SEMI-AUTOMATIC ASSIGNMENT OF WORK ITEMS

Jonas Helming, Holger Arndt, Zardosht Hodaie, Maximilian Koegel, Nitesh Narayan
Institut für Informatik,Technische Universität München, Garching, Germany
{helming, arndt, hodaie, koegel, narayan}@in.tum.de

Keywords: Machine Learning, Task Assignment, Bug Report, UNICASE, Unified Model, UJP

Abstract: Many software development projects maintain repositories managing work items such as bug reports or tasks. In open-source projects, these repositories are accessible for end-users or clients, allowing them to enter new work items. These artifacts have to be further triaged. The most important step is the initial assignment of a work item to a responsible developer. As a consequence, a number of approaches exist to semi-automatically assign bug reports, e.g. using methods from machine learning. We compare different approaches to assign new work items to developers mining textual content as well as structural information. Furthermore we propose a novel model-based approach, which also considers relations from work items to the system specification for the assignment. The approaches are applied to different types of work items, including bug reports and tasks. To evaluate our approaches we mine the model repository of three different projects. We also included history data to determine how well they work in different states.

1 INTRODUCTION

Many software development projects make use of repositories, managing different types of work items. This includes bug tracker systems like Bugzilla , task repositories like Jira and integrated solutions such as Jazz or the Team Foundation Server . A commonality of all these repositories is the possibility to assign a certain work item to a responsible person or team (Anvik 2006).

It is a trend in current software development to open these repositories to other groups beside the project management allowing them to enter new work items. These groups could be end-users of the system, clients or the developers themselves. This possibility of feedback helps to identity relevant features and improves the quality by allowing more bugs to be identified (Raymond 1999). But this advantage comes with significant cost (Anvik et al. 2006), because every new work item has to be triaged. That means it has to be decided whether the work item is important or maybe a duplicate and further, whom it should be assigned to. As a part of the triage process it would be beneficial to support the assignment of work items and automatically select those developers with experience in the area of this work item. This developer is probably a good candidate to work on the work item, or, if the developer will not complete the work item himself, he probably has the experience to further triage the work item and reassign it. There are several approaches, which semi-automatically assign work items (mostly bug reports) to developers. They are based on mining existing work items of a repository. We will present an overview of existing approaches in section 2.1.

In this paper we compare different existing techniques of machine learning and propose a new model-based approach to semi-automatically assign work items. All approaches are applied to a unified model, implemented in a tool called UNICASE (Bruegge et al. 2008). The unified model is a repository for all different types of work items. Existing approaches usually focus on one type of work item, for example bug reports. The use of a unified model enables us to apply and evaluate our approach with different types of work items, including bug reports, feature requests, issues and tasks. We will describe UNICASE more in detail in section 3.

UNICASE does not only contain different types of work items, but also artifacts from the system specification, i.e. the system model (Helming et al. 2009). Work items can be linked to these artifacts from the system specification as illustrated in Figure 1. For example a task or a bug report can be linked to a related functional requirement. These links provide additional information about the context of a work item, which can be mined for semi-automatic assignment, as we will show in section 4. Our new approach for semi-automatic task assignment, called model-based approach, processes this information. The results of this approach can be transferred to other systems such as bug trackers where bug reports can be linked to affected components.

We found that existing approaches are usually evaluated in a certain project state (state-based), which means that a snap shot of the project is taken at a certain time and all work items have a fixed state. Then the assigned work items are classified by the approach to be evaluated and the results are compared with the actual assignee at that project state. We use this type of evaluation in a first step. However, state-based evaluation has two shortcomings: (1) The approach usually gets more information than it would have had at the time a certain work item was triaged. For example, additional information could have been attached to a work-item, which was not available for initial triage. (2) No conclusion can be made, how different approaches work in different states of a project, for example depending on the number of work items or on personal fluctuations. Therefore we evaluated our method also “history-based” which means that we mine all states of the project history and make automatic assignment proposals in the exact instance when a new work item was created. We claim that this type of evaluation is more realistic than just using one later state where possible more information is available. We evaluate our approach by mining data from three different projects, which use UNICASE as a repository for their work items and system model. To evaluate which approach works best in our context as well as for a comparison of the proposed model-based approach we apply different machine learning techniques to assign work items automatically. These include very simple methods such as nearest neighbor, but also more advanced methods such as support vector machines or naive Bayes.

The paper is organized as follows: Section 2 summarizes related work in the field of automated task assignment as well as in the field of classification of software engineering artifacts. Section 3 introduces the prerequisites, i.e. the underlying model of work items and UNICASE, the tool this model is implemented in. Section 4 and 5 describe the model-based and the different machine learning approaches we applied in our evaluation. Section 6 presents the results of our evaluation on the three projects, in both a state-based and a history-based mode. In section 7 we conclude and discuss our results.

2 RELATED WORK

In this section we give an overview over relevant existing approaches. In section 2.1 we describe approaches, which semi-automatically assign different types of work items. In section 2.2 we describe approaches, which classify software engineering artifacts using methods from machine learning and which are therefore also relevant for our approach.

2.1 Task Assignment

In our approach we refer to task assignment as the problem of classifying work items to the right developer. Determining developer expertise is the basis for the first part of our approach. In our case this is done by mining structured project history data saved within the UNICASE repository.

Most of the approaches for determining expertise rely on analyzing the code base of a software project mostly with the help of version control systems. (Mockus & Herbsleb 2002) treat every change from a source code repository as an experience atom and try to determine expertise of developers by counting related changes made in particular parts of source code. (Schuler & Zimmermann 2008) introduce the concept of usage expertise, which describes expertise in the sense of using source code, e.g. a specific API. Based on an empirical study, (Fritz et al. 2007) showed that these expertise measures acquired from source code analysis effectively represent parts of the code base, which the programmer has knowledge for. (Sindhgatta 2008) uses linguistic information found in source code elements such as identifiers and comments to determine the domain expertise of developers.

Other task classification approaches use information retrieval techniques such as text categorization to find similar tasks. Canfora et al. (Canfora & Cerulo o 2005) demonstrate how information retrieval on software repositories can be used to create an index of developers for the assignment of change requests. (J. Anvik 2006) investigate applying different machine learning algorithms to an open bug repository and compare precision of resulting task assignments. (Anvik et al. 2006) apply SVM text categorization on an open bug repository for classifying new bug reports. They achieve high precision on the Eclipse and Firefox development projects and found their approach promising for further research. (Čubranić 2004) employ text categorization using a naive Bayes classifier to automatically assign bug reports to developers. They correctly predict 30% of the assignments on a collection of 15,859 bug reports from a large open-source project. Yingbo et al. (Yingbo et al. 2007) apply a machine learning algorithm to workflow event log of a workflow system to learn the different activities of each actor and to suggest an appropriate actor to assign new tasks to.

2.2 Artifact Classification

Machine learning provides a number of classification methods, which can be used to categorize different items and which can also be applied to software artifacts. Each item is characterized by a number of attributes, such as name, description or due date, which have to be converted into numerical values to be useable for machine learning algorithms. These algorithms require a set of labeled training data, i.e. items for which the desired class is known (in our case the developer who an item has been assigned to). The labeled examples are used to train a classifier, which is why this method is called "supervised learning". After the training phase, new items can be classified automatically, which can serve as a recommendation for task assignment. A similar method has been employed by (Čubranić 2004) who used a naive Bayes classifier to assign bug reports to developers. In contrast to their work, our approach is not limited to bug reports, but can rather handle different types of work items. Moreover, we evaluate and compare different classifiers. Also (Bruegge et al. 2009) have taken a unified approach and used a modular recurrent neural network to classify status and activity of work items.

3 PREREQUISITES

We implemented and evaluated our approach for semi-automated task assignment in a unified model provided by the tool UNICASE . In this section we will describe the artifact types we consider for our approach. Furthermore we describe the features of these artifacts, which will form the input for the different approaches. UNICASE provides a repository, which can handle arbitrary types of software engineering artifacts. These artifacts can either be part of the system model, i.e. the requirements model and the system specification, or the project model, i.e. artifacts from project management such as work items or developers (Helming et al. 2009)

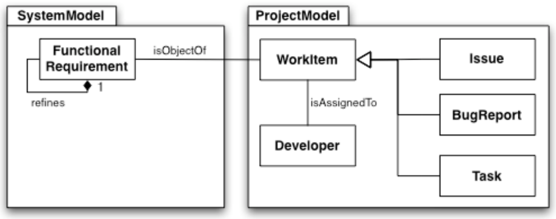

Figure 1: Excerpt from the unified model of UNICASE (UML class diagram)

Figure 1 shows the relevant artifacts for our approach. The most important part is the association between work item and developer. This association expresses, that a work item is assigned to a certain developer and is therefore the association we semi-automatically want to set. Work items in UNICASE can be issues, tasks or bug reports. As we apply our approach to the generalization work item it is not limited to one of the subtypes as in existing approaches. As we proposed in previous work ((J. Helming et al. 2009)), work items in UNICASE can be linked to the related Functional Requirements modeled by the association *isObjectOf*. This expresses that the represented work of the work item is necessary to fulfill the requirement. This association, if already existent adds additional context information to a work item. Modeled by the *Refines* association, Functional requirements are structured in a hierarchy. We navigate this hierarchy in our model-based approach to find the most experienced developer, described in section 4. As a first step in this approach, we have to determine all related functional requirements of the currently inspected work item. As a consequence this approach only works for work items, which are linked to functional requirements.
While the model-based approach of semi-automated task assignment only relies on model links in UNICASE, the machine learning approaches mainly rely on the content of the artifacts. All content is stored in attributes. The following table provides an overview of the relevant features we used to evaluate the different approaches:

Feature	Meaning
Name	A short and unique name for the represented work item.
Description	A detailed description of the work item.
ObjectOf	The object of the work item, usually a Functional Requirement.

We will show in the evaluation section, which features had a significant impact on the accuracy of the approach.
UNICASE provides an operation-based versioning for all artifacts (Koegel 2008). This means all past project-states can be restored. Further we can retrieve a list of operations for each state, for example when a project manager assigned a work item to a certain developer. We will use this versioning system in the second part of our evaluation to exactly recreate a project state where a work item was created. The goal is to evaluate whether our approach would have chosen the same developer for an assignment as the project manager did. This evaluation method provides a more realistic result than evaluating the approaches only on the latest project state. With this method both approaches, machine learning and model-based, can only mine the information, which was present at the time of the required assignment recommendation.

4 MODEL-BASED APPROACH

For the model-based assignment of work items we use the structural information available in the unified model of UNICASE. In UNICASE every functional requirement can have a set of linked work items. These are work items that need to be completed in order to fulfil this requirement.
The main idea of our model-based approach is to find the relevant part of the system for the input work item. In a second set we extract a set of existing work items, which are dealing with this part of the system. For a given input work item and based on this set we select a potential assignee. We will describe how this set is created using an example in Figure 2.
The input work item W is linked to the functional requirement B. To create the relevant set of work items (RelevantWorkItems(W)) we first add all work items, which are linked to functional requirement B (none in this example). Furthermore we add all work items linked to the refined functional requirement (A) and all work items linked to the refining requirements (C). In the example the set would consist of the work items 1 and 2. Futhermore, we recursively collect all work items from the refiningRequirements of A, which are neighbors of functional requirement B in the hierarchy (not shown in the example).
Using the set RelevantWorkItems(W) we determine expertise of each developer D regarding W (Expertise$_w$(D)). We defined Expertise$_w$(D) as the number of relevant work items this developer has already completed. After determining Expertise$_w$(D) for all developers, the one with highest expertise value is suggested as the appropriate assignee of the work item W.

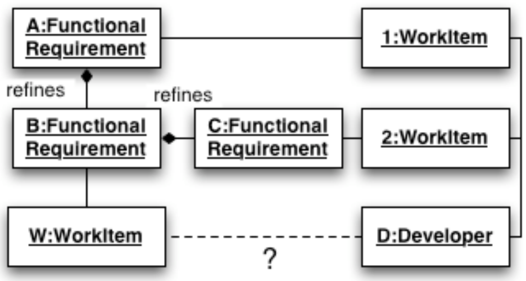

Figure 2: Example for the model-based approach (UML object diagram)

5 MACHINE LEARNING APPROACHES

We have used the Universal Java Matrix Library (UJMP) (Arndt et al. 2009) to convert data from UNICASE into a format suitable for machine learning algorithms. This matrix library can process numerical as well as textual data and can be easily integrated into other projects. All work items are aggregated into a two-dimensional matrix, where each row represents a single work item and the columns contain the attributes (name, description, ObjectOf association). Punctuation and stop words are removed and all strings are converted to lowercase characters. After that, the data is converted into a document-term matrix, where each row still represents a work item, while the columns contain information about the occurrence of terms in this work item. There are as many columns as different words in the whole text corpus of all work items. For every term, the number of occurrences in this work item is counted. This matrix is normalized using tf-idf (term frequency / inverse document frequency)

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

,where $n_{i,j}$ is the number of occurrences of the term t_i in document d_j, and the denominator is the sum of occurrences of all terms in document d_j.

$$idf_i = \log \frac{|D|}{|\{d : t_i \in d\}|}$$

The inverse document frequency is a measure of the general importance of the term: logarithm of total number of documents in the corpus divided by number of documents where the term t_i appears. A deeper introduction to text categorization can be found in (Sebastiani 2002).
We have not used further preprocessing such as stemming or latent semantic indexing (LSI) as our initial experiments suggested, that it had only a minor effect on performance compared to the selection of algorithm or features. We have used the tf-idf matrix as input data to the Java Data Mining Package (JDMP) (Arndt 2009), which provides an common interface to numerous machine learning algorithms from various libraries. Therefore we were able to give a comparison between different methods:

Constant Classifier
The work items are not assigned to all developers on an equal basis. One developer may have worked on much more work items than another one. By just predicting the developer with the most work items it is possible to make many correct assignments. Therefore we use this classifier as a baseline, as it does not consider the input features.

Nearest Neighbor Classifier
This classifier is one of the simplest classifiers in machine learning. It uses normalized Euclidean distance to locate the item within the training set which is closest to the given work item, and predicts the same class as the labeled example. We use the implementation IB1 from Weka (Witten & Frank 2002). We did not use k-nearest neighbors, which usually performs much better, because we found the runtime of this algorithm to be too long for practical application in our scenario.

Decision Trees
Decision trees are also very simple classifiers, which break down the classification problem into a set of simple if-then decisions which lead to the final prediction. Since one decision tree alone is not a very good predictor, it is a common practice to combine a number of decision trees with ensemble methods such as boosting (Freund & Schapire 1997). We use the implementation RandomCommittee from Weka.

Support Vector Machine (SVM)
The support vector machine (SVM) calculates a separating hyperplane between data points from different classes and tries to maximize the margin between them. We use the implementation from LIBLINEAR (Fan et al. 2008), which works extremely fast on large sparse data sets and is therefore well suited for our task.

Naïve Bayes
This classifier is based on Bayes' theorem in probability theory. It assumes that all features are independent which is not necessarily the case for a document-term matrix. However, it scales very well to large data sets and usually yields good results even if the independence assumption is violated. We use the implementation NaiveBayesMultinomial from Weka (Witten & Frank 2002) but also considered the implementation in MALLET , which showed lower classification accuracy (therefore we only report results from Weka).

Neural Networks
Neural networks can learn non-linear mappings between input and output data, which can be used to classify items into different classes (for an introduction to neural networks see e.g. (Haykin 2008)). We have tried different implementations but found that the time for training took an order of magnitudes longer than for the other approaches considered here. Therefore we were unable to include neural networks into our evaluation.

For the state-bases evaluation, we trained these classifiers using a cross validation scheme: The data has been split randomly into ten subsets. Nine of these sets were selected to train the classifier and one to assess its performance. After that, another set was selected for prediction, and the training has been performed using the remaining nine sets. This procedure has been performed ten times for all sets and has been repeated ten times (10 times 10-fold cross validation). For the history-based evaluation, the classifiers were trained on the data available at a certain project state to predict the assignee for a newly created work item. After the actual assignment through the project leader, the classifiers were re-trained and the next prediction could be made.

Depending on the approach, runtime for the evaluation of one classifier on one project ranged from a couple of minutes for LIBLINEAR SVM to almost two days for the nearest neighbor classifier. Although a thorough comparison of all machine learning methods would certainly have been interesting, we did not include a full evaluation on all projects and performed feature selection using LIBLINEAR, which was the fastest method of all. We argue that an algorithm for automatic task assigment would have to deliver a good accuracy but at the same time the necessary performance in terms of computing time to be useable in a productive environment. Therefore we could also discarded the classifiers nearest neighbour and random committee for the complete evaluation and report results only for UNICASE.

6 EVALUATION

In this section we evaluate and compare the different approaches of semi-automated task assignment. We evaluated the approaches using three different projects. All projects have used UNICASE to manage their work items as well as their system documentation. In section 6.1 we introduce the three projects and their specific characteristics. In section 6.2 we evaluate the approaches „state-based“. This means we took the last available project state and tried to classify all assignments post-mortem. This evaluation technique was also used in approaches such as [7]. Based on the results of the state-based evaluation we selected the best-working configurations and approaches and evaluated them history-based. We stepped through the operation-based history of the evaluation projects to the instant before an assignment was done. This state is not necessarily a revision from the history but can be a state in between two revisions. This is why we had to rely on the operation-based versioning of UNICASE for this purpose. On the given state we tried to predict this specific assignment post-mortem and compared the result with the assignment, which was actually done by the user. We claim this evaluation to be more realistic than the state-based as it measures the accuracy of the approach as if it had been used in practice during the project. Furthermore it shows how the approaches perform in different states of the project depending on the different size of existing data. As a general measure to assess performance we used the accuracy, i.e. the number of correctly classified developers divided by the total number of classified work items. This measure has the advantage of being very intuitive and easily comparable between different approaches and data sets. Other common measures such as precision or sensitivity are strongly dependent on the number of classes (number of developers) and their distribution and therefore would make it more difficult to interpret the results for our three projects.

6.1 Evaluation Projects

We have used three different projects as datasets for our evaluation. As a first dataset we used the repository of the UNICASE project itself, which has been hosted on UNICASE for nearly one year. The second project, DOLLI 2, was a large student project with an industrial partner and 26 participants over 6 month. The goal of DOLLI was the development of innovative solutions for facility management. The third application is an industrial application of UNICASE for the development of the browser game "Kings Tale" by Beople GmbH, where UNICASE has been used for over 6 months now. The following table shows the number of participants and relevant work items per project.

Table 1: Developer and work items per project

	UNICASE	DOLLI	Kings Tale
Developers	39	26	6
Assigned work items	1191	411	256
Linked work items	290	203	97

6.2 State-Based Evaluation

For the state-based evaluation we used the last existing project state. Based on this state we try to classify all existing work items and compare the result with the actually assigned person. In a first step (section 6.2.1) we evaluate the machine learning approaches. In a second step we evaluate the model-based approach.

6.2.1 Machine learning approaches

We have chosen different combinations of features as input of the application and applied the machine learning approaches described in section 4 as well as the model-based approach described in section 5. Our goal was to determine the approaches, configurations and feature-sets, which lead to the best results and re-evaluate those in the history-based evaluation (section 6.3). We started to compare different feature sets. As we expected the name of a work item to contain the most relevant information, we started the evaluation with this feature only. In a second and third run, we added the attribute description and the association ObjectOf. The size of the tf-idf matrix varied depending on the project and the number of selected features, e.g. for the UNICASE project, from 1,408 columns with only name considered to 4,950 columns with all possible features.

Table 2 shows the results of different feature sets for the support vector machine. In all evaluation projects, the addition of the features description and ObjectOf increased accuracy. The combination of all three attributes leads to the best results. As a conclusion we will use the complete feature set for further evaluation and comparison with other approaches.

Table 2: Different sets of features as input data

Name			
	UNICASE	DOLLI	Kings Tale
SVM	36.5% (±0.7%)	26.5% (±0.7)	38.9 (±1.4)
Name and description			
	UNICASE	DOLLI	Kings Tale
SVM	37.1% (±1.0)	26.9% (±1.0)	40.7% (±0.9
Name, description and ObjectOf			
	UNICASE	DOLLI	Kings Tale
SVM	38.0% (±0.5)	28.9% (±0.7)	43.4% (±1.7)

In the next step we applied the described machine-learning approaches using the best working feature set as input (Name, description and ObjectOf). As a base line we started with a constant classifier. This classifier always suggests the developer for assignment who has the most work items assigned. As you can see in table 3, we can confirm the findings of (Anvik et al. 2006), that SVM yields very good results. Random Committee performed quite badly in terms of accuracy and performance so we did not further evaluate them on all projects. The only competitive algorithm in terms of accuracy was Naïve Bayes, which was however worse on the Kings Tale project. As there was no significant difference between SVM and Naïve Bayes we chose SVM for further history-based evaluation due to the much better performance.

Table 3: Different machine learning approaches state-based

	UNICASE	DOLLI	Kings Tale
Constant	19,7%	9,0%	37,4%
SVM (LibLinear)	38.0% (±0.5)	28.9% (±0.7)	43.4% (±1.7)
Naïve Bayes	39.1% (±0.7)	29.7% (±0.9)	37.8% (±1.7)
Random Committee	23.2% (±0.2)		
Nearest Neighbor	6.9% (±0.1)		

6.2.2 Model-based approach

In the second step of the state-based evaluation we applied the model-based approach on the same data, which yields in surprisingly good results (see Table 4). The first row shows the accuracy of recommendations, when the model-based approach could be applied. The approach is only applicable to work items, which were linked to functional requirements. The number of work items the approach could be applied to is listed in Table 1. It is worth mentioning that once we also considered the second guess of the model-based approach and only linked work items, we achieved accuracies of 96.2% for the UNICASE, 78.7% for DOLLI and 94.7% for the Kings Tale project. For a fair overall comparison with the machine learning approaches, which are able to classify every work item we calculate the accuracy for all work items, including those without links, which could consequently not be predicted. Table 4 shows that the accuracy classifying all work items is even worse than the constant classifier. Therefore the model-based approach is only applicable for linked work items or in combination with other classifiers.

Table 4: Model-based approach

	UNICASE	DOLLI	Kings Tale
Linked work items	82,6%	58,1%	78,4%
All work items	19,9%	20,7%	29,3%

We have shown that the model-based approach can classify linked work items based on the ObjectOf reference. Therefore the approach basically mines, which developer has worked on which related parts of the system in the past (see section 5). One could claim that the machine learning approaches could also classify based on this information. Therefore we applied the SVM only on linked work items with all features and also only using the ObjectOf feature. The results (see Table 5) show that linked work items are better classified than non-linked. But even a restriction to only the feature ObjectOf did not lead to results as good as the model-based approach. Therefore we conclude to use the model-based approach whenever it is applicable and classify all other elements with SVM.

Table 5: Classification of linked work items

	UNICASE	DOLLI	Kings Tale
Constant	29,3%	18,3%	40,3%
SVM all features	53,9%	33,4%	50,2%
SVM only ObjectOf	49,7%	23,8%	49,2%

6.3 History-based Evaluation

In the second part of our evaluation we wanted to simulate the actual use case of assignment. The problem with the state-based evaluation is, that the system has actually more information at hand, as it would have had at the time, a work item was assigned. Consequently we simulated the actual assignment situation. Therefore we used the operation-based versioning of UNICASE in combination with an analyzer framework provided by UNICASE. This enables us to iterate over project states through time and exactly recreate the state before a single assignment was done. Note that this state must not necessarily and usually also does not conform to a certain revision from versioning but is rather an intermediate version between to revisions. By using the operation-based versioning of UNICASE we are able to recover these intermediate states and to apply our approaches on exactly that state. For the machine learning approach (SVM) we trained the specific approach based on that state. For the model-based approach we used the state to calculate the assignment recommendation. Then, we compared the result of the recommendation with the assignment, which was actually chosen by the user. For the history-based evaluation we selected the two best working approaches from the state-based evaluation, SVM and the model-based approach. We applied the model-based approach only on linked work items.

We applied SVM and the model-based approach on the UNICASE and the DOLLI project. The Kingsthale project did not capture operation-based history data and was therefore not part of the history-based evaluation. As expected the results for all approaches are worse than in the state-based evaluation (see Table 6). Still all applied approaches are better than the base line, the constant classifier. An exception is the model-based approach applied on the DOLLI project, which shows slightly better results in the history-based evaluation. We believe the reason for this is that the requirements model, i.e. the functional requirements, and the related work items were added continuously over the project runtime. Therefore at the states when the actual assignment was done, the model-based approach could calculate its recommendation based on a smaller, but more precise set of artifacts. Furthermore we can observe, that the results for the UNICASE project differ largely from the state-based evaluation compared to the DOLLI project. A possible explanation for this is the higher personal fluctuation in the UNICASE project. This fluctuation requires the approaches to predict assignments for developers with a sparse history in the project and is therefore much more difficult. In the state-based evaluation the fluctuation is hidden, because the approaches can use all work items of the specific developer no matter when he joined the project.

Table 6: History-based (aggregated accuracy) UC= UNICASE

	UC history	UC state	DOLLI history	DOLLI state
Const.	22%	19,7%	7%	9,0%
SVM	29%	38.0%	27%	28.9%
Model-based	75%	82,6%	61%	58,1%

Figure 3 and 4 show the accuracy over time for the UNICASE project and SVM and model-based approach, respectively. All presented charts show two lines. The first line (black) shows the aggregated accuracy over time. The second line (dotted black) shows the aggregated accuracy for last 50 (DOLLI) and 100 (UNICASE) revisions and therefore reveals short time trends. In the selected time frame, both approaches do not fluctuate significantly. This shows, that both approaches could be applied to a continuous project, were developers join and leave the project.

In contrast to the continuous UNICASE, we investigated the DOLLI project from the beginning

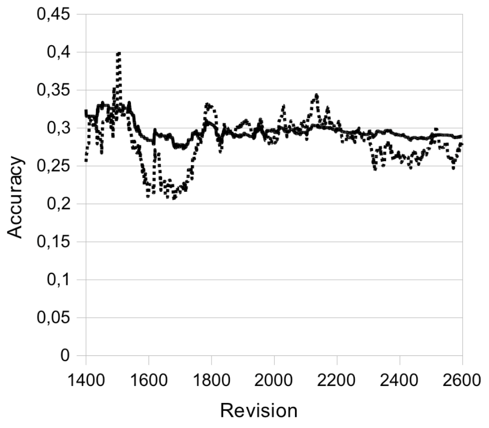

Figure 3: SVM - UNICASE

to the end (Figure 5 and 6) including project start-up and shutdown activities. We observe that SVM lacks in accuracy at the beginning, where new developers start to work on the project. For an efficient classification the SVM approach has to process a reasonable set of work items per developer. Therefore a high accuracy is only reached to the end of the project. A closer look at the accuracy of the model-based approach shows that it decreases at the end of the project. Starting from around revision 430 there has been a process change in the project as well as a reorganization of the functional requirements. This clearly affects the results of the model-based approach as it relies on functional requirements and their hierarchy. In contrast to the model-based approach, SVM seems to be quite stable against this type of change.

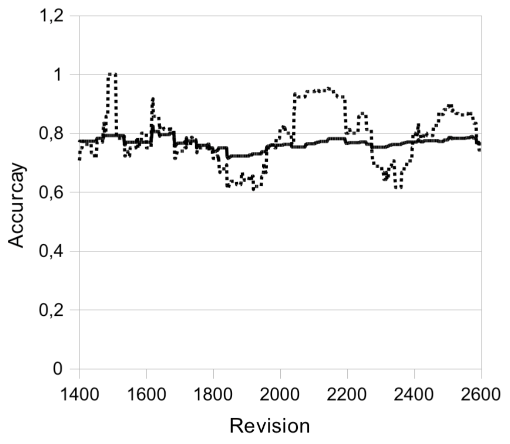

Figure 4: Model-based - UNICASE

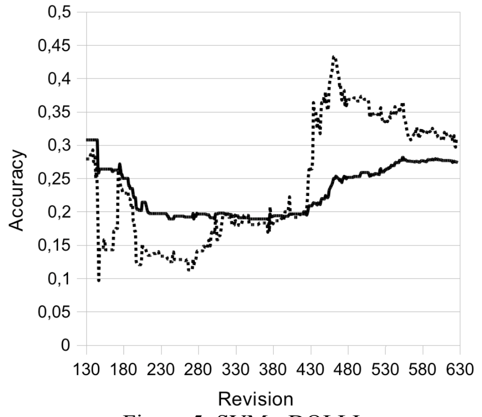

Figure 5: SVM - DOLLI

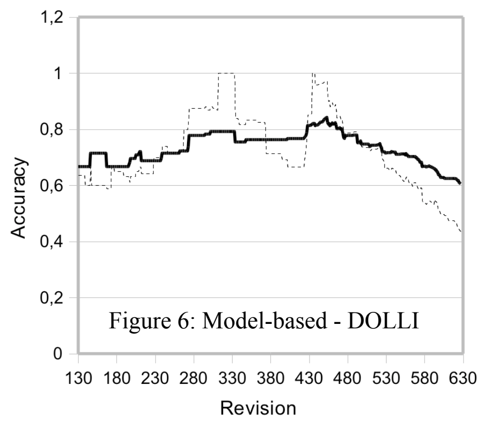

Figure 6: Model-based - DOLLI

7. CONCLUSION

We applied machine learning techniques as well as a novel model-based approach to semi-automatically assign different types of work items. We evaluated the different approaches on three existing projects. We could confirm the results from previous authors that the support vector machine (SVM) is an efficient solution to this classification task. The naïve Bayes classifier can lead to similar results, but the implementation we have used showed a worse performance in terms of computing time. The model-based approach is not applicable to all work items as it relies on structural information, which is not always available. However it showed the best results of all approaches whenever it was applicable.

The model-based approach relies on links from work items to functional requirements and is therefore not directly applicable in other scenarios than UNICASE, where these links do not exist. Although we believe that it can be transferred to other systems where similar information is provided. Bug trackers often allow to link bug reports to related components. Components on the other hand have relations to each other, just like the functional requirements in our context. An obvious shortcoming of the model-based approach is that it requires a triage by the affected part of the system no matter which model is used. On the one hand we believe, that it is easier for users to triage a work item by the affected part of the system rather than assign it, especially if they do not know the internal structure of a project. On the other hand if a project decides to use both, links to related system parts and links to assignees, the model-based approach can help with the creation of the latter.

In the second part of our evaluation, we tried to simulate the use case in a realistic assignment scenario. Therefore we applied the two best working approaches over the project history and predicted every assignment at exactly the state, when it was originally done. As a consequence all approaches can process less information than in the first part of the evaluation, which was based on the last project state. As expected the history-based evaluation leads to lower accuracies for all approaches. The model-based approach is less affected by this scenario than the SVM. A possible reason for that is that the model-based approach is not so much depending on the size of the existing data but more on its quality. This assumption is underlined by the behavior of the model-based approach during massive changes in the model, leading to lower results. In contrast to that, the SVM was not so sensible to changes in model, but more to fluctuations in the project staffing.

We conclude that the best solution would be a hybrid approach, i.e. a combination of the model-based approach and SVM. This would lead to high results for linked work items, but would also be able to deal with unlinked items.

REFERENCES

Čubranić, D., 2004. Automatic bug triage using text categorization. In *SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*. S. 92–97.

Anvik, J., 2006. Automating bug report assignment. In *Proceedings of the 28th international conference on Software engineering*. S. 940.

Anvik, J., Hiew, L. & Murphy, G.C., 2006. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*. Shanghai, China: ACM, S. 361-370. Available at: http://portal.acm.org/citation.cfm?id=1134285.1134336

Arndt, H., Bundschus, M. & Naegele, A., 2009. Towards a next-generation matrix library for Java. In *COMPSAC: International Computer Software and Applications Conference*.

Bruegge, B. u. a., 2009. Classification of tasks using machine learning. In *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*.

Bruegge, B. u. a., 2008. Unicase – an Ecosystem for Unified Software Engineering Research Tools. In *Workshop Distributed Software Development - Methods and Tools for Risk Management*. Third IEEE International Conference on Global Software Engineering, ICGSE. Bangalore, India, S. 12-17. Available at: http://www.outshore.de/Portals/0/Outshore/ICGSE_2008_Workshop_Proceedings.pdf.

Canfora, G. & Cerulo, L., How software repositories can help in resolving a new change request. *STEP 2005*, 99.

Fan, R.E. u. a., 2008. LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research*, 9, 1871–1874.

Freund, Y. & Schapire, R.E., 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1), 119–139.

Fritz, T., Murphy, G.C. & Hill, E., 2007. Does a programmer's activity indicate knowledge of code? In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. S. 350.

Haykin, S., 2008. *Neural networks: a comprehensive foundation*, Prentice Hall.

Helming, J. u. a., 2009. Integrating System Modeling with Project Management–a Case Study. In *International Computer Software and Applications Conference, COMPSAC 2009*. COMPSAC 2009.

Holger Arndt, I.I., The Java Data Mining Package–A Data Processing Library for Java.

Koegel, M., 2008. Towards software configuration management for unified models. In *Proceedings of the 2008 international workshop on Comparison and versioning of software models*. S. 19–24.

Mockus, A. & Herbsleb, J.D., 2002. Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering*. S. 503–512.

Raymond, E., 1999. The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12(3), 23–49.

Schuler, D. & Zimmermann, T., 2008. Mining usage expertise from version archives. In *Proceedings of the 2008 international working conference on Mining software repositories*. S. 121–124.

Sebastiani, F., 2002. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1), 1–47.

Sindhgatta, R., 2008. Identifying domain expertise of developers from source code. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. S. 981–989.

Witten, I.H. & Frank, E., 2002. Data mining: practical machine learning tools and techniques with Java implementations. *ACM SIGMOD Record*, 31(1), 76–77.

Yingbo, L., Jianmin, W. & Jiaguang, S., 2007. A machine learning approach to semi-automating workflow staff assignment. In *Proceedings of the 2007 ACM symposium on Applied computing*. S. 345.