Auto-tagging Emails with User Stories Using Project Context

S. M. Sohan, Michael M. Richter and Frank Maurer

Department of Computer Science
University of Calgary
2500 University Drive NW Calgary
AB, Canada, T2N 1N4
{smsohan,mrichter,frank.maurer}@ucalgary.ca

Abstract. In distributed agile teams, people often use email as a knowledge sharing tool to clarify the project requirements (aka user stories). Knowledge about the project included in these emails is easily lost when recipients leave the project or delete emails for various reasons. However, the knowledge contained in the emails may be needed for useful purposes, such as, re-engineering a software, changing vendor and so on. But, it is difficult to relate texts such as emails to certain topics because the relation is not explicit. In this paper, we present and evaluate a technique for automatically relating emails with user stories based on their text and context similarity. Agile project management tools can use this technique to automatically build a knowledge base that is otherwise costly to produce and maintain.

Key words: Distributed Agile, Collaboration, Software Documentation, Agile Tool

1 Introduction

In agile projects, requirements are commonly expressed using "user stories" in everyday language. These stories are not formalized and the meaning cannot be formally extracted. An example user story is as follows:
As a shopper, I want to pay online to checkout my shopping cart using MasterCard, Visa or Amex credit card from a secured web page only.
However, as the developers start developing the stories, they often consult with customers and other teammates to further clarify such user stories. Face-to-face communication is used as the principal communication medium between customers and developers in agile processes [1] [2] [3] [5]. In distributed teams, people often use emails where face-to-face communication is not an option [7]. For an example, the developer may send the following email to the customer to further clarify the example user story.
Subject: *Clarification required on online credit card payment*
Hi Bob:
Please clarify if the shoppers need to provide the security code of the credit card while doing checkout.

But, as someone leaves the team, it becomes hard to access that knowledge when needed later down the road. This paper presents and evaluates a solution that addresses this issue.

In this paper, we present a machine learning technique, Case Based Reasoning, to automatically relate emails with specific user stories. We do it by looking at the text, people and temporal similarity between emails and user stories. The results show that a well trained software can auto-tag emails with user stories. Also, we found that combining context with text similarity helps to find the related user stories with higher accuracy than using just text match alone.

We discuss the following topics in the next sections: Section 2 contains the problem statement and Section 3 contains related works in this area. Next, Section 4 presents our solution details and section 5 illustrates the solution with an example. We discuss the results and evaluation of our solution in Section 6 and finally, we summarize the paper at the conclusion.

2 The Problem

In collocated agile teams people mostly use face-to-face communication to share project related knowledge. But, in distributed agile teams, especially in hugely different time zones, people often use emails or text based communication for this purpose. As a side effect, distribution is beneficial for capturing knowledge for long term use in agile teams. However, even people in collocated teams often use emails to communicate with their customers and other stakeholders that are not located in the same office.

In software development projects, sometimes developers leave the team for various reasons. As said before, in distributed agile projects, important knowledge is often shared by emails. To transfer this knowledge for future use, one first needs to find the project related emails. Secondly, to build a usable knowledge-base, these emails should be related to specific user stories. If this is done, then another developer of the team will be able to easily find the necessary information when required.

From an end users' point of view, manually finding and relating the emails with user stories is a time-consuming and costly process. So, if a software can do this, then it may work as a knowledge-base even after someone leaves the team.

The core technical challenge in devising such a software solution is understanding the emails and relating them to specific user stories. The relation between an email and a user story is not explicit. The idea of using story ids or some kind of explicit markers in the emails also imposes the fact that one needs to look up the id in advance. Other approaches like modified email clients also imposes a behavioral change or a learning curve, which is often difficult for the people at the business end. We propose the auto-tagging to be a minimal change solution of the existing email process.

To find out the implicit relation between a free-format email and a use story, a software needs to handle similarity between two texts, which is a standard problem. However, pure text retrieval limits how accurate the assignment of

email and user stories can be. Using context information has the potential to increase this accuracy. So, a software needs to be able to combine both the text and context similarity for auto-tagging emails with user stories.

3 Related Work

From the agile software development perspective, customer collaboration and interaction among individuals are valued over following a strict plan or process [1]. El-Shinnawy et al. found that face-to-face communication is the richest form of communication [6]. But, globally distributed teams need to use asynchronous communication tools to make up for time zone difference and schedule conflicts. Layman et al. suggests that email offers a useful and prompt solution in such needs [7].

Distributed agile projects often use globally-available project management tools to facilitate awareness on everyday activities [7]. There are commercial and free web-based agile project management tools like VersionOne[8], ScrumPad [9], XPlanner [10] etc. Some of these tools offer message threads and project wiki for knowledge sharing. The use of wiki was also advised by Chau et al. [11] and Auer et al [12]. However, in a case study based on suggestions from [7], Korkala et al. found email to be the preferred medium for sharing knowledge in asynchronous communication [13]. But, none of the available tools can intelligently grab and auto-tag the emails with specific project artifacts. So, people are either forced to use the tool features (e.g. message threads or wiki) or the knowledge is no more available in a single shared place.

Previous works explored mining emails and software repositories. Cubranic et al. explored mining information from public forums with software artifacts and code repositories in their "Hipikat" project [14]. Before this, Berlin et al. implemented a group memory system called Teaminfo [15] that collected all emails from a given mail address and categorized the mails depending on predefined patterns. Our project follows the same approach for collecting the collaboration with the exception that we make use of the context information of an agile user story to auto-tag the emails. This context is formed by the developer, customer and iteration time frame of the user stories which are matched against the email meta-data. Using our solution, one can see all emails associated with a user story based on their text and context similarity.

However, for understanding emails and tagging with specific user stories a software has to deal with document indexing and retrieval. One approach to the retrieval process takes a purely document oriented view where one does not get information about the content of a document but rather on its existence. This is however of little interest for this paper. The second approach is of central interest in this paper, where one studies the content of a document using statistical methods. In the center of interest was the study of co-occurrence and its second order extensions. This goes back to H. Schuetze [18] and has been extended in various ways. All these extensions require a large amount of documents that are not typically available as user stories in software projects. However, the

problem addressed by this paper needs to utilize a domain specific similarity that combines both text matching and context matching.

4 Our Approach

4.1 Assumptions

Our solution is based on the following assumptions for auto-tagging emails with user stories.:

1. An email is potentially related to a user story when:
   - it is sent during the iteration time frame of the user story and
   - it is among people that are customer and/or developer of the user story and
   - there is a minimum degree of free text similarity between the two.
2. A web-based project management software is used to manage the distributed agile project.
3. The software knows about a project's team, user stories, iteration dates and scopes.
4. Each project in the project management software has its own email address.

4.2 The Project Context

The Project Context of an agile user story is defined by the following attributes:

1. **Temporal context:** User stories are developed in *iterations* defined by specific start and end dates. We define these time-boxes as the temporal context of the user stories.

2. **People context:** User stories are assigned to *team members* and owned by *customers*. We define these as the people context of the user stories.

We use this context information in combination with the text similarity to guess if an email is related to a user story.

4.3 Similarity Measure

Based on the above assumptions, we used Case Based Reasoning (CBR) to find the nearest user story of an email. In CBR, each case of the case-base is described using attribute-value pairs where attributes have defined types, for example, integer, symbol, free text etc. According to local-global principle, CBR uses two kinds of similarity measures, **i) local similarity** and **ii) global similarity** [16]. Local similarities between an example and a case are measured for each attribute individually. On the other hand, global similarity combines the local similarities.

In our solution, the case-base of the CBR system contained the user stories from multiple projects. An email was treated as a new example and the target

was to find the most related case (user story) from the case-base. So, we transformed the email attributes to map with the user story attributes as shown in Table 1.

Table 1. Mapping Between Email and User Story Attributes

Email	**User Story**	**Type**
Sender	Developer or Customer	Symbol
Recipient	Developer or Customer	Symbol
Email date	Iteration time frame	Date
Email text (subject + body)	Description	Free text

Source: *primary*

Using this mapping, we computed three local similarities between an email and a user story for three attributes. For the local similarity computation we used the following formulae:

$$S_{Date} = \begin{cases} 1 & \text{iteration start} \leq \text{email date} \leq \text{iteration end} \\ 0 & \text{if email date is within the buffer of the story's iteration} \\ -1 & \text{else} \end{cases} \quad (1)$$

$$S_{People} = \begin{cases} 1 & \text{both the developer and customer are present in email} \\ 0.5 & \text{either developer or customer is present in the email} \\ 0 & \text{else} \end{cases} \quad (2)$$

$$S_{Text} = [0, 1], \text{ Free text similarity score (See below)} \quad (3)$$

Here, computing text similarity is technically the most challenging one and a software needs to pay special attention to understand and find relevance between two texts. We used a statistical approach called OKAPI BM25 [19] text ranking formula for this purpose. This formula uses bag-of-words retrieval function and ranks the documents based on the frequency of query terms appearing in the documents. This formula and some of its newer variants are being used in document retrieval, such as by web-search engines.

Next, we computed the global similarity between an email and a user story using a weighted sum of the three local similarities as follows:

$$S_{Global} = (W_{Date} * S_{Date} + W_{People} * S_{People} + W_{Text} * S_{Text}) / (W_{Date} + W_{People} + W_{Text}) \quad (4)$$

where,

$$W_{Date} = \text{relative weight of date similarity} \quad (5)$$

$$W_{People} = \text{relative weight of people similarity} \quad (6)$$

$$W_{Text} = \text{relative weight of text similarity} \quad (7)$$

4.4 The Architecture

Next, we present the block diagram of the system in Figure 1. As shown in the figure, the core work is done by a web-based project management tool. This tool already knows about the user stories, their developers, customers and also iteration schedule. The following list explains the architecture as a workflow:

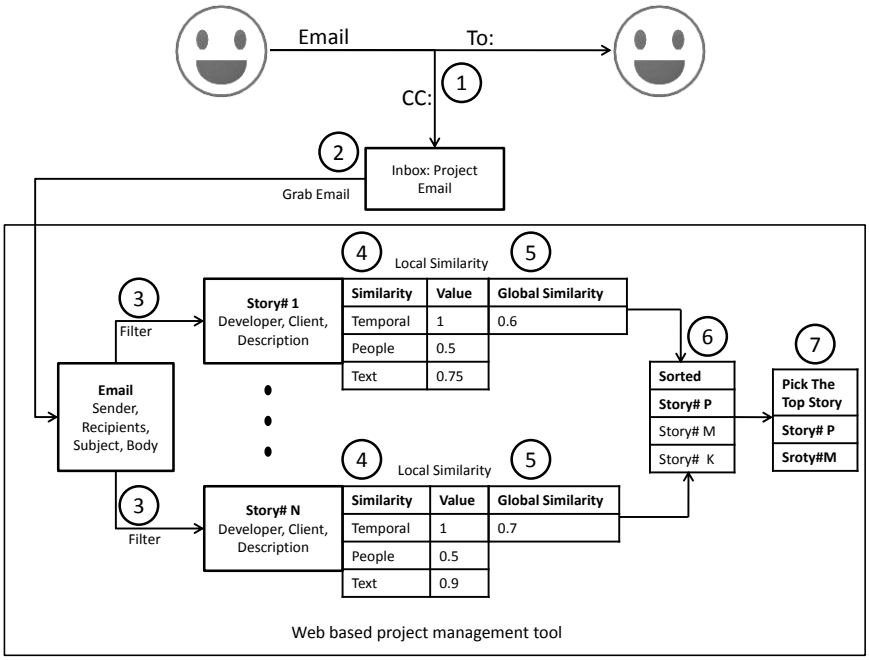

Fig. 1. The solution workflow steps

1. **Copy emails to project mail address:** Whenever an email is sent to someone about a project, the"project email" is added in the CC. This is the only change in the business process that needs to be implemented by the distributed team.
2. **Grab email:** The web-based project management software grabs all incoming emails at the project's email inbox using POP or IMAP.
3. **Filter:** Next, the search space is reduced by filtering out the user stories from far past or future compared to the email date.
4. **Compute local similarity:** The program computes three local similarity measures between the email and existing user stories using the equations (1), (2) and (3).

5. **Compute global similarity:** Next, the program computes the global similarity measures using equation (4).
6. **Sort:** After this, user stories are sorted according to the global similarity measures in descending order.
7. **Pick:** Finally, we pick the user stories having global similarity scores above a predefined threshold. As a result, the project management software can suggest multiple user stories that are a possibly discussed in a single email and make it available as a knowledge-base.

4.5 Learning Weights

To compute the global similarity between email and user stories using equation (4), our software learned the relative weights using a simple Reinforcement Learning [17] technique. The learning data was from four real world software projects as outlined in the data collection section later. First, we tagged emails with user stories exclusively for each attribute: data, people and text. The number of correct tagging using an attribute was normalized and used as the initial weight for that attribute. Next, the following algorithm was used to learn the relative weights:

```
date_weight     = initial_date_weigth
people_weight   = initial_people_weight
text_weight     = initial_text_weight

for all_training_emails do |email|
   result = find_most_similar_user_story(email)
   is_correct = result.guessed_story == email.actual_story
   is_date_similar = result.date_weight > date_threshold

   if (is_correct and is_date_similar) or
      (!is_correct and !is_date_similar) then
    date_weight = reward(date_weight)
   else
    date_weight = punish(date_weight)
   end
   #do the same for people and text similarity
   end
end
```

We used this algorithm to tune the relative weights following a simple reward-punishment approach. The weight of an attribute is rewarded with additional value if i) the attribute was similar and the email was related to the user story or ii) the attribute was not similar and the email was not related the user story. Otherwise, the weight for that attribute was punished with a decrease in value. This training helped in finding the relative importance for the attributes based on our training data.

4.6 CBR Implementation

We looked into several off-the-shelf tools for implementing the CBR system. But some tools that were capable of handling both text and non-text attributes were too complex to modify for our solution. On the other hand, the simpler ones lacked the support for free text attributes. So, we developed a simple CBR system from scratch.

We used Ruby programming language (v. 1.8.6) on Rails Framework (v. 2.3) to develop the CBR. We could also use an existing agile project management tool, but a custom-built one was easier for this experiment. The front-end was a web-based user interface for defining projects, teams, iterations and user stories. Once the data was provided to the system, it automatically guessed the potentially relevant user stories for an email.

The back-end of the CBR was implemented using a Microsoft SQL Server 2008 database. It was populated with the case-base. The similarity measures, as shown in equations (1-4), were then computed using custom Transact-SQL functions. For the free-text similarity, we used the built-in FREETEXTTABLE function of Microsoft SQL Server 2008 Advanced Services. This function ranks texts using the OKAPI BM25 formula.

The front-end ruby code used the back-end database functions to get the similarity measures. Then it guessed the relevant user stories for an email if the similarity score was above a threshold. However, if the guess was incorrect, the user interface provided an option to manually override the relation to a correct one. Having this overriding capability ensures the final verdict can always come from actual people using the system to rectify the mistakes in the automated guessing.

4.7 Evaluation Data

The size and composition of the data are summarized in Table 2.

Table 2. Training and evaluation Data

Project Name	**Description**	**Users**	**Iterations**	**It. Len. (days)**	**User Stories**	**Emails**
BI for Car Dealers	A decision support tool for vehicle dealers	7	11	14	70	213
ManyWheels	A web application for transporters and shippers	5	12	14	97	158
VarsityDays	A social networking application for school sports teams	5	15	14	88	46
MindAnd Market	A project collaboration tool	3	6	7	28	40
Total					**283**	**457**

Source: *www.ScrumPad.com*

The data for learning the weights and evaluating the implementation was obtained from four projects that used the agile project management tool called ScrumPad [9]. ScrumPad offers a forum for each project where people can collaborate using message threads on specific user stories. These linked messages helped us in both training and evaluating our implementation.
Three of the four projects, BI for Car Dealers, ManyWheels and VarsityDays, were developed by Code71, Inc. (www.Code71.com) and developers were from Bangladesh and USA on the same teams. On the other project, MindAndMarket, the team had 1 person from Bangladesh and 2 from Belgium and worked for a Belgian company called Belighted (www.Belighted.be).
The techniques presented in this paper can be applied to any data set that has similar characteristics. ScrumPad helped us to evaluate the technique using real industry data collected over a few years across different project. But other than this, our solution is independent of ScrumPad.

5 An Example

The following example illustrates the step-by-step email auto-tagging process. Here is a small list of four user stories from a project:

Table 3. User Stories

#	Iteration	Customer	Developer	Description
1	#1, Dec 14-25, 2008	C1	D1	As a/an 'VM User' I want to add/edit/view dealer account profile so that VM can earn revenue from charging a monthly subscription fee. Dealers will have three contacts: Billing, General Manager and Owner. Also, there will be a primary and secondary contact. More information is attached.
2	#1, Dec 14-25, 2008	C1	D2	As a/an 'VM Admin' I want to add/edit/view VM users. Administrator must be able to see username but not password VM Users will be one of inside rep and outside rep. Required fields and more information are attached...
3	#2, Dec 28, 2008-Jan 08, 2009	C1	D1	As a/an 'VM user' I want to load DMV data from Experian from a CSV/Excel file into VM database. Sample data attached.
4	Not Assigned	C1	D1	As a/an 'VM User' I want to activate and deactivate a dealer

Source: *www.ScrumPad.com*

While working on the project, developer D1 sent the following email to C1:

To: *C1*
From: *D1*
Date: *Dec 14, 2008*
Subject: *Initial Questions on* ***Dealer setup***
Body:
Please clarify the following questions regarding dealer setup data-
1. Is it ok to assume that the billing, GM and Owner contacts of a dealer will also contain username/passwords. Note that, the main and secondary contacts as well as the Used car manager contact contain user name/passwords.
2. What data should we collect for Physical Address and Mailing Address of a dealer? Is it free text? Or collected as fields street1, street2, city, state, zip, country?
3. Is it possible that for a dealer both an inside and outside salesman is assigned?
4. Apart from the name and pricing, is there any other field associated with the "program" main/platinum information?
5. What is meant by the "Location" field of the outside VM Rep?
6. Is it possible to provide us with a sample input data that will be used to set up an inside/outside VM user's commission? Please provide examples of default, bonus, retension and special commissions.
7. What are the required fields of all the dealer setup fields?
8. The Business Address is mentioned twice under the 5.6.1.5 - do we need to capture two business addresses?

Fig. 2. An example email

Now for this email, the similarity computation with the user stories yields the following results: Here, the similarity scores are computed based on Equations

Table 4. Similarity Scores

Story#	**Date Similarity**	**Date Weight**	**People Similarity**	**People Weight**	**Text Similarity**	**Text Weight**	**Global Similarity**
1	1	33	1	28	0.29	39	**0.72**
2	1	33	0.5	28	0.26	39	0.57
3	0	33	1	28	0.08	39	0.31
4	-1	33	1	28	0.25	39	0.05

Source: *Primary*

1-4 of Section 4.3. From the above table we see that Story #1 is the most similar story to the given email with a similarity score of 0.72. This is greater than the threshold of 0.58 (details discussed in the Evaluation section below) and thus the system auto-tags the email with Story #1. However, the other stories are not considered as related as those failed to reach the threshold similarity score. We ran the auto-tagging solution on the evaluation data in the same way as illustrated in this example and found the following results.

6 Results

The evaluation results showed that after some training, the system could correctly auto-tag 70% emails. We present our findings in detail in the next subsections.

6.1 Evaluation

We learned the relative weights by training the system with real life data from four distributed agile projects. For training and evaluation purpose, we partitioned these data into two parts, i) Training data and ii) Evaluation data. A total of 150 user stories and 250 emails from the four projects were used as training data. Also, we changed the training and evaluation data sets to reduce the impact of special cases. Table 3 shows the relative weights after training:

Table 5. Relative weights before and after training

Relative weights	Initial	After training
Date weight	34	33
People weight	16	28
Text weight	50	39

Source: *primary*

After learning these weights, we added the rest of the user stories into the case-base. Then, we presented the rest 207 unseen emails (from four projects) to the system in random order without specifying their relations with user stories and found the following results:

Table 6. Evaluation Results

1. Total number of evaluation emails: 207
2. Number of emails actually related to user stories: 200
3. Guesses using text and context similarity: 90% (187 out of 207)
4. Correct guesses using text and context similarity: 70% (144 out of 207)
5. Correct guesses using only text match: 47% (97 out of 207)

For the evaluation, we had 200 emails out of total 207 emails that were actually related to specific user stories. The rest 7 emails were related to their respective projects but not to any particular user stories. Such emails usually talked about general project related matters, such as architectural decision, milestones etc. The auto-tagging system was judged wrong if it made tagged any such generic project related emails to a user story. Here is an example at Figure 3:

Subject: *FTP Folder Convention*
Hi All:
At FTP we have the following folders:-
1. Experian - Only DMV data files should be dropped here.
2. Deploy - All Deployment related files should be dropped here.
The rest of the folders are for data sources as indicated in the names. Please make sure the folders are used correctly. It will be easy to manage in future.

Fig. 3. An example email that is not related to any specific user story

We set a **minimum threshold similarity score of 0.58** for making a guess. This cut-off point was found through trial and error. Using this cut-off the system made guesses for 90% or 187 out of 207 evaluation emails. A value above or below this threshold resulted in lower guesses or higher mistakes respectively. With this cut-off value, the system didn't make any guess for 10% emails because the most similar user story for the emails had a similarity score less than 0.58. So, in effect, this minimum threshold value was the optimum one for the given training data.

Out of the 187 guesses, the software rightly guessed the related user stories for 77% or 144 emails. Considering the total evaluation input of 207 emails, this is a 70% accuracy. It should be noted that the presence of the free text component leads to fuzzy match and a 100% accuracy may not be achieved as a result of this.

However, using only text matching, the system could correctly guess the user stories for 47% or 97 out of 207 emails. This is 23% lower compared to the case when both text and context similarities were used. So, we found that Case Based Reasoning helped in improving the accuracy of the system. Some of the user stories of a project often share a common vocabulary. As a result, when comparing with an email, the text similarity score of a user story from far past or future may be higher than the ones in the same time context of the email. Also, people assigned to the user stories as developers or customers are more likely to send emails on the stories compared to other individuals. So, adding temporal and people similarity scores helped in eliminating some of the outliers in terms of text matching alone.

6.2 Limitations

We recognize a few limitations of our solution as follows:

1. **Human subject study:** Although our results show that the email auto-tagging accuracy improves using a project context, it is not explored if this actually makes a difference in distributed agile teams. A human subject study may investigate this question.
2. **Possible bias in data:** Our implementation is trained and evaluated using data from multiple projects. But all these data are collected from a single source, www.ScrumPad.com, and hence there might be an unintentional bias

in the data. This limitation may be overcome by evaluating the system using data from various other sources.
3. **Size of data:** We recognize the fact that, the evaluation results could be statistically more significant with a larger size of the data.

6.3 Future Work

The following is a list of potential future extensions of our current implementation:-

1. **Evaluation with distributed agile team:** An actual evaluation with a distributed agile team is required to see the effectiveness of our email auto-tagging solution.
2. **Use of algorithms:** In future, we would like to try using other algorithms for learning relative weights and finding text similarity. The choice of other algorithms may produce better accuracy than the current results.
3. **Inclusion of more attributes:** It might be possible to improve the accuracy of the solution by adding more attributes. For example, an email might contain useful knowledge as attachments. Also, the subject line of an email may carry more specific knowledge than the body, but in the present solution both are treated as email text and no distinction is made in terms of importance. Adding such changes to the existing solution may produce better results.

7 Conclusions

In related work, we found that a significant portion of knowledge in distributed agile software development is actually shared through emails. In this paper, we described a solution to capture this information for reuse when people leave a project. The result indicates that it is possible to automatically build a reusable knowledge base from the emails.

Our implementation combined both text and context similarity. We discovered that this combination produced better results than using only free text matching. However, we anticipate that it is possible to extend this solution with using better algorithms and adding new features. Our ongoing work will focus on improving the accuracy of this solution by adding several candidate extensions.

References

1. Manifesto for Agile Software Development, web http://agilemanifesto.org
2. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley Professional, Reading, MA (2000)
3. Beedle, M. and Schwaber, K.: Agile Software Development with SCRUM. Prentice Hall, Englewood Cliffs, NJ (2001).

4. Cockburn, A.: Agile Software Development. Addison-Wesley, Reading, MA (2002).
5. Ambler, S., Jeffries, R.: Agile Modeling: Effective Practice for Extreme Programming and the Unified Process. Addison-Wesley, Reading, MA (2002).
6. El-Shinnawy, M., Markus, L.: Acceptance of Communication Media in Organizations: Richness or Features? IEEE Transactions on Professional Communication 41, pp. 242-253, IEE Press, New York (1998)
7. Layman, L., William, L, Damian, D. and Bures, H.: Essential Communication Practices for Extreme Programming in a Global Software Development Team, Information and Software Technology, vol. 48, Iss. 9, pp. 781-794 (2006)
8. Agile Project Management, Scrum, and Agile Development Tool, VersionOne, http://www.VersionOne.com
9. An Agile/Scrum Project Management Tool, ScrumPad, http://www.ScrumPad.com
10. XPlanner, http://www.XPlanner.org
11. Chau, T. and Maurer, F.: Knolwledge Sharing in Agile Software Teams. Logic Versus Approximation, vol. 3075, pp. 173-183. Springer, Heidelberg (2004)
12. Auer, S., Dietzold, S. and Riechert, T.: OntoWiki: A Tool for Social, Semantic Collaboration. The Semantic Web - ISWC 2006, vol. 4273, pp. 736-749 (2006).
13. Korkala, M., Pikkarainen, M. and Conboy, K.: Distributed Agile Development: A Case Study of Customer Communication Challenges. Agile Processes in Software Engineering and Extreme Programming, vol. 31, pp. 161-167. Springer, Heidelberg (2009)
14. Cubranic, D., Murphy, G. C., Singer, J., Booth, K. S.: Hipikat: A Project Memory For Software Development, Software Engineering, IEEE Transactions on, vol.31, no.6, pp. 446-465, IEEE Press, New York (2005)
15. Berlin, L.M., Jeffries, R., ODay, V.L., Paepcke, A., and Wharton, C.: Where Did You Put It? Issues in the Design and Use of a Group Memory, In: Proceeding of SIGCHI Conference. Human Factors in Computing Systems, pp. 23-30 (1993)
16. Richter, M. M.: Knowledge Containers. In Ian Watson, editor, Readings in Case-Based Reasoning. Morgan Kaufmann Publishers (2003).
17. Sutton, R. S., Andrew, G. B.: Reinforcement Learning: An Introduction. MIT Press (1998)
18. Schuetze, H. and Pedersen, J.O.: Information Retrieval Based on Word Senses. In: Proceedings of the Symposium on Document Analysis and Information Retrieval, vol. 4, pp. 161-175 (1995).
19. Robertson, S., Walker, S., Jones, S., Hancock-Beaulieu, M. and Gatford, M.: Okapi at TREC-3. In: Proceedings of the Third Text REtrieval Conference (TREC 1994). Gaithersburg, USA (1994).
20. Robertson, S., Walker, S. and Hancock-Beaulieu, M.: Okapi at TREC-7, In: Proceedings of the Seventh Text REtrieval Conference, Gaithersburg, USA (1998).