Recommending Software Artifacts from Repository Transactions

Joern David

Technical University Munich, Institute of Computer Science I1
Boltzmannstrasse 3, 85748 Garching, Germany
david@in.tum.de

Abstract. The central problem addressed by this interdisciplinary paper is to predict related software artifacts that are usually changed together by a developer. The working focus of programmers is revealed by means of their interactions with a software repository that receives a set of cohesive artifact changes within one commit transaction. This implicit knowledge of interdependent changes can be exploited in order to recommend likely further changes, given a set of already changed artifacts. We suggest a hybrid approach based on *Latent Semantic Indexing* (LSI) and machine learning methods to recommend software development artifacts, that is predicting a sequence of configuration items that were committed together. As opposed to related approaches to repository mining that are mostly based on symbolic methods like *Association Rule Mining* (ARM), our connectionist method is able to generalize onto unseen artifacts. Text analysis methods are employed to consider their textual attributes. We applied our technique to three publicly available datasets from the *PROMISE Repository of Software Engineering Databases*. The evaluation showed that the connectionist LSI-approach achieves a significantly higher recommendation accuracy than existing methods based on ARM. Even when generalizing onto unseen artifacts, our approach still provides an accuracy of up to 72.7% on the given datasets.

Keywords: Change impact analysis, recurrent neural networks, latent semantic indexing, repository mining, artifact recommendation.

1 Introduction

Change impact analysis is an activity within the field of change management that is meant for planning changes to be made to a system [3]. The predominant goal of the work at hand is to predict related software artifacts that are usually changed together due to their explicit or latent interdependencies. Typically software artifacts like requirement documents, use case diagrams or source code are managed as *configuration items* and are kept under version control. A *Configuration Management (CM) aggregate* [4], or shortly *commit set*, is a composite of configuration items that are committed within the same repository transaction. The software developers' working focuses should be learned

N.T. Nguyen et al. (Eds.): IEA/AIE 2008, LNAI 5027, pp. 189–198, 2008.

from their interaction with the artifact repository by means of *learning by example*. Developers deal with a work package of a software project and complete each of their working steps with a commit operation on the artifact repository. Especially two distinct tasks are to be solved by learning from the repository transaction data.

1. **Suggesting further changes:** Guiding a beginner or an experienced programmer who is unfamiliar with a certain subsystem and its configuration items. When a developer is working on a programming task, *relevant software artifacts* such as documentation, UML diagrams, code, multimedia files, etc. that might require corresponding changes should be *proactively* recommended: *"Programmers who changed these artifacts also changed.."* [18].
2. **Generalization:** The connectionist system should go beyond the state-of-art in guiding programmers, which is mostly based on symbolic association rules. The recommendation of related artifacts should even work for completely unlearned configuration items, which have not been under version control at the time of training. Based on their textual and conceptual similarity, the neural network provides a *content-based* assessment of unseen artifacts.

The combination of neural networks with *Latent Semantic Indexing* (LSI) is a novel and promising approach to change impact analysis, which is hardly investigated so far [17,8].

2 Related Work

Navigation and artifact recommendation as well as change impact analysis are highly relevant research areas [9,7,18]. In *"Mining Version Histories to Guide Software Changes"* Zimmermann et al. [18] applied association rule mining to analyze artifact changes that frequently occur together and thus are empirically related to each other. The current *situation* of a software developer is considered as a set of file changes that is used for mining association rules with a corresponding antecedent part *on the fly* (only on demand). The consequent parts of length one of all matching rules are ranked by confidence and are presented to the user. The implementation was done as an *Eclipse* plugin, such that the recommendations appear as a list in an integrated window of the development environment.

The main task of all recommendation systems is to discover the underlying functional interests that lead to common navigational activity among several users. Fu et al. already tried to exploit the tacit knowledge incorporated in the navigation history [5]. They propose an information recommendation system called *SurfLen*, which suggests interesting web pages to users. The underlying data mining technique is again association rule mining [2], which is used to discover *frequent 2-itemsets* containing web page URLs like $\{\{p, p_1\}, \{p, p_2\}, \ldots, \{p, p_n\}\}$. When the user is reading page p, then the associated pages $\{p_1, \ldots, p_n\}$ are recommended. So the items are URLs here and itemsets are ranked by an

intersection measure $rank(U_i, S_j) = |U_i \cap S_j|$ between users' browsing histories $U_i = \{p_1, \ldots, p_m\}$ and mined frequent itemsets $S_j = \{p'_1, \ldots, p'_m\}$. The best k itemsets according to the *rank*-measure are recommended, when the current browsing history matches one of the stored ones.

The software *NavTracks* keeps track of the navigation history of software developers by forming associations between related files, as described in *NavTracks: Supporting navigation in software maintenance* [16]. Its *symbolic algorithm* is based on navigation events and constructs associations between visited nodes. In the conducted case study, the navigation patterns of three users doing their everyday software maintenance tasks were captured. The *NavTracks* algorithm achieved an average accuracy of 29%, that is the next file was correctly recommended in 29% of the presented event streams respectively. Finally the industrial paper *"Empirical Software Change Impact Analysis using Singular Value Decomposition (SVD)"* [15] shows the use of SVD to analyze the changes in software project files that occur together. These are counted in a quadratic frequency matrix which is then decomposed by SVD in order to obtain the significances of the *association clusters*. These clusters indicate the strength of the associations between files. Likewise in the work at hand, a frequency matrix of keyterms and text units is computed and analyzed by SVD in a preprocessing step. This is a powerful method for removing redundancy and for eliciting the correlated latent concepts from an empirical dataset.

In both first named papers [18] and [5] the recommendation of software artifacts or web pages respectively is based on association rule mining. Thus useful *predictive information* in form of the artifact content is given away, if solely relying on symbolic items without consulting the similarity between artifacts. This drawback can be avoided by considering a rich representation of the visited entities instead of reducing them to meaningless unique identifiers.

3 The Recommendation Engine

The *Recurrent Neural Network* (RNN) is employed as enabling technology in this paper. The main purpose of the developed neural network model is to learn pairs of input and target sequences of coherent artifacts and to predict the most likely *target sequence* of related artifacts. Sequences of cohesive commit transactions upon a versioning repository are learned by the RNN in a supervised training process. Such a sequence is an ordered series of *multi-represented objects* $v_i \in V$ that consists of an input and target part $v_1, v_2, \ldots, v_k \mapsto v_{k+1}, v_{k+2}, \ldots, v_{k+m}$. Multi-represented objects [1] capture several attributes (aspects) of the same object like *Name*, *Identifier* and domain-specific attributes that store domain information: *FileType*, *FilePath*, *AuthorID*, etc. The values of these repository transaction descriptors are technically incorporated into a *Vector Space Model* (VSM) ([12], cp. section 4), which is an algebraic model equipped with a similarity measure where each dimension corresponds to a domain attribute. So configuration items are realized as multi-represented objects with a variable number of domain-specific attributes.

Sets A of committed software artifacts, the *Configuration Management aggregates*, are split into input and target part X and Y, where X runs through all subsets of A. This can be done in $n := \sum_{k=1}^{N-1} \binom{N}{k} = 2^N\text{-}2$ many ways, where $N := |A|$ is the cardinality of the commit set A. Apparently the training patterns P are obtained as mapping onto the complementary set of the generated CM-subsets: $P \equiv X \mapsto (A \setminus X)$. Since the RNN is a sequence processor, these sets are transformed into *sequences* by imposing an arbitrary but fixed lexicographical order onto the configuration items within each of the two parts X and Y. To be comparable with the ARM-based related work [18] and to reduce complexity, the target part of a sequence was also restricted to length one, which constrains the obtainable recall.

The neural prediction is computed on the basis of the recent *transaction history* $\boldsymbol{x}_{t-k}, \ldots, \boldsymbol{x}_t$ that reflects the user behavior. Figure 1 illustrates the topology and the information flow of the proposed recurrent neural network. $\boldsymbol{y}_{t+1}, \ldots, \boldsymbol{y}_{t+m}$ represents the associated sequence of target artifacts, which are predicted to correlate strongest with the observed user behavior. An important characteristics of the recurrent network design is its inherent temporal memory. Due to the *temporally unfolded network topology*, the RNN can learn the *sequential structure* of a set of node sequences. Thus the navigation behavior of the user is explicitly modeled in the *hidden state layer* $\boldsymbol{s}_{t-k}, \ldots, \boldsymbol{s}_{t+m-1}$. Its neuron records (the block arrows depicted in figure 1) serve as *hidden* and as *context units*, because $\boldsymbol{s}_{t-1}$ provides the context for the recursive computation of the subsequent hidden state $\boldsymbol{s}_t$. Thereby one input vector $\boldsymbol{x}_t$ or output vector $\boldsymbol{y}_{t+1}$ corresponds to one configuration item.

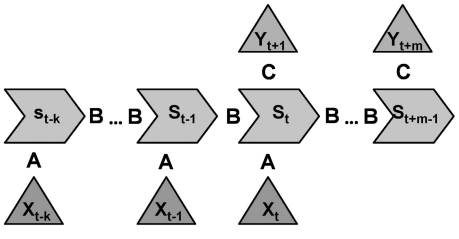

Fig. 1. Schematic topology of the proposed modular recurrent network RNN. The block arrows indicate the internal state transition $\boldsymbol{s}_t \rightarrow \boldsymbol{s}_{t+1}$. A, B and C are weight matrices. $\boldsymbol{x}_t$ is the external input vector at time t, $\boldsymbol{y}_{t+1}$ is the correspondingly predicted output vector. The depicted block arrow direction shows the forward propagation phase.

In order to process variably long node sequences $\boldsymbol{x}_{t-k}$, ..., $\boldsymbol{x}_t$, $\boldsymbol{y}_{t+1}$, ..., $\boldsymbol{y}_{t+m}$ as part of the same training set, we developed a modular recurrent neural network. The technical design of the RNN is defined by the following recurrent model of forward propagation.

- A is a ${}^h\mathbb{R}^{d_1}$, B is a ${}^h\mathbb{R}^h$ and C is a ${}^{d_2}\mathbb{R}^h$ matrix.
- d_1 is the dimensionality of the input- and d_2 is the dimensionality of the output feature space. f is a sigmoid activation function (logistic function).

– $h = dim(\boldsymbol{s}_i)$ is the dimensionality of the state layer ($i = t\text{-}k, \ldots, t{+}m$). h is independent from d_1 and d_2 and was set to $h = 20$ (experimentally determined) to provide sufficient network resources.

$$\boldsymbol{s}_t = f(B\boldsymbol{s}_{t-1} + A\boldsymbol{x}_t) \tag{1}$$

$$\boldsymbol{o}_{t+1} = f(C\boldsymbol{s}_t), \qquad f(x) = \frac{1}{1+\exp(-x)} \tag{2}$$

$$\boldsymbol{o}_{t+i} \overset{training}{\longrightarrow} \boldsymbol{y}_{t+i},\ i = 1, \ldots, m \tag{3}$$

As introduced above, $\boldsymbol{s}_t \in S$ stands for the internal state at the discrete time step t. The crucial recurrent equation 1 combines an external input $\boldsymbol{x}_t$ with the previous state $\boldsymbol{s}_{t-1}$ to the subsequent state $\boldsymbol{s}_t$, which indirectly depends on all foregoing external inputs $\boldsymbol{x}_{t-k}, \ldots, \boldsymbol{x}_{t-1}$ *and* internal states $\boldsymbol{s}_{t-k}, \ldots, \boldsymbol{s}_{t-1}$. In case of supervised network training, the target entities $\boldsymbol{y}_{t+1}, \ldots, \boldsymbol{y}_{t+m}$ are known, while during network application the *output sequence* $\boldsymbol{o}_{t+1}, \ldots, \boldsymbol{o}_{t+m}$ is computed solely based on the respective inputs. The RNN is trained with a modified *Backpropagation Through Time* (BPTT) algorithm [6] and is able to process variably dimensional vectors $\boldsymbol{x}_i$ and $\boldsymbol{y}_j$. There are no further external inputs after t, since the observed input sequence is exhausted. For $t{+}1, t{+}2, \ldots$ the network propagates activations $(\boldsymbol{s}_t)_{t\in\mathbb{N}}$ only through the *state transition matrix* B of the *continuous internal state layer*, which enables infinitely long prediction sequences in theory. Independent of the length k of the input sequence or the length m of the target sequence, the weight matrices A, B and C are continuously adjusted in the training process (*online training*), because they are reused in every time step t. This implies that there are exactly three matrix instances for the training of all variably long sequences. During one training epoch, the network adapts at runtime to the individual structure of the respective node sequence with its input and target part. The advantage of the flexible network topology is that a prediction can still be computed when only a subsequence $(v_i, v_{i+1}, \ldots, v_k) \subset (v_1, v_2, \ldots, v_k)$, $i > 1$, of the input part is available. This property is very useful for guiding programmers that start to explore a new working space, since the recommendation can be done incrementally. When observing an input (v_1, v_2) for example, v_3 is predicted and added to the current input sequence in turn. This extended input (v_1, v_2, v_3) is subsequently used as basis for the next recommendation and so on. In the operational application of the trained network, two functions are provided:

1. *Recognition and Recovery:* When presenting an input sequence of artifacts to the RNN that equals a set of trained configuration items, the remaining items of that CM aggregate are *recovered* and *recommended*.
2. *Generalization:* A virtually unknown sequence of software artifacts is fed into the RNN, which is then able to predict related artifacts that are most likely changed together.

4 Application to Textual Artifact Descriptors

Each artifact that contains unstructured text has to be transformed into a numerically processable representation. The content is considered by a text mining approach that provides a rich artifact representation with the semantics of the textual attributes of the repository transaction data, since conceptual similarity between artifacts is imposed thereby.

Rich Representation Based on Text Mining. In case of textual content, we apply well-known text mining methodologies for computing a numerical document representation. According to the vector space model [12], a feature vector $\boldsymbol{x} \in \mathbb{R}^d$, $d := d_1$, (cp. section 3) is computed for each multi-represented object $v \in V$ of the knowledge base V (*bag-of-words* approach). The well-known preprocessing steps *stemming* [11] and *stop-word removal* were realized by the *Apache Lucene* indexing and search framework[1]. The vector space model can be refined by applying *Latent Semantic Indexing* [8] to the set of obtained feature vectors. Thereby a matrix $M_{i,j}$ of keyword frequencies per text unit (e.g. file paths, sentences, paragraphs, sections, documents) is spanned. The rows denote the frequency of occurrence for term i in text unit j, which is the textual content of an artifact in the knowledge base. In this case study, the text units are the file paths of the software artifacts and the terms are the single path components like the artifact name. The matrix is decomposed by *Singular Value Decomposition*, which is a generalization of the *Principal Component Analysis* (PCA) that determines the inherent key concepts that characterize all d-dimensional feature vectors. SVD is able to analyze the correlations among terms as well as the correlations between text units and comprised terms, which are described by a non-quadratic matrix. Thereby the term-frequency matrix $M = UDW^T$ is decomposed into two orthonormal matrices U and W and one diagonal matrix D. After *diagonalizing* that matrix M, the singular values $\sigma_j = D_{j,j}$ in the diagonal of the matrix D reveal the insignificant dimensions, which can be discarded. These k least informative dimensions with singular values $\sigma_{d-k}, \sigma_{d-k+1}, \ldots, \sigma_n$ are ignored by the transformation to a (d-k)-dimensional subspace. The resulting feature vectors $\boldsymbol{x}_j \in \mathbb{R}^{d-k}$ represent the content of an artifact $v_j \in V$: $\boldsymbol{x}_j^T := (W_{1,j}^T, W_{2,j}^T, \ldots, W_{d-k,j}^T)$, where $W_{i,j}^T$, $i = 1, \ldots, d\text{-}k$, $j = 1, \ldots, |V|$ are the entries of the transposed right-singular matrix. This global vector space model enables the RNN to learn the latent semantics of the domain knowledge.

Software Repository Transaction Data. We applied our recommendation technique to repositories of versioning data from three independent projects, namely *Nickle*, *XFree86* and *X.org* (http://{nickle.org, xfree86, x}.org). The used datasets stem from the *PROMISE Repository of Software Engineering Databases* [13], which is a collection of publicly available datasets and tools to serve researchers in building predictive software models (PSMs). Massey [10] has logged and analyzed the publicly available CVS archives of these projects

[1] http://lucene.apache.org

resulting in datasets with the same structure, such that we could conduct a uniform and comparable evaluation. The file names like "*INSTALL-X.org*" or "*Makefile*" (attribute *FilePath* in table 1) are the only content-based attributes of the CVS transaction data and are considered as text units in terms of section 4 for constructing the vector space model. The remainder of the attributes like *external* (boolean, author is different from submitting person) are mostly nominal and thus do not provide the possibility to compute similarity among their discrete values. On the other hand, the metric attributes *lines_added* and *lines_removed* revealed to be hardly significant and thus were excluded. Table 1 shows the schema of the logged repository transactions. Now the commit records

Table 1. Data schema of the repository transactions from the *XFree86 Project* concerning the configuration items with an excerpt of their attributes

FileType	FilePath	AuthorID	Revision	Commit Date
"code"	"config/imake/imake.c"	1	2.0	"2004-04-03 22:26:20"
"devel-doc"	"INSTALL-X.org"	1	1.6	"2005-02-11 03:02:53"
"doc"	"config/cf/Fresco.tmpl"	1	1.2	"2005-02-11 03:02:53"
"unknown"	"LABEL"	2	1.2	"2003-07-09 15:27:23"

were grouped according to their discrete commit dates, obtaining sets of artifacts that were committed by a specific author at the respective point of time (atomic). The contents of each *CM aggregate* were split into input and target sequences and one pair made up a single training pattern for the neural network.

5 Evaluation

Compared to association rule mining, which is a purely symbolic approach operating on atomic artifact identifiers, our connectionist technique paired with text mining methods is able to incorporate actual content into the learning and recommendation process. The association rule mining approach that is often used in related work [18] is not able to assess changes or visits of *new* artifacts in principle, since their (latent) semantics are ignored by the symbolic representation. On the other hand, we did not address source code fragments like done by Zimmermann et al. [18] but focused on versioned files in general, which are distinguished by type as listed in table 1.

For our case study we chose arbitrary subsets of 2,970 till 5,350 commit actions by 6 till 39 anonymous developers, depending on the respective project. The *generalization* accuracy for the target attribute *FilePath* was evaluated by arbitrary test sets with different numbers of configuration items from the three projects. The repository transaction data was arbitrarily split up into a training set of $\frac{7}{8}$ and a test set of $\frac{1}{8}$ of the relevant commit actions by interleaving (taking each 8^{th} pattern). Since no internal order is specified on the *CM aggregates*, the artifacts of one commit set C are represented by the possible combinations of ordered input and target sequences making up one training pattern

($input \mapsto target$). Thus the RNN learns the committed items as subsequences of the form $(C \supset (item^C_{i_1}, item^C_{i_2}, \ldots, item^C_{i_n}) \mapsto item^C_{j_1})$, where the target sequence only consists of one item. Despite the relatively short sequences of 3.3 configuration items per *CM aggregate* on average (*min* 1, *max* 34), we use a neural network for predicting related software artifacts because of its *generalization capability* [14]XX, which reveals its effect when visiting *unseen* artifacts that were not part of the training sequences. A successful recommendation is given, if a set of items $H := \{item_{i_1}, item_{i_2}, \ldots, item_{i_m}\} \subset C$ that was not trained before is presented and the network predicts one or several items of the complementary set $T := C \setminus \{item_{i_1}, item_{i_2}, \ldots, item_{i_m}\}$. In general, all items $u \in H$ of the input set are presented and one or several items $v \in T$ of the target subset are expected as correct recommendation, which counts as hit and is summed over all test cases.

The singular value decomposition of the term-document matrix as integral step of LSI succeeded to discover the latent semantic concepts of the textual attribute *FilePath* and optimized the term-frequency based *bag-of-words* approach. We were able to reduce the vector space dimensionality to $d'_1 = d_1 - k = 159$ by $k = 7$ while loosing only 1.06% of the variance in the training set (nearly lossless transform, *XFree* dataset). This was possible because high-dimensional bag-of-words vectors are usually *sparse*, that is to say those contain many 0-entries. More formally spoken, the term-document matrix $M \in \mathbb{R}^{d_1 \times |V|}$ (d_1 is the number of keyterms in the knowledge base) does not have full rank, but only $rank(M) = d_1 - k$. We did not conduct an actual dimension reduction with information loss by $k > 7$, because any removal of further dimensions led to a significant downgrade of the prediction accuracy[2].

Before presenting the quantitative results of our recommendation technique, we show two exemplary item recommendations generated from partially *unseen* commit histories of the *XFree* project. Both results were validated by hand using appropriate queries on the transaction database.

- *1 input, 1 target item:* ***config/cf/xfree86.cf*** $\mapsto$ ***config/cf/xf86site.def***
  It stands to reason that *config/cf/xf86site.def* was predicted due to its conceptual similarity to the input item *config/cf/xfree86.cf* that resides in the same subdirectory *config/cf*, which is detected by LSI-based text mining.
- *2 input, 1 target item:* ***Makefile.std, registry*** $\mapsto$ ***Imakefile***
  In this case the item *Imakefile* of type *build* was recommended, since the neural network had learned that committing the items *Makefile.std* and *registry* mostly implies committing the *Imakefile*, too.

We trained the network on up to 455 training patterns resulting from a basic set of 3,000 commits, whereupon the target items in all test sets could be inferred with a maximal accuracy of 72.72% in case of the *X.org* project. For the *Nickle* dataset we achieved an accuracy of 61.29% compared to 69.23% for the *XFree* dataset. Note that the residual training error could not be minimized completely (relative training error $0.25\% \leq \delta \leq 0.75\%$) and still might leave

[2] The smallest kept singular value $\sigma_{159} = 0.999 \approx \frac{6}{100}\sigma_1$ was still significant (X.org).

room for improvements. We did not strive for a perfect mapping of all training sequences in order to avoid overfitting. The significance of all results is quite high, since the dependent variable *FilePath* (attribute to be predicted) can take 167 (XFree) till 1047 (X.org) different values, depending on the project. Assuming an equal distribution of these values, guessing a single target item based on a given set of committed X.org project items would result in a hit probability of $1/1047 = 0.096\%$ on average. We suppose that even better results would be achievable in the presence of more meaningful and more predictive attributes than the path of a software artifact, which does not offer a variety of meaningful keyterms that would form a broad basis for textual similarity. The lack of rich content in the *FilePath* attribute, whose values mostly consist of only one till three terms, represents a deficiency compared to richer multi-represented domain objects.

6 Conclusion

We have introduced semantic learning and prediction of related artifact changes based on our *Recurrent Neural Network* (RNN). The RNN enables learning of variably long node sequences on graph-based and domain-specific knowledge bases such as transaction repositories. We trained the network on the cohesive configuration items of each commit set in a supervised manner leading to a recommender system that fulfills two main tasks:

1. ***Preventing Incomplete Change.*** Guidance of programmers by observing their working set and recommending related artifacts to prevent errors resulting from incomplete changes.
2. ***Inference of Related Artifacts.*** The learned dependencies among the items of each commit set are exploited by the RNN, which *generalizes* onto *unseen* software artifacts and thus even provides advice for new configuration items.

For both functionalities a knowledge representation based on *Latent Semantic Indexing* of text attributes builds the foundation for semantic recommendation. The evaluation was conducted by three independent datasets from the *PROMISE Repository of Software Engineering Databases* and revealed a prediction accuracy for partially unseen artifact sets of 67.75% altogether (maximum of 72.72% for the XFree project), based on a result space of 167 till 1047 different artifacts. Compared to existing methods based on association rule mining whose authors report accuracy values of about 26% till 50% [18], our approach even achieves a higher performance when applied to untrained item sets, which is not possible with symbolic techniques at all. For example, the evaluation of the *NavTracks* algorithm (cp. section 2) conducted by Singer, Robert and Storey [16] led to an accuracy of about 29%, based on the logged navigation pathways of three developers.

References

1. Achtert, E., Kriegel, H.-P., Pryakhin, A., Schubert, M.: Hierarchical density-based clustering for multi-represented objects. In: Workshop on Mining Complex Data (MCD 2005), ICDM, Houston, TX, Institute for Computer Science, University of Munich (2005)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proc. of the 20th Int. Conf. on Very Large Data Bases, VLDB, Santiago, Chile (1994)
3. Bohner, S., Arnold, R.: Software change impact analysis, pp. 1–28. IEEE Computer Society Press, Los Alamitos (1996)
4. Bruegge, B., Dutoit, A.H.: Object-Oriented Software Engineering Using UML, Patterns, and Java. Prentice-Hall, Englewood Cliffs (2004)
5. Budzik, J., Fu, X., Hammond, K.J.: Mining navigation history for recommendation. In: Proceedings of the 5th international conference on Intelligent user interfaces, New Orleans, Louisiana, United States, pp. 106–112. ACM Press, New York (2000)
6. Callan, R.: Neuronale Netze im Klartext. Pearson Studium, London (2003)
7. Cubranic, D., Murphy, G.C.: Hipikat: Recommending pertinent software development artifacts. In: 25th International Conference on Software Engineering (ICSE 2003), pp. 408–418 (2003)
8. Deerwester, S., Dumais, S.T., Furnas, G.W., Harshman, R., Landauer, T.K.: Indexing by latent semantic analysis. Technical Report 6 (1990)
9. Gall, H., Hajek, K., Jazayeri, M.: Detection of logical coupling based on product release history. In: International Conference on Software Maintenance (ICSM 1998) (1998)
10. Massey, B.: Nickle repository transaction data, Computer Science Dept., Portland State University, Portland, OR, USA (2005)
11. Porter, M.: An algorithm for suffix stripping. Technical Report 3 (1980)
12. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Communications of the ACM 18, 613–620 (1975)
13. Sayyad Shirabad, J., Menzies, T.: The PROMISE repository of software engineering databases. School of Information Technology and Engineering, University of Ottawa, Canada (2005)
14. Schmidhuber, J.: Discovering neural nets with low kolmogorov complexity and high generalization capability. Neural Networks 10, 857–873 (1997)
15. Sherriff, M., Lake, M., Williams, L.: Empirical software change impact analysis using singular value decomposition. IBM, North Carolina State University (2007), ftp://ftp.ncsu.edu/pub/unity/lockers/ftp/csc_anon/tech/2007/TR-2007-13.pdf
16. Singer, J., Elves, R., Storey, M.-A.: Navtracks: Supporting navigation in software maintenance. In: Proceedings of the International Conference on Software Maintenance (2005)
17. Syu, I., Lang, S.D., Deo, N.: Incorporating latent semantic indexing into a neural network model for information retrieval. In: Proceedings of the fifth international conference on Information and knowledge management, pp. 145–153 (1996)
18. Zimmermann, T., Weißgerber, P., Diehl, S., Zeller, A.: Mining version histories to guide software changes. In: International Conference on Software Engineering (ICSE 2004) (2004)