1. Introduction

Software changes everyday. During software change, one of the most common and important activities that developers undertake is *concept location in source code* (i.e., finding what to change in the source code). Meanwhile, managers need to *estimate the development effort* of the proposed changes. In addition, managers and developers alike must know the impact of the changes on the software quality and use *software quality prediction tools* to get the answer. What do these three important software engineering activities have in common? State of the art solutions to these problems rely on the use of data mining techniques (e.g., classification, clustering, feature subset selection, regression, rule learning, text retrieval, etc.). *The quality of the solutions to these tasks depends on the software engineers' ability to customize generic data mining algorithms to specific software engineering data.*

A common and critical problem with data mining solutions to software engineering tasks is that most algorithms use a variety of parameters, which are domain and data specific. *Reusing a set of parameter values from one application to another usually leads to poor results.* For example, Bayes classifiers use the "M" and Laplace factors to handle low frequency counts. Standard default values are M=2 and L=1, [121] but there is no telling what is the best configuration for a particular domain. *These standard parameter values are usually derived empirically from applications in domains outside software engineering.* For example, default parameter values used by many information retrieval techniques are established based on data from text retrieval tasks on natural language corpora. Recent empirical research in applications of text retrieval in software engineering [18, 30, 35] showed that best results are in fact obtained for configurations different from those used in natural language retrieval applications. More than that, the quality of the results is highly dependent on these configurations [59, 97]. The observation also holds for many defect prediction approaches [70]. For most data mining algorithms, the space of possible parameter values is extremely large. Finding the right combination is often based on intuition rather than on science. Without a well defined methodology that allows all these parameters to be effectively customized for applications in software engineering, *the research in the field is limited to proof-of-concept efforts and generalization of the results is rarely achieved or ensured*. This customization problem is one of the main reasons that stop many data mining solutions to software engineering problems from migrating from the research labs to industry.

1.1. Research goals and anticipated results

The goal of our research is to *improve software engineering problems that are solved using generic data mining algorithms*. We specifically address three important software engineering tasks: *concept location* in source code during software change, *defect prediction*, and software *development effort estimation*. These three tasks are exemplars of other software engineering activities that share common solutions based on data mining, such as: search based software engineering applications, applications based on mining software repositories, change prediction and impact analysis, traceability link recovery between software artifacts, tasks using recommendation systems, etc. We expect that our work will extend to all these applications, will result in their improvement, and it will accelerate industry adoption.

In order to achieve these goals we will address the following research tasks:

1. **Define and evaluate a mechanism for the efficient configuration of data mining applications on software engineering data.** The mechanism will be based on tool support for which we will develop a framework that will be instantiated for a variety of combinations of *data mining algorithm* x *software engineering task* x *software system data*. Each algorithm has a set of parameters, which needs to be customized for the given task and software. The resulting customization agent will assist the software engineering user in efficiently selecting the best configuration, which includes a set of algorithms and their parameter values, customized for the particular task and software system. What is required here is not a brute force exploration of all the options; rather we need an intelligent exploration of the configurations that work best for the given context. The goal of the customization agent is to find the best configuration without trying all possibilities, which is too expensive to be practical.

2. **Improve state of the art approaches to *concept location*, *defect prediction*, and *software development effort estimation*.** We will instantiate the previously defined framework for these three tasks and we will conduct extensive empirical evaluation in academic and industrial settings. We will augment several existing solutions to these problems with the customization agent and deploy them with our students and our industrial partners.

   These three software engineering tasks have properties that make them representative for different categories of tasks, each requiring somewhat different evaluation techniques. Concept location is a semi-automated technique, which heavily involves developers. Data collection in this case is slow and the validation of results is prone to human error. Effort and defect prediction are often completely automated, which allows for faster and less error prone evaluation. Defect prediction uses a large variety of data, which means we can analyze fewer data points. On the other hand, effort prediction relies on less data and allows for generation of results faster. The work on this research task will also provide with an implicit validation of the usefulness of the proposed customization methodology (from the first research task).

1.2. Intellectual merit

Software engineering (SE) applications of data mining (DM) face unique challenges [118] compared to other fields, where DM is used every day (e.g., medicine, bioinformatics, business, finances, security, etc.), as in software engineering the data is extremely heterogeneous. For example, a DM algorithm applied on genetic information from one species can be easily used, with the same configuration, on genetic information of other species. The data is different, but the underlying structure is very similar. On the other hand, in SE no two real-world software systems are that much alike, even within the same application domain. Hence algorithm customization is needed for every system, sometimes even for a subsystem, when developed by others.

The PIs have more than two decades of combined research experience in applying data and text mining techniques to solve SE problems, such as: defect prediction, concept location in source code, effort and cost prediction, impact analysis and change prediction, recovery of traceability links between software artifacts, bug triage, etc. [9-10, 12-13, 15-16, 19, 26, 30-31, 36-38, 40, 49-50, 54, 56, 58-59, 61-63, 65-89, 97, 99, 101, 104, 109-110, 115]. In all instances, we had to make significant adjustments to the default parameters of the various DM algorithms we used. For example, one of the PIs (Menzies) worked with NASA contractors for a decade trying to inject DM technology into their projects [70]. In many cases the contractors were routinely using sub-optimum settings. Addressing these gaps in the field will help this research area mature and generate practical solution for the software industry.

This proposal blends together research in DM and SE and aims to transform the algorithm parameter selection problem from art into science. The solutions we propose will open the door to new applications of DM to SE problems, not addressed today. This field of research is poised to grow at the same fast pace as SE data grows. With the growth in size and complexity of software systems and data, it is impossible for individuals to understand it and reason about them without tool support. We foresee a paradigm shift in SE, where more and more SE tasks will be reinterpreted as optimization or search problems, which require DM-based solutions. This research will facilitate such a transformation.

1.3. Motivations, challenges, and the importance of the proposed research

Data mining techniques have been proposed by many researchers to extract what is relevant to the stakeholders (i.e., developers, managers, testers, etc.) and help them understand data about a software system, its development process, and to make predictions about its future quality, cost, and evolution. Many traditional SE tasks and newer research areas already rely heavily on the use of DM techniques. The newer approaches include: search based software engineering (SBSE), mining software repositories (MSR), recommendation systems in software engineering (RSSE), predictor models in software engineering (PROMISE), etc. A recent cover article on data mining and machine learning in *IEEE Computer* features software engineering as one of the four major application areas [108]. The conclusion of the article on data mining in SE is that "much work needs to be done to further adapt general-purpose

mining algorithms or develop specific algorithms to satisfy the unique requirements of SE data and tasks"[118]. Our proposed work aims at pushing this research field forward by providing software engineers with a mechanism for adapting general-purpose DM algorithms to specific SE data and tasks.

DM in SE poses unique challenges [118], as many SE tasks demand the mining of multiple correlated data types, including both sequence and text data, or text data and graph data, used together to achieve the best results. The data to be mined is often very large; e.g., when searching in open source repositories, execution traces, textual data in software, static analysis data, etc. Many DM tasks are performed off-line, but some require just-in-time solutions. Last but not least, software engineers are no experts in data mining and data mining researchers are no experts in SE data. This poses problems when using off the shelf solutions for SE problems. One of such problems is the focus of our proposed research – customizing DM algorithms for the task and data at hand.

We describe here a detailed example to highlight the scope of this problem. We will revisit the example in Section 3.2, in order to instantiate the proposed customization agent. We extensively used text retrieval methods to help developers search the source code during software change (i.e., to perform concept location). In such an application the source code is transformed into a document collection (i.e., corpus), which is indexed with an information retrieval (IR) method. The user then formulates queries used by the IR engine to find similar documents, which are then matched to the source code. The following choices and parameters arise in this case:

- What granularity should the corpus have? That is, what should be a document in the corpus: a class, a file, or a method? → at least 3 choices.
- Should we include comments or not in the corpus? → 2 choices (yes/no).
- Should we stem the terms in the corpus or not? → 2 choices (yes/no).
- If so, which stemmer to use? → at least 2 choices.
- Should we filter out commonly used terms (i.e., stop-words) → 2 choices (yes/no).
- Should we split identifiers or not? → 2 choices (yes/no).
- Should we keep the original form of the identifier? → 2 choices (yes/no).
- Which IR method to use for indexing: a vector space model (VSM), Lucene, Latent Semantic Indexing (LSI), Latent Dirichlet Allocation (LDA), etc. → at least 5 choices.
- These IR methods have their own parameters. For example, in LSI is important to establish the dimensionality reduction factor, LDA has a user defined number of topics, and so on. → in average each IR method has at least 3 parameters with many possible values for each.
- What similarity measure should we use between two documents? → at least 2 choices.
- Should we include relevance feedback in the process? Relevance feedback allows for the automatic reformulation of queries. → 2 choices (yes/no).
- If so, which relevance feedback algorithm to use? → at least 3 choices.
- Relevance feedback methods have several parameters each. → at least 4 parameters with many possible values each.
- Should we cluster the corpus or the results? → 3 choices.
- If so, what clustering method to use? → at least 3 choices.
- Most clustering methods use some parameters (e.g., think of the k from k-Means). → at least 1 parameter with several possible values.

One can easily see that the space of possible parameter combinations used to calibrate the tools is staggering. Let us assume we want to make changes for a medium sized software system with 10,000 methods. The naïve approach to decide what configuration to use is to try each combination of those billion possible by searching among the 10,000 methods for a number of changes (for which we know the results), find the configuration that performs the best, and then validate the best configuration on another set of changes (for which we also know the results), observing if the performance is maintained or improved. Some of the steps here are fast (order of seconds), e.g., running a single search. Other steps take minutes, e.g., preprocessing the source code, while some of the steps take hours, e.g., indexing the source code. In other words, this approach is far from practical. The alternative, adopted usually by

researchers and practitioners is to use "default" configurations established in other application fields – in this case the default values are established in text retrieval in natural language. This solution is rather an educated guess, which often leads to poor results. We expect that our proposed work will eliminate much of the guessing, while improving the time needed to select a good configuration. In turn, the DM results will help the users better comprehend the software engineering data and thus improve the performance of the underlying tasks.

2. Background and Related Work

More and more SE tasks are making use of DM algorithms due to their capability to extract important information from large amounts of data [118]. In this proposal we are focusing on the use of DM algorithms to address three such tasks, i.e., *concept location*, *defect prediction,* and *effort estimation*. We highlight here the DM based solutions to these tasks and their importance in software engineering.

2.1. Software defect prediction

Building high quality software and locating any defects as early as possible is extremely important, especially in the case of mission critical software. This is, however, a very *expensive* procedure and prone to *biases* when performed by software engineers, as they tend to focus more on some aspects of the system than others [46, 55]. *Software defect prediction* addresses these problems by constructing predictive classification models from code attributes to enable a timely and accurate identification of fault-prone modules. [45]. It is an activity that spans the entire life span of a software system. Numerous DM algorithms have been applied to address this problem. Lessmann et al. [45] offers one of the most extensive recent surveys on data mining algorithms used for this task. All these algorithms build predictors for modules being defective or not, using static code features extracted from a project (for an examination of methods to learn defect predictors from other features, see [92]). Lessmann et al. studied 16 DM algorithms, classified in six categories: statistical classifiers (which include Latent Dirichlet Allocation (LDA), Logistic Regression, Naïve Bayes classifiers, etc.), nearest-neighbor methods (including k-Nearest Neighbor and K-Star), neural networks (multilayer perceptron and Radial Basis Function Network), Support Vector Machine-based classifiers (SVM, Linear programming, etc.), decision tree learners (C 4.5, etc.), and ensemble methods (random forests, etc.). Recent approaches to defect prediction based on DM algorithms [8, 27, 51, 61, 70, 75, 89-90] are in the focus of our proposed work.

2.2. Software development effort estimation

Software development effort estimation is one of the key activities in the lifecycle of a software system, as the estimates produced by this activity are used as input to project plans, iteration plans, investment analysis, pricing processes, etc. In spite of its importance, it still remains largely an unsolved problem [91]. Model-based effort estimation (not involving a human) has been addressed by a broad range of algorithms, many of which represent DM algorithms. Myrtveit et al. [91] offer a taxonomy of effort estimation methods divided into "spare-data methods" and "many-data methods". When data is scarce, Myrtveit et al. report that case-based algorithms [114] may be most appropriate. On the other hand, when data is more plentiful, algorithms like neural networks, or regression [6] may be more suitable. Other work applying DM algorithms for software effort estimation include [71, 88] Note that while the interpretation of the output may differ, there are many cases where the same DM algorithms are used for both effort estimation and defect prediction. This is important in our proposed work, as we will leverage these similarities in solutions. For example, Logisitic Regression, neural networks, and nearest neighbor algorithms are used in both activities [45, 91].

2.3. Concept location in software

We address the *concept location* problem (also referred to as *feature location*, *concept assignment*, *concern/aspect identification*) in the context of software change. In this context, concept location is defined as finding the places in the software where changes are to be made in response to a change request, such as, a bug description or a new feature request. Other processes (i.e., impact analysis) are

usually employed to determine the extent of the change. In consequence, concept location is one of the most common and important activities undertaken by developers [105].

When working on large software systems, developers need tool support for concept location and DM algorithms have been used to provide this support. For example, [25] uses a pattern mining algorithm to mine execution traces of a program and extract features, whereas in [42] the authors make use of feature subset selection, clustering and pattern matching to find the set of canonical features in a software system and the implementation overlap between features. In [43] traces are seen as time series and a dynamic time warping is used to identify the features in each trace. Sequential pattern discovery was applied in [112] to identify the implementation of software features specified through a number of feature-specific task scenarios. Other DM algorithms applied in concept location include: subgraph mining [11], genetic and hill climbing algorithms [32], and Latent Dirichlet Allocation [47-48, 53]. Many solutions use text mining algorithms and formulate concept location as a text retrieval problem: [122] uses the Vector Space Model (VSM), Latent Semantic Indexing has been used in many different approaches [49, 62-63, 96-97, 99, 103, 119], including in combination with clustering execution traces [44], etc. We will primarily target the text mining approaches to concept location, many of which were proposed originally by the PIs [62-63, 96-97, 99].

3. Proposed Work

As we highlighted before, the nature of the data in the SE applications makes the customization of data mining algorithms difficult as there is no underlying common structure among software systems. However, we can take advantage of the specifics of SE applications. First, there is always a human in the loop. This means she can be part of the customization process and we should make use of this opportunity. Second, data in SE is not produced at the same (fast) rate as in some other fields. It is produced at a slow enough rate that we can take advantage of the times between bursts of data to do the customization. More than that, we can collect performance data with respect to prior customizations from which we can learn to speed up future customizations. This proposal aims at taking advantage of these possibilities. The humans that benefit from and are involved in the mining process of software data play different roles, such as, developer, manager, architect, etc. We will refer to all these as simply the "user" or the "software engineer".

This section presents the technical details of mechanism we are proposing for the customization process of the DM algorithms. We also present a detailed example on how to use this approach on concept location in software using text mining and relevance feedback (i.e., an instance of the three SE tasks we plan to address). We highlight the specifics of applications to defect prediction and effort estimation (i.e., the other two SE tasks we address), and we present the evaluation strategy we plan.

3.1. Selecting the best configurations for data mining applications in SE

The proposed mechanism for customization is supposed to help any developer of techniques that use DM algorithms to solve SE tasks. Each such technique should incorporate a customization component. Our proposal is to develop a framework, which can be instantiated for any number of combinations (i.e., context) of: *software engineering task (T)* x *software system data (D)*. Each instance will result in a customization agent (CA), which solves an optimization problem. The problem is to find the best configuration (C) given the current context T x D. The configuration C is defined as a set of DM algorithms (A), where each algorithm is customized with specific values of their parameters (P): $C = \{a_i(p_i) \mid a_i \in A, p_i \in P\}$. The technical solutions we propose draw from DM and optimization fields.

Figure 1 (see next page) shows the main components of the CA and how it works. The central idea of the CA is to find the best configuration for a particular data set by exploring just a small fraction on the total number of configurations. The CA assumes that not all the possible configurations lead to different effects. That is, the CA can assess new configurations, without exercising them, but rather by remembering what happened with past configurations. The CA will not be just a cache of past configurations. Rather it will be an automatic research assistant that can also propose new task executions. CA will be implemented using active learning [14] to reflect over the history of

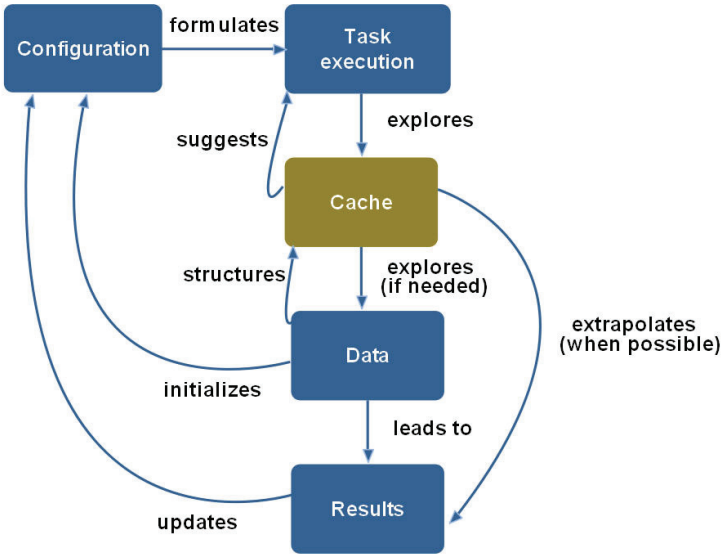

Figure 1. The customization agent workflow

configurations tried in the past to suggest the next most informative configuration to try next. The CA will have a component that tracks the configurations used in the past as well the results of those configurations (*C* x *R*). These will be stored together with the context (*T* x *D*). In short, the resulting CA will achieve the followings:

a. Store all the completed tasks and their results (*T*, *R*), with the respective configurations (*C*). The quality of the results also needs to be recorded in form of performance scores (*S*).
b. Measure the similarity between tuples of the following form: *software data* x *configuration* x *task* x *results*, that is, (*D*, *C*, *T*, *R*). This will support reusing parameter values learned from a previous version of the same system or from other, somewhat similar systems.
c. Efficiently group past data (i.e., configurations with results and context - *D*, *C*, *T*, *R*) and then perform a nearest neighbor analysis for new configurations. We propose to use a new clustering technique, called *bunching*, which combines a test case generation method (i.e., pairwise testing) with an incremental discretizer, described in Section 3.4.
d. Reflect over old tasks and their context to offer the most informative configurations to try, while blocking configurations that are uninformative (i.e., that fall into a "smooth" region where the results can be extrapolated from completed tasks).

We plan to use a Vector Space Model (VSM) [113] to record the tuples (*D*, *C*, *T*, *R*), as it is well suited for this kind of heterogeneous data. For every element of the context (*T* x *D*), we will have multiple vectors corresponding to the particular configurations and the results (*C* x *R*). This will allow us to reason about all this data at the same time. The CA will need to know if a particular configuration led to good results, so it will need to record *performance scores* (*S*) for each set of results. We distinguish two kinds of *performance scores*:

- Unsupervised scores (e.g. cosine similarity, which measures intra-cluster similarity);
- Supervised scores where the results have been labeled by some authoritative oracle. When such labels exist, is it possible to compare the estimate of the learner with the desired label. Such comparative measures are very common in information retrieval and include: (i) recall (e.g., the percentage of defective modules found by a defect predictor) and (ii) precision (e.g., the percent of correct estimates of the total estimates).

We have to keep in mind that for many SE applications, very little of the data is labeled by an oracle. For example, it is unlikely that any programmer will label the 20,000+ classes of a system as "relevant"

or "irrelevant" to their current task. Hence, the CA needs to make conclusions using mostly unsupervised scores or a small minority of supervised scores. When the CA is deployed and used, one of the consequences will be the accumulation of supervised scores over time. When a performance score is available from supervised examples, after multiple runs of the learner, we will collect both the *median* and *variance* of those scores.

The CA will be tagging all configurations with the performance scores (if they are available) or with "nil" (if not). The CA then groups the configurations and results into many clusters using the *bunching* algorithm described in Section 3.4. We say that a cluster is *linear* if, on average, performance scores can be interpolated from neighboring instances (and, otherwise, it is *non-linear*). In linear clusters, if a cluster contains three items *x, y, z* and *y, z* are the nearest neighbors to x, then the performance scores of *x* should be the same as a linear interpolation between *y* and *z*. We adopt here an assumption, just like in analogy-based effort estimation (e.g.. [114]), that the k^{th} nearest neighbors to the test instance are *linear*; i.e., the effort for one instance can be averaged from its neighbors. Consequently, if a new proposed configuration falls into a linear cluster, then the CA needs not waste CPU time executing that configuration. Rather, it just interpolates the result, and returns.

Less obviously, once linear clusters are indentified, it is wise to avoid them (when choosing the next configuration), as new configurations that fall into linear clusters add no information about the effects of different configurations. We discuss below methods that will allow the CA to avoid linear clusters.

As mentioned above, for many SE applications, most of the data will be tagged "nil" (i.e., only a small subset of the data will have supervised labels). Hence, before the CA can interpolate, it will need to handle the "nils". One solution is to pester a software engineer for thousands of labels for all instances in a data set. Given the time constraints of a working engineer, this is impractical. A more practical solution is to employ *label propagation* methods that infer missing labels from existing labels. We say that a cluster is *labeled* if there exists (a) some minimum number of supervised scores; and (b) there is a majority label in that cluster. Clusters with few labels can be populated via the *majority labeling* algorithm [17]. In majority labeling, all items in a cluster get labeled according to the most frequent label in that cluster. We must note that it may be unwise to use majority labeling when supervised results in a cluster are rare. In this case, the CA will use a *cluster propagation* algorithm [123]. In cluster propagation, the labels from one cluster are inferred from neighboring with a higher frequency of supervised labels. This repeats transitively to other neighbors until all the clusters are labeled.

Once the clusters are labeled, the CA can be informative to group the clusters into regions by:

- *Categorizing each cluster as linear or non-linear* (as defined above) then ask a classifier to learn a model that predicts for cluster linearity.
- *Look for near-by clusters with very different labels*. If such feuding neighbors can be found, it is useful to find the data that separates the neighbors (this data will be a hyperplane that separates the clusters in an n-dimensional space – determined by our vector space representation). If we draw a line between the centroid of the feuding clusters, we can generate SE task for the data along that line (the data on the line represents configurations).

Once these regions are known, we can rule out large sections of the search space for different configurations by ignoring regions which are linear or, on average, produce results that are significantly worse than other regions. A human can now check the clusters, when needed. The key here is organizing the cache into a tree of clusters; after that comparing a new configuration with the old data is done faster than executing the task. Section 3.4 will explain how our clustering method can be significantly faster than performing the SE task for each configuration.

Another way to find large regions that can be ignored is by reflecting over the set of recorded results (*R*). *R* is different for each context *T* x *D* (i.e., task and data). For example: defect predictors select the subset that they suspect are defective; instance-based learners find the subset that is the k^{th} nearest to a test instance; informative retrieval learners select a subset that they think is relevant to a query; etc.

Regardless of the context, syntactically the results are stored the same way (subsets of the rows of data from our vector space). By clustering the set of results, we can find regions of the data that are exercised by similar configurations. In this way, we can learn equivalences between different

configurations: two configurations are equivalent if they return the same results (for the task execution). If such equivalences can be found, then this would be another method of identifying, and then avoiding options that repeat the processing of previously explored configurations.

3.2. Customization for concept location in software

We will revisit the example from Section 1.3 and explain how we can instantiate the CA for this SE task (i.e., concept location in software using text retrieval and relevance feedback). In our previous work we investigated the use of relevance feedback during text based static concept location in software [30]. Our work was akin to other research in the field, in as much as it focused on proof-of-concept. We realized that it is impossible to investigate the millions of possible configurations described in Section 1.3 so we decided on the following configuration (based on our previous experience and existing literature):

- Use Lucene [52] as the indexing engine with its standard parameters. Lucene is a text retrieval technique based on VSM with a specific ranking mechanism.
- Use the Rocchio equation [111] for the query reformulation using the user provided relevance feedback. This is a common choice in information retrieval applications outside software engineering, when using a VSM model for indexing.
- Use method level granularity for the corpus creation (i.e., each document is a method).
- Separate identifiers and keep the original form as well.
- Use the Porter stemmer [94].
- Do not cluster the data or the results.

With this configuration, concept location is performed as follows. The software system that is about to be changed is converted into a corpus and indexed with Lucene. Each method in the software will have a corresponding document in the corpus, which has in turn a vector representation, created by Lucene. Developers receive a change request (e.g., a bug report or a new feature request) and use it as a query to retrieve methods in the source code that will be modified in order to implement the change request. After each query developers investigate the top n results and let the system know which ones of them are relevant which are not to the change task at hand. The system then automatically reformulates the query (using the Rocchio equation) and retrieves a new set of methods, until the right one is found.

With all the parameters related to the corpus creation and indexing set, we are left with four parameters used by the Rocchio equation. These are: the weight of the terms in the original query (a), the weight of the terms in documents marked as relevant (b), the weight assigned to the terms in documents marked as irrelevant (c), and the number of documents marked as relevant/irrelevant in each round (n). So, in this case, the *configuration* we need to find is a set of values for $\{a, b, c, n\}$ that result in fastest concept location. For each of these parameters, we considered the following possible values: $a \in \{0.25, 0.5, 0.75, 1\}$; $b \in \{0, 0.25, 0.5, 0.75, 1\}$; $c \in \{0, 0.25, 0.5, 0.75, 1\}$; and $n \in \{1, 2, 3, 4, 5\}$. These values determine the number of possible configurations: 4*5*5*5 = 500. The Rocchio algorithm has additional parameters, which we did not vary, such as: the ordering scheme used to rank the terms in the relevant and irrelevant methods (i.e., there are at least 6 different ordering schemes), the number of terms from the relevant and irrelevant methods, which will be added or deleted from the query (we used 10), etc.

Even in this simplified setting, determining the best configuration with the four parameters turned out to be a daunting task. We used a standard AI heuristic search method, that is, Hart's ASTAR algorithm [34]. We tried this approach on data from four different software systems, and we used 16 actual bug reports as change requests (four for each system), for which we had the patches (i.e., we knew what methods need to be changed to fix the bugs). For each change request we tried the 500 possible configurations for $\{a, b, c, n\}$. For each configuration, we asked ASTAR to find the optimum set of "tags" for each method (where the "tags" are one of "relevant or irrelevant"), which retrieve the changed methods faster. Basically, ASTAR was simulating a human making these choices.

With this setup it took the algorithm 14 days to run (about 20 hours for each change request per system) on a standard desktop with Ubuntu Linux, 4GB of RAM, and a 3GHz processor. Not much of that CPU time was consumed by the ASTAR algorithm itself; rather, it was spent on lower-level functions running queries over the thousands of methods in each system. One disturbing result from the

above study was that each system returned a different "best" configuration, indicating clearly that the above CPU intensive process may have to be repeated whenever studying a new data set. Our experience revealed that the gains will come from reducing the number of configurations we need to evaluate.

How will the CA work in this situation? The task (*T*) is concept location. The data (*D*) is composed of four sets, one for each software system analyzed. Each such set has two components: a corpus, obtained from the source code of the system by converting each method into a document and the four bug descriptions (i.e., change requests). Each document in the corpus, including the bug descriptions, has an associated vector used by Lucene. The algorithms employed (*A*) are Lucene and Rocchio. The set of parameters (*P*) that we customize is {*a, b, c, n*} – i.e., the Rocchio parameters. For each bug, we formulate a query that also gets an associated vector generated by Lucene. The CA will need to record the results for each execution of the task (i.e., locate a method to be changed for one bug). The results (*R*) in this case are made of the *n* methods that the user inspects after each query and the relevance feedback it provides to the tool (i.e., relevant or not). The stored data also includes the subsequent queries, generated by Rocchio, with their top *n* results and subsequent feedback tags. Precision is the key measure for concept location performance. A concept location technique (or configuration of it) is better than another one when the user inspects fewer methods to find the one that needs to change. We use the number of inspected methods as the performance score (*S*). All the above represent the data stored by the CA, represented as vectors in the VSM. Do not confuse the CA's VSM representation with the one Lucene uses – they are in fact used together – the Lucene vectors are part of the larger ones that the CA creates.

The CA will replace the ASTAR based approach we used in the past. Its goal will be to provide the user with the best configuration {*a, b, c, n*} to use for a particular system. Clearly the performance of the CA will be impacted by the amount of past data stored and we expect to get better as more of it is available. The CA will run automatically and will exercise possible configurations. In some cases, it will need to run the queries, select all possible combinations of relevance feedback values and estimate how far the target method is. The gain will be when the CA has enough information not to run a query, but instead to extrapolate the result from the stored data in its cache, which should be faster.

In other variants of concept location, which we will investigate under this proposal (i.e., using dependencies and execution traces in addition to the textual data), the use of the CA will be similar. The data in the vector space will be of course different for various approaches, though a large part of each vector will be the same for all concept location application that use at least one of the same DM algorithms. We expect larger gains (when extrapolating) in some cases, as for example, an execution usually takes more effort in more complex techniques for concept location (i.e., it includes at least an execution trace and dependency analysis).

3.3. Customization for defect prediction and effort estimation

Due to space restrictions we decided to give a detailed example only for one of the three tasks. Methodologically, the CA will be used the same way for our other two target SE tasks. The main differences are in the details. The data used here is different hence the compositions of the vectors stored in the cache will differ. Nonetheless, the same model can be used and the same similarity measures and clustering methods. The advantage of vector space models is that they are not sensitive to the type of data they represent. When we instantiate the CA from concept location to defect prediction, we just have to specify what data is stored in the vectors. For example, when dealing with a defect predictor, the data stored in the vectors for each software component (e.g., method, class, etc.) will be composed of metrics used by the predictor (e.g., coupling, cohesion, complexity, etc.), rather than their textual representation as in the previous example. Performance data will be in form of precision and recall of the predictions. Some predictors use a variable number of software quality attributes to build their model; in these cases we can consider them as part of the configuration.

We also need to change the data recording mechanism. For example, for concept location we need to store developer actions, whereas for defect prediction only code attributes. An advantage is that these two tasks have less human involvement in the task execution (i.e., the user does not need to provide feedback at each step). In consequence, we expect to be able to record more data and faster in the cache. The CA

should result in improvements sooner than in the case of concept location. We will discuss this issue in more details in the evaluation part (Section 3.5).

3.4. Optimization of the customization agent

The core requirement for the CA is that the above processing takes less time than exercising all configurations. For this to work, the key feature of the CA is fast incremental clustering of large vector spaces. Unless we can cluster efficiently, the CA will not scale.

It is important to note that the vectors that the CA must cluster can be very wide (as described in the previous sections). By the time we consider all the options required, e.g., selecting and configuring pre-processors, feature selector, discretizers, classifiers, etc., we may be looking at hundreds of options. Worse, when the CA is handling task execution information, we may be looking at vectors tens of thousands wide (for example, in IR based concept location the vector would span over the terms found in all the methods of the software). We already have experience with clustering large data sets of these types of operators on vector space models artifacts [41, 61, 70, 76, 104, 117]. However, the existing tools require very complex and costly computations that need to be reiterated every time the data changes. We need to avoid these issues.

Tools that can reason about these structures have some known limitations. For example, many local learning schemes have slow runtimes. Frank et al's locally weighted Bayes classifier weights all data by its distance to some central point [28]. It performs better than standard Bayes, but runs much slower since the neighborhood of each test case is computed at test time. In the worse case (when the test set is the entire set of vectors V) this search takes time $O(|V|^2)$ since the distance between each pair of documents must be computed. While theoretically tractable, $|V|$ can be impractically large. Our experience with hierarchical distance trees [4-5], rotation methods [64], and approximate distance measures [3] has not been encouraging either.

Hence, we propose to use a clustering technique inspired by research in software testing [116]. *Bunching* groups V training documents into B "*bunches*", where B is much less than $|V|$. A full search is required within each bunch to find an exact match, but if each bunch stores (for example) 1% of the data, then any $O(|V|^2)$ search algorithm is reduced to $(1/100)^2$; i.e., ten thousand times less than $O(|V|^2)$. The CA will search on the bunches for the best configurations.

We plan to create bunches using two algorithms: the IPO *pairwise generator* [116] and the PID *discretizer* [29]. IPO is a 2-factor combinatorial test vector generator. Given vectors with cells ($d_i w_j$) holding the number of times term w_j appears in document d_i, (i.e., typical values in a VSM, where documents span a space determined by their terms), the PID discretizer converts the values seen for term w into a small number of bins B. We call the number of bins per term the "cardinality" of that term. In our example from Section 3.2, we had four Rocchio parameters {*a, b, c, n*}, each with a few possible values: four values for *a*, and five for each of the other three parameters. This leads to 500 different configurations to try – IPO will consider each configuration as a vector. IPO would generate only 30 configurations to test, which represent 30 vectors uniformly spaced across all the 500 possible vectors (the same cannot be said for any random generator). Each of the IPO vectors is a suitable candidate for creating a bunch.

Bunching should run orders of magnitude faster than (a) hierarchical distance trees; (b) rotation methods; or (c) Athitsos's approximate distance measures, for two reasons. First, B is smaller than $|V|$ and second, most of the computational cost of bunching can be performed off-line *before any data arrives*:

- Pre-determining the cardinality of the terms, we can compute the bunches even before seeing any data (given the right discretization policies, it is possible to make such a pre-determination – see bellow).
- Once the bunches are computed, we can also build an index into the bunches. For example, we can run a cover tree algorithm on the bunches to generate a hierarchical distance tree. Note that the generation of this index will be much faster than building a cover tree on the original data (since we are processing only B bunches and not $|V|$ data items). As with bunch generation, the tree could be computed before any data arrives.

- Once the bunches are created and indexed, then loading V_1 training documents into the bunches and processing V_2 test documents is a four-stage process:
  1. Map each member of the training data V_1 to its nearest bunches; i.e., O(V_1*log(B));
  2. Train one learner from the training data at each bunch;
  3. Map each member of the test data V_2 to its nearest bunch; i.e., O(V_2*log(B));
  4. Apply the learner at that bunch.

Depending on the choice of learner, the above process will take at most linear time. For example, stage (4) will take time O(A*C) for Bayes classifiers where A is the number of terms/attributes and C is the number of classes (not in OO sense but classification). Stage (2) may be slower for non-Bayesian learning schemes. For example, while the Bayesian approach takes linear time, applying some rotation algorithm during stage (2) can take polynomial time; e.g., PCA takes time O(B*A^2) [24]. But even this polynomial time rotation will be faster over the B bunches than $|V|$ vectors since IPO will ensure B is smaller than $|V|$.

The bunching optimization assumes that it is possible *to accurately and quickly pre-determine the cardinality of each term*. This is an issue that must be carefully assessed. Clearly, bunching could suffer from the same drawbacks as Athitsos' method; i.e., the fewer the bunches, the more approximate (and hence, the more erroneous) the reasoning. One of the goals of this project is understanding the trade-offs between number of bunches, runtimes, and reasoning quality. We will use Gama & Pinto's PID incremental discretizer to determine this cardinality [29]. PID maintains two arrays called *lower* and *upper*. The lower array is a standard discretizer with breakpoints b_i and frequency counts c_i. The upper array is an equal width discretizer used as a temporary database summarizing the input data. The upper array is much longer than the lower one; e.g., if the goal is to return 10 bins, then PID recommends the larger array be of size 100 to 200. New data increments the frequency counts in the upper array and, if those counts grow too large, PID splits that bin in two new bins, each with half the frequency of the old bin. Occasionally, the contents of the upper array are summarized into a new set of breakpoints b_i and c_i in the lower array. Yang and Webb [121] report that this very simple scheme is nearly as effective as numerous other schemes and runs seven to ten times faster, because (a) it is very simple to implement and (b) when updating the lower array, PID only has to study the frequency counts of the upper array (and not all the original data).

Any cached computation of bunches must be deleted and recreated if a term's bins change (i.e., if after recreating PID's lower array, the new array is different to the old one). This *rebunching* event is an expensive process since it can require a new run of IPO, cover trees, and a complete pass through all historical data. An open issue with this research is the frequency of rebunching. In limited studies on small data sets, we find rebunching is a rare event [7] but this needs to be confirmed on larger samples.

It is insightful to compare bunching with the obvious alternative method; i.e., cluster the data and initialize the bunches with the cluster centroids. However, bunching is very different than clustering:

- Clustering works by examining all pairs of points, multiple times (e.g., the k-means clustering algorithm computes the distance from all data to their nearest centroid, adjusting that centroid as the algorithm progresses).
- Clustering runs over all the data and any new data item can (potentially) alter the clusters;

On the other hand, bunching:

- Partitions the data into a small, stochastically selected, initial set (for the initial discretization), followed by a scan of the remaining data.
- After examining the small stochastically sample initial set, the bunches are fixed. Hence, for most of the reasoning, there is no computation cost associated with reflecting on how new data affects old bunches (exception: rebunching, which is likely not to be a common case).

3.5. Evaluation plan

The goal of our research is to improve the customization of data mining techniques that directly support specific software engineering tasks. We will focus our evaluation plan on assessing the effect of using the CA for three important SE tasks: *concept location* in source code during software change, *defect*

prediction, and software *development effort estimation*. Both PIs have extensive experience in addressing them (see Section 2 for details) and have access to a significant amount of past data from empirical evaluations of solutions to these tasks. Some of these are available in the PROMISE repository [1] (one of the PIs, Tim Menzies is the curator of this repository). Menzies is a highly regarded and referenced expert in defect prediction [70, 75, 85-86, 89-90] and effort estimation [66, 71, 74, 77, 79, 88]. Much of Marcus' previous work was devoted to the use of text mining methods, combined with dynamic and static analysis of the code, to support concept location [30, 49, 63, 96-97, 99, 102-103, 105, 107]. He also used text mining techniques to define metrics that were used for defect prediction [61]. These three tasks have characteristics that require somewhat different approaches in the evaluation. At the same time though, they also have enough common features that allow us to employ a general evaluation model for all three.

Once the CA framework is defined and implemented, we will need to instantiate it for each of the three SE tasks. Remember that each task has various solutions. We will investigate several solutions for each task. The evaluation plan for each task consists of two stages.

In the **first stage** we will employ replication and reenactment [39] of previous empirical studies, which were initially performed by us (and others) without the use of customization. Reenactment is somewhat different than replication, as it allows avoiding certain validity threats in the process, which is often needed when there are humans in the loop. It also allows for much faster collection of the data. For example, a replication of a concept location task would imply a user to find the methods relevant to a change request and eventually implement the change to verify the results. The problem here is that a change may be done in a variety of ways by different people and it also may take a long time. With reenactment, on the other hand, we can repeat the concept location part without the need for the complete implementation. A necessary condition for reenactment is the availability of previous data. In our example, we would need to have past change requests and the actual changes in the code that resulted (e.g., a bug description and its patch). Then we reenact concept location starting from the same change request. When we stop, if the located method is present in the previous change set (or patch), then we consider it a success, otherwise a failure. This technique allows us to avoid the humans in the loop and simulate their choices, much like how we did in the example using ASTAR for the concept location task. There we simulated a user by trying all possible tags she would choose in a real setting. In these cases we need to look at the best case scenario (i.e., at each step in the process we assume the human makes the best choice), the worse case scenario, and of course the average. This technique will allow us to assess relatively quickly how much different configurations affect the solutions to our SE tasks and also to see how well the CA chooses the best configurations. We are of course loosing the human factor, but that will not influence the best and worse case scenarios. This issue is remedied in the second stage.

For defect predication and effort estimation, replication of previous experiments is easier and desirable; hence we do not need to resort to reenactment.

In the **second stage** we will evaluate the CA and its effect on the three SE tasks on real-time data collected from academic environment and industry setting. We will deploy the CA with students in our classes and research labs and conduct case studies in concept location, defect prediction, and effort estimation. One of the PIs (Marcus) is teaching a software engineering sequence of classes at Wayne State University (one undergraduate and three graduate classes – CSC 4110, CSC 6110, CSC 7110, and CSC 8110). The undergraduate class and the introductory graduate class (CSC 4110 and CSC 6110) include projects that involve students performing changes to existing open source software. They use tools and techniques we developed previously for concept location. These will be augmented with the CA and the data will be collected. The advanced SE classes deal with effort estimation and defect prediction, among other topics. Usually at least one team of students undertakes a project on these issues every class. We will use the CA in conjunction with these class projects. Students' performance will be recorded and compared with data we have from previous semesters. We will also have groups of students that use the CA and groups that do not, then we will compare their results.

Academic settings are not always good approximations for industrial environments. They allow us to perform multiple studies, with relatively few related data points in each context. What we need in addition are longitudinal studies, performed over longer periods of time, collecting many data points in

the same context. We will collaborate with Mr. Andrey Sergeyev, who is a Senior Software Development Engineer in the Pro/E Core Geometry Group at Parametric Technology Corporation (PTC) in Boston, MA (see the attached commitment letter). He is a former student of one of the PIs (Marcus) and together they pioneered the use of text mining for concept location [63]. His development group is routinely using some of the DM based techniques for concept location. Under this project, Sergeyev will deploy the CA in his development group and monitor its use. We plan to collect data for about 18 months and then assess the effect of the CA compared with previous data. We will also see how the efficiency of CA will evolve given more collected data.

In all studies we will monitor the performance measures. For concept location, we mainly will monitor the number of methods developers investigate before deciding on a change (i.e., this can be seen as an effort measure). For defect prediction, the standard precision and recall values with respect to the set of predicted modules. For effort estimation, we will analyze the accuracy of the estimates.

As mentioned before, the three tasks present different challenges. For defect predictors, we expect a far larger set of possible configurations than for effort estimation. More code attributes and DM algorithms are used for defect prediction than for effort estimation. This will affect the performance of the CA. On the other hand, effort estimation will result in smaller configuration space and will be best to start the evaluation on. Finally, concept location is the task that has the human deeply involved in the loop. The challenge in this case is collecting data for the CA cache during development. The solution we propose is based on our previous work on developer monitoring [20]. We developed an Eclipse plug-in that records how developers search the source code (i.e., what queries they run, what files they access, what views they use, etc.). It is an unobtrusive mechanism, which we will adapt for our needs (we need in fact less information here) and also implement a version that works with MS Visual Studio. They are the main development environments used by our students and industry collaborators.

3.6. Potential pitfalls and alternatives

While we expect that the CA will bring improvement in addressing the three SE tasks we focus on in this proposal, we have to wait for the empirical data to see how much and at what cost. Assuming we design and implement all technical aspects correctly, the data may still prove us wrong in the end. That is, it may take the CA too long to select the best configurations to be of any practical use, or the effect of various configurations on the task performance is negligible. Our work so far (see Section 3.2) indicates that, at least in the case of concept location using relevance feedback, the values of the parameters have a strong influence on the results. We expect that to hold for all tasks and instances we will investigate. It may turn out that the CA will not be very effective compared to simpler approaches. This will be an indication that there is little we (or machines) can learn from past SE data that is usable for new actions, i.e., there will be no "smooth" regions of the data where we can make predictions or there is little connection between the regions with higher variance of the data. Such a finding will not help the three SE tasks, but if proven correct, it could be quite an astonishing result for the future of DM applications in SE. The way we see it, this is a win-win situation. We either improve the three tasks (and possibly others), or we learn something new and valuable that will impact the field drastically and open new research directions.

3.7. Research plan

The project is planned for three years:

1. The main effort during the first year will be devoted to the detailed design and implementation of the CA framework and its instantiations for the three SE tasks (with several solutions each). The major deliverables will be the specific CAs (i.e., software tools). This will include the monitoring tools we discussed, which will be part of the CA for concept location. Software testing will be the primary evaluation mechanism. Meanwhile we will design in details the empirical studies we plan to carry out and select the data we will use.
2. The first part of the second year will be devoted to the first stage of the evaluation plan, based on reenactments. The results will allow us to update the CA framework as needed. More

development effort will be involved. We will finalize the design of the evaluation studies and we will deploy the CA with our industry partners and also start the studies with our students.

3. The first part of the third year we will finalize the collection of the empirical data and analyze them. The analysis will indicate whether we need any additional changes to the CAs. If not, we will define a generalized CA framework, which can be used with other applications of DM in SE.

3.8. Integration of research and education

This proposal builds on the research strengths of the PIs and provides a great opportunity to enhance our teaching efforts and the SE curriculum. In particular, Menzies is regularly teaching a set of data mining courses at West Virginia University (WVU): CS473 - Data Mining and CS573 - Advanced Data Mining. He plans to include applications of DM algorithm to SE problems into the course. In parallel, Marcus will teach a graduate course at Wayne State University (WSU), CSC 6140 – Knowledge Based Software Engineering. This course has not been taught in the Department in more than 6 years and he will revise it to focus on DM applications in SE. We expect that the students in these classes will acquire very important knowledge and practice in a field that is moving from pioneering into mainstream. The current SE curriculum does not offer any DM training to students and we hope that will change in the future. Our efforts will be a first step in that direction.

4. Broader Impact

Our proposal focuses on research and educational activities that target the software engineering research community, the software engineering practitioners, the data mining researchers, and computer science students in general. The research objectives will bring together work from different communities such as software engineering and data mining.

4.1. Broadening the participation of under-represented groups

Due to the specific location of Wayne State University (WSU), we have unique educational responsibilities to initiate outreach programs to traditional underrepresented minority students in the Detroit Metropolitan Area. We annually invite high school students from underrepresented groups to visit our research laboratory. We will also encourage undergraduate students to participate in testing the tools produced by our laboratory with the goal to attract them to pursue graduate education and engage them in research.

West Virginia University (WVU) has a successful record of including minority students into undergraduate and graduate research. For example, research labs at WVU continue to maintain close to 50% participation of women/minority graduate research assistants.

Women students will be encouraged and supported to apply to the annual CRA-W Distributed Mentor Project (CRA-W DMP) and the CDC Distributed Mentor Project (CDC DMP) in order to obtain research experience and mentoring outside of WSU and WVU. The investigators' current and future women graduate advisees will be encouraged and supported to attend CRA-W career mentoring workshops and to network with senior women in the field. Dr. Marcus has a history of supporting the involvement of underrepresented categories of students, as one of the former M.S. graduate students (Ms. Denise Comorski), a current M.S. student (Ms. Nariman Ammar), and a current Ph.D. student (Ms. Sonia Haiduc) are all women. For example, Ms. Haiduc participated during March 30-31, 2007 in the Michigan Celebration of Women in Computing Workshop, held at the Kellogg Biological Station in Hickory Corners, MI and to the CRA-W Graduate Cohort in San Mateo, CA, in March 2009.

Two of the faculty in the Computer Science Department at WSU, Dr. Monica Brokmeyer and Dr. Farshad Fotouhi, have current NSF funding to address the severe under-representation of minorities, women, and first-generation college students in the computing and information technology. They formed an alliance between the Department of Computer Science at WSU and Focus: HOPE in Detroit, Michigan to implement the Information Management and Systems Engineering (IMSE) program, which is designed to guide and support disadvantaged students at critical junctures from a GED through the completion of a

post-secondary degree. Rather than duplicating their efforts, we will work closely with our colleagues to recruit suitable students from the targeted group to be involved in the proposed research.

4.2. Impact on the research community and industry

The three SE tasks we target to improve are frequent problems addressed in industry. Improving them should have a significant impact in the software engineering practice. As highlighted in Sections 1.3 and 2, there are many more applications of DM in SE than the three we address here. We expect that the results of this research will apply with little effort to many of those applications. We will make our findings, tools, and results available to the research community. Our comprehensive evaluation plan will make sure that the research is relevant to both academia and industry. We expect that by improving the DM algorithm customization problem, many existing applications will make a significant step towards industry adoption.

5. Results from Prior NSF Support

This proposal builds on the experience of the PIs, who started collaborating on related work in the past two years [76]. Each investigator has previous and current funding from NSF, which generated research results published in many papers and it supported the training of graduate students.

Dr. Menzies was awarded in July 2008 an NSF grant (CCF-0810879) on "Automatic Quality Assessment: Exploiting Knowledge of the Business Case", together with Dr. Bojan Cukic (Co-PI) from WVU. The task of this research is to see if adding humans into the inner loop of software data miners improves the learning of defect predictors. This work already resulted in several paper submissions currently under review as well as published results [41, 117]. The research focuses on adding business knowledge to the evaluation criteria of the learning. Through the analysis of sixteen projects from public repositories, we observed that software quality does not necessarily benefit from the prediction of fault prone components. The inclusion of misclassification cost in model evaluation may indicate that even the "best" models achieve performance no better than trivial classification. In addition, we found that the selection of examples for use in data mining can be tuned to local relevancy criteria.

Dr. Marcus was funded as Co-PI under the NSF grant CCF-0438970 (April 2005 - March 2009), to investigate the "Design of Incremental Change". The research is a collaboration with faculty from WSU, Dr. Vaclav Rajlich (PI) and Dr. Sorin Draghici (Co-PI). The objective of the research was to study and define the activities of Incremental Change (IC) design, with an emphasis on the aspects of the IC design that are different from the classical design of completely new software. Specifically, the project focused on the investigation and automation in part of concept location in source code, with emphasis on the search of program dependencies, analysis of execution traces, and the use of IR techniques. The results were published in several papers, theses, and dissertations [20-23, 33, 49, 57, 59-62, 76, 95-107, 119-120], two of which [96, 99] received the *Best Paper Award* at the 14th and 15th IEEE Conference on Program Comprehension (ICPC) in 2006 and 2007, respectively.

More recently (July 2008), Dr. Marcus was awarded a new NSF grant CCF-0820133 on "Supporting software evolution by the combined analysis of textual and structural information". The research is in collaboration with Dr. Vaclav Rajlich (Co-PI) from WSU. The goal of the project is to improve classical structured static analysis of software with the use of IR to support change impact analysis in Object-Oriented software. The results of the work so far were published in [2, 30, 76, 93, 104] and the main finding is that the conceptual coupling between classes, defined using textual information in the source code, is an excellent change predictor when combined with existing structural coupling metrics [61].

Finally, Dr. Marcus was awarded in August 2009 the NSF CAREER Award (CCF- 0845706) for the "Management of Unstructured Information During Software Evolution". The research will define and evaluate an infrastructure for the management of the textual information present in software systems, which will be used to define a conceptual model of an existing software system that will complement the traditional structural, behavioral, and architectural models. Early results on assessing and detecting the quality of the software lexicon were published in [2], as well improvements to existing static concept location techniques [30].

References Cited

[1] "Promise Repository", Online at http://promisedata.org/.

[2] Abebe, S. L., Haiduc, S., Tonella, P., and Marcus, A., "Lexicon Bad Smells in Software", in *Proceedings 16th IEEE Working Conference on Reverse Engineering (WCRE2009)*, 2009, pp. 95-99.

[3] Athitsos, V., Alon, J., and Sclaroff, S., "Efficient Nearest Neighbor Classification Using a Cascade of Approximate Similarity Measures", in *Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005, pp. 486-493.

[4] Bentley, J. L., "K-d Trees for Semidynamic Point Sets", in *Proceedings 6th Annual Symposium on Computational Geometry*, Berkley, California, United States, 1990, pp. 187-197.

[5] Beygelzimer, A., Kakade, S., and Langford, J., "Cover Trees for Nearest Neighbor", in *Proceedings 23rd International Conference on Machine Learning*, Pittsburgh, Pennsylvania, 2006, pp. 97-104.

[6] Boehm, B., Clark, B., Horowitz, E., Brown, A. W., Reifer, D., Chulani, S., Madachy, R., and Steece, B., *Software Cost Estimation with Cocomo II*, Prentice Hall, 2000.

[7] Boland, D. J., *Data Discretization Simplified: Randomized Binary Search Trees for Data Preprocessing* West Virginia University, Morgantown, West Virginia, MS Thesis, 2007.

[8] Bouktif, S., Sahraoui, H., and Antoniol, G., "Simulated annealing for improving software quality prediction", in *Proceedings 8th Annual Conference on Genetic and Evolutionary Computation*, 2006, pp. 1893-1900.

[9] Chen, Z., Menzies, T., Port, D., and Boehm, B., "Feature subset selection can improve software cost estimation", in *Proceedings International Workshop on Predictor Models in Software Engineering (PROMISE)*, 2005, pp. 1-6.

[10] Chen, Z., Menzies, T., Port, D., and Boehm, B., "Finding the right data for software cost modeling", *IEEE Software*, 22, 6, 2005, pp. 38 - 46

[11] Cheng, H., Lo, D., Zhou, Y., Wang, X., and Yan, X., "Identifying Bug Signatures Using Discriminative Graph Mining", in *Proceedings International Symposium on Software Testing and Analysis (ISSTA)*, 2009, pp. 141-151.

[12] Chiang, E. and Menzies, T., "Position paper: Summary of simulations for very early lifecycle quality evaluations", in *Proceedings Workshop on Software Process Simulation Modeling (PROSIM)*, 2003

[13] Chiang, E. and Menzies, T., "Simulations for very early lifecycle quality evaluations", *Software Process: Improvement and Practice* 7, 3-4, 2003, pp. 141 - 159.

[14] Cohn, D., Atlas, L., and Ladner, R., "Improving generalization with active learning", *Machine Learning*, 15, 2, 1994, pp. 201-221.

[15] Connell, M. and Menzies, T. J., "Quality metrics: Test coverage analysis for smalltalk.", in *Proceedings Tools Pacific*, 1996.

[16] Cornford, S. L., Feather, M. S., Dunphy, J. R., Salcedo, J., and Menzies, T., "Optimizing spacecraft design optimization engine development: Progress and plans", in *Proceedings IEEE Aerospace Conference*, 2003, pp. 3681- 3690.

[17] Dasgupta, S. and Hsu, D., "Hierarchical sampling for active learning", in *Proceedings International Conference on Machine Learning (ICML)*, 2008, pp. 208-215.

[18] De Lucia, A., Fasano, F., Oliveto, R., and Tortora, G., "Recovering Traceability Links in Software Artefact Management Systems", *ACM Transactions on Software Engineering and Methodology*, 16, 4, 2007.

[19] Dekhtyar, A., Hayes, J. H., and Menzies, T., "Text is software too", in *Proceedings International Workshop on Mining Software Repositories (MSR)*, 2004, pp. 22-26.

[20] Dit, B., *Monitoring the Searching and Browsing Behavior of Developers in Eclipse during Concept Location*, Wayne State University, Detroit, M.S. Thesis, 2009.

[21] Dit, B. and Marcus, A., "Improving the Readability of Defect Reports", in *Proceedings International Workshop on Recommendation Systems for Software Engineering (RSSE'08)*, 2008, pp. 47-49.

[22] Dit, B., Poshyvanyk, D., and Marcus, A., "Measuring the Semantic Similarity of Comments in Bug Reports", in *Proceedings 1st International Workshop on Semantic Technologies in System Maintenance (STSM'08)*, 2008.

[23] Dong, Y., *IRiSS and JIRiSS - Interactive searching tools for concept location in software systems*, Wayne State University, Detroit, M.S. Thesis, 2007.

[24] Du, Q. and Fowler, J. E., "Low-Complexity Principal Component Analysis for Hyperspectral Image Compression", *International Journal of High Performance Computing Applications*, 22, 4, November 2008, pp. 438-448.

[25] El-Ramly, M., Stroulia, E., and Sorenson, P. G., "From Run-Time Behavior to Usage Scenarios: An Interaction-Pattern Mining Approach", in *Proceedings Eighth ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2002, pp. 315-324.

[26] Feather, M. S. and Menzies, T., "Converging on the optimal attainment of requirements", in *Proceedings IEEE Joint Conference On Requirements Engineering ICRE'02 and RE'02*, 2002, pp. 263-270.

[27] Fenton, N., Neil, M., Marsh, W., Hearty, P., Radliński, K., and Krause, P., "On the effectiveness of early life cycle defect prediction with Bayesian Nets", *Empirical Software Engineering*, 13, 5, 2008, pp. 499-537.

[28] Frank, E., Hall, M., and Pfahringer, B., "Locally Weighted Naive Bayes", in *Proceedings Conference on Uncertainty in Artificial Intelligence*, 2003, pp. 249-256.

[29] Gama, J. and Pinto, C., "Discretization from Data Streams: Applications to Histograms and Data Mining", in *Proceedings ACM Symposium on Applied Computing*, Dijon, France, 2006, pp. 662-667.

[30] Gay, G., Haiduc, S., Marcus, A., and Menzies, T., "On the Use of Relevance Feedback in IR-Based Concept Location", in *Proceedings IEEE International Conference on Software Maintenance (ICSM)*, 2009, pp. 351-360.

[31] Geletko, D. and Menzies, T., "Model-based software testing via incremental treatment learning", in *Proceedings 28th Annual NASA Goddard Software Engineering Workshop (SEW)*, 2003, pp. 82-90.

[32] Gold, N., Harman, M., Li, Z., and Mahdavi, K., "Allowing Overlapping Boundaries in Source Code using a Search Based Approach to Concept Binding", in *Proceedings 22nd IEEE International Conference on Software Maintenance (ICSM'06)*, 2006, pp. 310-319.

[33] Haiduc, S. and Marcus, A., "On the Use of Domain Terms in Source Code", in *Proceedings of the 16th IEEE International Conference on Program Comprehension (ICPC'08)*, 2008, pp. 113-122.

[34] Hart, P. E., Nilsson, N. J., and Raphael, B., "A formal basis for the heuristic determination of minimum cost paths", *IEEE Transactions on Systems Science and Cybernetics*, 4, 2, 1968, pp. 100-107.

[35] Hayes, J. H., Dekhtyar, A., and Sundaram, S. K., "Advancing candidate link generation for requirements tracing: the study of methods", *IEEE Transactions On Software Engineering*, 32, 1, 2006, pp. 4-19.

[36] Haynes, P. and Menzies, T., "C++ is Better than Smalltalk?", in *Proceedings Tools Pacific*, 1993, pp. 75-82.

[37] Haynes, P. and Menzies, T., "The Effects of Class Coupling on Class Size in Smalltalk Systems", in *Proceedings Tools*, 1994, pp. 121–129.

[38] Haynes, P., Menzies, T., and Phipps, G., "Using the size of classes and methods as the basis for early effort prediction; empirical observations, initial application; a practitioners experience report", in *Proceedings OOPSLA Workshop on OO Process and Metrics for Effort Estimation*, 1995, pp. 148.

[39] Jensen, C. and Scacchi, W., "Discovering, Modeling, and Reenacting Open Source Software Development Processes", *New Trends in Software Process Modeling Series in Software Engineering and Knowledge Engineering*, 18, 2006, pp. 1-20.

[40] Jiang, Y., Cukic, B., and Menzies, T., "Fault prediction using early lifecycle data", in *Proceedings 18th IEEE International Symposium on Software Reliability (ISSRE)*, 2007, pp. 237–246.

[41] Jiang, Y., Cukic, B., and Menzies, T., "Can Data Transformation Help in the Detection of Fault-Prone Modules?", in *Proceedings 2008 Workshop on Defects in Large Software Systems*, Seattle, Washington, 2008, pp. 16-20.

[42] Kothari, J., Denton, T., Shokoufandeh, A., and Mancoridis, S., "Reducing Program Comprehension Effort in Evolving Software by Recognizing Feature Implementation Convergence", in *Proceedings 15th International Conference on Program Comprehension (ICPC)*, 2007, pp. 17-26.

[43] Kuhn, A. and Greevy, O., "Exploiting the Analogy between Traces and Signal Processing", in *Proceedings 22nd International Conference on Software Maintenance(ICSM)*, 2006, pp. 320-329.

[44] Kuhn, A., Greevy, O., and Girba, T., "Applying Semantic Analysis to Feature Execution Traces", in *Proceedings IEEE Workshop on Program Comprehension through Dynamic Analysis (PCODA 2005)*, 2005, pp. 48-53.

[45] Lessmann, S., Baesens, B., Mues, C., and Pietsch, S., "Benchmarking classification models for software defect prediction: A proposed framework and novel findings", *IEEE Transactions on Software Engineering*, 34, 4, 2008, pp. 485-496.

[46] Leveson, N., *Safeware System Safety And Computers*, Addison-Wesley, 1995.

[47] Linstead, E., Cristina Lopes, and Baldi, P., "An Application of Latent Dirichlet Allocation to Analyzing Software Evolution", in *Proceedings Seventh International Conference on Machine Learning and Applications*, 2008, pp. 813-818.

[48] Linstead, E., Rigor, P., Bajracharya, S., Lopes, C., and Baldi, P., "Mining concepts from code with probabilistic topic models", in *Proceedings 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07)*, 2007, pp. 461-464.

[49] Liu, D., Marcus, A., Poshyvanyk, D., and Rajlich, V., "Feature Location via Information Retrieval based Filtering of a Single Scenario Execution Trace", in *Proceedings 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07)*, 2007, pp. 234-243.

[50] Liu, Y., Menzies, T., and Cukic, B., "Data sniffing - monitoring of machine learning for online adaptive systems", in *Proceedings IEEE Tools with Artificial Intelligence(ICTAI)*, 2002, pp. 16- 21.

[51] Lo, D., Cheng, H., Han, J., Khoo, S., and Sun, C., "Classification of Software Behaviors for Failure Detection: A Discriminative Pattern Mining Approach", in *Proceedings 15th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2009.

[52] Lucene, Online at http://lucene.apache.org/.

[53] Lukins, S. K., Kraft, N. A., and Etzkorn, L. H., "Source Code Retrieval for Bug Localization Using Latent Dirichlet Allocation", in *Proceedings 15th Working Conference on Reverse Engineering*, 2008, pp. 155-164.

[54] Lum, K., Hihn, J., and Menzies, T., "Studies in software cost model behavior: Do we really understand cost model performance?", in *Proceedings International Conference of the International Society of Parametric Analysts (ISPA)*, 2006.

[55] Lutz, R. and Mikulski, C., "Operational anomalies as a cause of safety-critical requirements evolution", *Journal of Systems and Software*, 65, 2, 2003, pp. 155-161.

[56] Maletic, J., Munson, E., Marcus, A., and Nguyen, T., "Combining Traceability Link Recovery with Conformance Analysis via a Formal Hypertext Model", in *Proceedings 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'03)*, 2003, pp. 47-54.

[57] Marcus, A., Comorski, D., and Sergeyev, A., "Supporting the Evolution of a Software Visualization Tool through Usability Studies", in *Proceedings 13th IEEE International Workshop on Program Comprehension (IWPC'05)*, 2005, pp. 307-316.

[58] Marcus, A. and Maletic, J., "Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing", in *Proceedings 25th IEEE/ACM International Conference on Software Engineering (ICSE'03)*, 2003, pp. 125-137.

[59] Marcus, A., Maletic, J. I., and Sergeyev, A., "Recovery of Traceability Links Between Software Documentation and Source Code", *International Journal of Software Engineering and Knowledge Engineering*, 15, 4, 2005, pp. 811-836.

[60] Marcus, A. and Poshyvanyk, D., "The Conceptual Cohesion of Classes", in *Proceedings 21st IEEE International Conference on Software Maintenance (ICSM'05)*, 2005, pp. 133-142.

[61] Marcus, A., Poshyvanyk, D., and Ferenc, R., "Using the Conceptual Cohesion of Classes for Fault Prediction in Object Oriented Systems", *IEEE Transactions On Software Engineering*, 34, 2, 2008, pp. 287-300.

[62] Marcus, A., Rajlich, V., Buchta, J., Petrenko, M., and Sergeyev, A., "Static Techniques for Concept Location in Object-Oriented Code", in *Proceedings 13th IEEE International Workshop on Program Comprehension (IWPC'05)*, 2005, pp. 33-42.

[63] Marcus, A., Sergeyev, A., Rajlich, V., and Maletic, J., "An Information Retrieval Approach to Concept Location in Source Code", in *Proceedings 11th IEEE Working Conference on Reverse Engineering (WCRE'04)*, 2004, pp. 214-223.

[64] McNames, J., "A Fast Nearest-Neighbor Algorithm Based on a Principal Axis Search Tree", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23, 9, September 2001, pp. 964-976.

[65] Menzies, T., "Practical machine learning for software engineering and knowledge engineering", in *Handbook of Software Engineering and Knowledge Engineering*, World-Scientific, 2001.

[66] Menzies, T., Chen, Z., Hihn, J., and Lum, K., "Selecting best practices for effort estimation", *IEEE Transactions on Software Engineering*, 32, 11, 2006, pp. 883-895.

[67] Menzies, T., Chiang, E., Feather, M., Hu, Y., and Kiper, J. D., "Condensing uncertainty via incremental treatment learning", in *Software Engineering with Computational Intelligence*, Khoshgoftaar, T. M., Ed., 2003.

[68] Menzies, T., Dekhtyar, A., Stefano, J. S. D., and Greenwald, J., "Problems with Precision: A Response to “Comments on ‘Data Mining Static Code Attributes to Learn Defect Predictors’”", *IEEE Transactions on Software Engineering*, 33, 9, 2007, pp. 637-640.

[69] Menzies, T., Elwaras, O., Hihn, J., Feathearnd, B., Boehm, M., and Madachy, R., "The business case for automated software engineerng", in *Proceedings International Conference on Automated Software Engineering (ASE)*, 2007, pp. 303-312.

[70] Menzies, T., Greenwald, J., and Frank, A., "Data Mining Static Code Attributes to Learn Defect Predictors", *IEEE Transactions on Software Engineering*, 33, 1, 2007, pp. 2-13.

[71] Menzies, T. and Hihn, J., "Evidence-based cost estimation for better quality software", *IEEE Software*, 23, 4, 2006, pp. 64-66.

[72] Menzies, T. and Hu, Y., "Data mining for very busy people", *IEEE Computer*, 36, 11, 2003, pp. 22 - 29.

[73] Menzies, T., Kiper, J., and Feather, M., "Improved software engineering decision support through automatic argument reduction tools", in *Proceedings 2nd International Workshop on Software Engineering Decision Support*, 2003.

[74] Menzies, T., Lum, K., and Hihn, J., "The deviance problem in effort estimation", in *Proceedings International Workshop on Predictor Models in Software Engineering (PROMISE)*, 2006.

[75] Menzies, T., Lutz, R., and Mikulski, C., "Better analysis of defect data at NASA", in *Proceedings 15th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2003.

[76] Menzies, T. and Marcus, A., "Automated Severity Assessment of Software Defect Reports", in *Proceedings IEEE International Conference on Software Maintenance (ICSM)*, 2008, pp. 346-355.

[77] Menzies, T., Port, D., Chen, Z., and Hihn, J., "Simple software cost estimation: safe or unsafe?", in *Proceedings International Workshop on Predictor Models in Software Engineering (PROMISE)*, 2005, pp. 1-6.

[78] Menzies, T., Port, D., Chen, Z., Hihn, J., and Stukes, S., "Specialization and extrapolation of induced domain models: Case studies in software effort estimation", in *Proceedings International Conference on Automated Software Engineering (ASE)*, 2005.

[79] Menzies, T., Port, D., Chen, Z., Hihn, J., and Stukes, S., "Validation methods for calibrating software effort models", in *Proceedings 27th International Conference on Software Engineering (ICSE)*, 2005, pp. 587-595.

[80] Menzies, T., Raffo, D., Setamanit, S., Hu, Y., and Tootoonian, S., "Model-based tests of truisms", in *Proceedings 17th International Conference on Automated Software Engineering (ASE)*, 2002, pp. 183.

[81] Menzies, T. and Richardson, J., "Making sense of requirements, sooner", *IEEE Computer*, 39, 10, 2006, pp. 112-114.

[82] Menzies, T., Setamanit, S., and Raffo, D., "Data mining from process models", in *Proceedings Workshop on Software Process Simulation Modeling (PROSIM)*, 2004.

[83] Menzies, T. and Singh, H., "How AI can help SE; or: Randomized search not considered harmful", in *Proceedings 14th Canadian Conference on Artificial Intelligence*, 2001, pp. 100-110.

[84] Menzies, T. and Sinsel, E., "Practical large scale what-if queries: Case studies with software risk assessment", in *Proceedings 15th International Conference on Automated Software Engineering (ASE)*, 2000, pp. 165-173.

[85] Menzies, T. and Stefano, J. S. D., "How good is your blind spot sampling policy?", in *Proceedings 8th IEEE Conference on High Assurance Software Engineering*, 2003, pp. 129-138.

[86] Menzies, T. and Stefano, J. S. D., "More success and failure factors in software reuse.", *IEEE Transactions on Software Engineering*, 29, 5, 2003, pp. 474 - 477

[87] Menzies, T., Stefano, J. S. D., and Chapman, M., "Learning early lifecycle IV&V quality indicators", *9th International Software Metrics Symposium*, 2003, pp. 88- 96.

[88] Menzies, T., Stefano, J. S. D., Cunanan, C., and Chapman, R., "Mining repositories to assist in project planning and resource allocation", in *Proceedings International Workshop on Mining Software Repositories (MSR)*, 2004, pp. 75–79.

[89] Menzies, T., Stefano, J. S. D., Orrego, A., and Chapman, R., "Assessing predictors of software defects", in *Proceedings Workshop on Predictive Software Models*, 2004.

[90] Menzies, T., Turhan, B., Bener, A., Gay, G., Cukic, B., and Jiang, Y., "Implications of Ceiling Effects in Defect Predictors", in *Proceedings 4th International Workshop on Predictor Models in Software Engineering*, Leipzig, Germany, 2008, pp. 47-54.

[91] Myrtveit, I., Stensrud, E., and Shepperd, M. J., "Reliability and Validity in Comparative Studies of Software Prediction Models", *IEEE Transactions on Software Engineering*, 31, 5, 2005, pp. 380-391.

[92] Nagappan, N., Murphy, B., and Basili, V. R., "The Influence of Organizational Structure on Software Quality: An Empirical Case Study", in *Proceedings 30th International Conference on Software Engineering (ICSE)*, 2008.

[93] Petrenko, M., *On Use of Dependency and Semantics Information in Incremental Change*, Wayne State University, Detroit, PhD Thesis, 2009.

[94] Porter, M., "An Algorithm for Suffix Stripping", *Program*, 14, 3, 1980, pp. 130-137.

[95] Poshyvanyk, D., *Using Information Retrieval to Support Software Maintenance Tasks*, Wayne State University, Detroit, PhD Thesis, 2008.

[96] Poshyvanyk, D., Guéhéneuc, Y. G., Marcus, A., Antoniol, G., and Rajlich, V., "Combining Probabilistic Ranking and Latent Semantic Indexing for Feature Identification", in *Proceedings 14th IEEE International Conference on Program Comprehension (ICPC'06)*, 2006, pp. 137-146.

[97] Poshyvanyk, D., Guéhéneuc, Y. G., Marcus, A., Antoniol, G., and Rajlich, V., "Feature Location using Probabilistic Ranking of Methods based on Execution Scenarios and Information Retrieval", *IEEE Transactions On Software Engineering*, 33, 6, 2007, pp. 420-432.

[98] Poshyvanyk, D. and Marcus, A., "The Conceptual Coupling Metrics for Object-Oriented Systems", in *Proceedings 22nd IEEE International Conference on Software Maintenance (ICSM'06)*, 2006.

[99] Poshyvanyk, D. and Marcus, A., "Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code", in *Proceedings 15th IEEE International Conference on Program Comprehension (ICPC'07)*, 2007, pp. 37-46.

[100] Poshyvanyk, D. and Marcus, A., "Using Information Retrieval to Support Design of Incremental Change of Software", in *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07)*, 2007, pp. 563-566.

[101] Poshyvanyk, D. and Marcus, A., "Using Traceability Links to Assess and Maintain the Quality of Software Documentation", in *Proceedings International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'07)*, 2007, pp. 27-30.

[102] Poshyvanyk, D., Marcus, A., and Dong, Y., "JIRiSS - an Eclipse plug-in for Source Code Exploration", in *Proceedings 14th IEEE International Conference on Program Comprehension (ICPC'06)*, 2006, pp. 252-255.

[103] Poshyvanyk, D., Marcus, A., Dong, Y., and Sergeyev, A., "IRiSS - A Source Code Exploration Tool", in *Proceedings 21st IEEE International Conference on Software Maintenance (ICSM'05)*, 2005, pp. 69-72.

[104] Poshyvanyk, D., Marcus, A., Ferenc, R., and Gyimóthy, T., "Using Information Retrieval based Coupling Measures for Impact Analysis", *Empirical Software Engineering*, 2009, pp. 5-32.

[105] Poshyvanyk, D. and Marcus, A., "Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code", in *Proceedings 15th IEEE International Conference on Program Comprehension (ICPC'07)*, 2007, pp. 37-48.

[106] Poshyvanyk, D., Petrenko, M., and Marcus, A., "Integrating COTS Search Engines into Eclipse: Google Desktop Case Study", in *Proceedings of the 2nd International ICSE'07 Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques (IWICSS'07)*, 2007, pp. 6-10.

[107] Poshyvanyk, D., Petrenko, M., Marcus, A., Xie, X., and Liu, D., "Source Code Exploration with Google", in *Proceedings 22nd IEEE International Conference on Software Maintenance (ICSM'06)*, 2006, pp.

[108] Ramakrishnan, N., "The Pervasiveness of Data Mining and Machine Learning", *IEEE Computer*, August, 2009, pp. 28-29.

[109] Ramakrishnan, S. and Menzies, T., "An ongoing OO software engineering measurement experiment ", in *Proceedings International Conference on Software Engineering: Education and Practice (SEEP)*, 1996, pp. 160-165.

[110] Ramakrishnan, S., Menzies, T., Hasslinger, M., Bok, P., McCarthy, H., Devakadadcham, B., and Moulder, D., "On building an effective measurement system for oo software process, product and resource tracking", in *Proceedings Tools Pacific*, 1996, pp. 239-247.

[111] Rocchio, J. J., "Relevance feedback in information retrieval", in *The SMART Retrieval System - Experiments in Automatic Document Processing*, Prentice Hall, 1971, pp. 313-323.

[112] Safyallah, H. and Sartipi, K., "Dynamic Analysis of Software Systems Using Execution Pattern Mining", in *Proceedings 14th International Conference on Program Comprehension*, 2006, pp. 84-88.

[113] Salton, G., *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer*, Addison-Wesley, 1989.

[114] Shepperd, M. and Schofield, C., "Estimating software project effort using analogies", *IEEE Transactions on Software Engineering*, 23, 12, 1997, pp. 736-743.

[115] Stefano, J. S. D. and Menzies, T., "Machine learning for software engineering: Case studies in software reuse", in *Proceedings IEEE Tools with Artificial Intelligence (ICTAI)*, 2002, pp. 246- 251.

[116] Tai, K. C. and Lie, Y., "A Test Generation Strategy for Pairwise Testing", *IEEE Transactions on Software Engineering*, 28, 1, 2002, pp. 109-111.

[117] Turhan, B., Menzies, T., Bener, A., and Distefano, J., "On the Relative Value of Cross-Company and Within-Company Data for Defect Prediction ", *Empirical Software Engineering*, 2009, pp. 540 - 578.

[118] Xie, T., Thummalapenta, S., Lo, D., and Liu, C., "Data Mining for Software Engineering", *IEEE Computer*, August 2009, 2009, pp. 55-62.

[119] Xie, X., Poshyvanyk, D., and Marcus, A., "Support for Static Concept Location with sv3D", in *Proceedings 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT'05)*, 2005, pp. 102-107.

[120] Xie, X., Poshyvanyk, D., and Marcus, A., "3D Visualization for Concept Location in Source Code", in *Proceedings 28th IEEE/ACM International Conference on Software Engineering (ICSE'06)*, 2006, pp. 839-842.

[121] Yang, Y. and Webb, G. I., "Weighted Proportional k-Interval Discretization for Naive-Bayes Classifiers", in *Advances in Knowledge Discovery and Data Mining*, 2003, pp. 565.

[122] Zhao, W., Zhang, L., Liu, Y., Sun, J., and Yang, F., "Sniafl: Towards a static non-interactive approach to feature location", in *Proceedings 26th International Conference on Software Engineering (ICSE'04)*, 2004, pp. 293–303.

[123] Zhu, X., Lafferty, J., and Ghahramani, Z., "Combining active learning and semi-supervised learning using Gaussian fields and harmonic functions", in *Proceedings International Conference on Machine Learning (ICML)*, 2003.