/>.
- [9] Greg Hodgdon. PANDA : Proof Assistant for Natural Deduction Analysis. Technical Report of MCS Project, Arizona State University, November 2001.
- [10] Tigris. Scarab. [Online] Available <http://scarab.tigris.org/>.
- [11] Apache Organization. Turbine. [Online] Available <http://jakarta.apache.org/turbine/>.
- [12] Apache Organization. Torque. [Online] Available <http://db.apache.org/torque/>.
- [13] Bugzilla Organization. bugzilla.org. [Online] Available <http://www.bugzilla.org/>.
- [14] Apache Organization. The apache spamassassin project. [Online] Available <http://spamassassin.apache.org/>.
- [15] Davor Svetinovic, Daniel M. Berry, and Michael Godfrey. Concept identification in object-oriented domain analysis: Why some students just don't get it. In *RE '05: Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 189–198, Washington, DC, USA, 2005. IEEE Computer Society.
- [16] Ettore Merlo, Ian McAdam, and Renato De Mori. Feed-forward and recurrent neural networks for source code informal information analysis. *Journal of Software Maintenance*, 15(4):205–244, 2003.
- [17] I. Hsi, C. Potts, and M. Moore. Ontological excavation: unearthing the core concepts of the application. In *WCRE 2003: Proceedings of 10th Working Conference on Reverse Engineering*, pages 345–353, 2003.
- [18] J. M. Favre. Cacophony: metamodel-driven software architecture reconstruction. In *WCRE 2004: Proceedings of the 11th Working Conference on Reverse Engineering*, pages 204–213, 2004.
- [19] A. van Deursen, C. Hofmeister, R. Koschke, L. Moonen, and C. Riva. Symphony: view-driven software architecture reconstruction. In *WICSA 2004: Proceedings of Fourth Working IEEE/IFIP Conference on Software Architecture*, pages 122–132, 2004.
- [20] Andrian Marcus, Andrey Sergeyev, Vaclav Rajlich, and Jonathan I. Maletic. An information retrieval approach to concept location in source code. In *WCRE '04: Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04)*, pages 214–223, Washington, DC, USA, 2004. IEEE Computer Society.
- [21] Andy Zaidman, Toon Calders, Serge Demeyer, and Jan Paredaens. Applying webmining techniques to execution traces to support the program comprehension process. In *CSMR '05: Proceedings of the Ninth European Conference on Software Maintenance and Reengineering*, pages 134–142, Washington, DC, USA, 2005. IEEE Computer Society.
- [22] Eclipse Organization. Eclipse. [Online] Available <http://www.eclipse.org>.