WikiDev 2.0: Discovering Clusters of Related Team Artifacts

Ken Bauer, Marios Fokaefs, Brendan Tansey, and Eleni Stroulia

Department of Computing Science
University of Alberta
{kwbauer,fokaefs,tansey,stroulia}@cs.ualberta.ca

Abstract

Most software development today is a team activity. Project team members collaboratively work on the tasks necessary to accomplish the various project milestones. The work is usually asynchronous, not orchestrated by any explicit workflow, sometimes geographically distributed, and involves the use of a variety of tools, which do not always interoperate. Version-control repositories are essential in supporting this collaboration but cannot satisfactorily address the problem of traceability of interdependencies among the artifacts produced by the individual tools. In the *WikiDev 2.0* collaboration tool, we propose to address these problems by adopting a wiki as the central platform in which to integrate information about the various artifacts of interest, to cluster this information in clusters of relevant artifacts, and to present views on this information that cut across the individual tool boundaries. In this paper, we discuss the central clustering algorithm in *WikiDev 2.0* and we evaluate its effectiveness with a case study.

1 Introduction

Most software development today is collaborative. The complexity of most software projects implies the need for teamwork. Project teams are sometimes geographically distributed and work in a distributed, asynchronous manner. Team members take the responsibility for accomplishing individual tasks (sometimes through centralized processes and others through their own initiative), use development tools appropriate for their individual tasks (and not always adopted by the whole team, especially when the team is distributed), communicate through a variety of means (including formal documentation od tasks and code and informal email and messaging), and share work products through version-control tools. On a regular basis, as milestones are reached, the various work products are integrated and tested as a whole.

There is substantial empirical evidence for the advantages behind collaborative software development. However, there is also evidence that inefficient communication, ineffective administration, and inconsistencies in shared work products are the major challenges that the software teams face [5, 13]. The problem is further exacerbated by the abundance of tool support for individual lifecycle activities, which are not always universally adopted by the whole team, either because individual team members do not need some specialized tools for their tasks or because they may prefer alternatives. These challenges sometimes result in a lack of project-status awareness by individual team members, who, although very familiar with their own tasks, end up having little knowledge about the overall project. An easily accessible, web-based tool that would provide up-to-date information of interest about the current status of all project work products and their interdependencies would greatly alleviate this concern.

The community, recognizing this problem, is working towards several solutions to address it. Let us mention two well known and representative examples. IBM's Jazz [12] is "a scalable, extensible team collaboration platform for integrating work across the phases of the development lifecycle", but can be quite complex to learn and use effectively for smaller teams. On the other end of the spectrum, DrProject [18] "is a web-based project management portal that integrates revision control, issue tracking, mailing lists, a wiki, and other tools that software development teams need to succeed". Our own earlier work on WikiDev [14] and CVSChecker [15] was conceptually similar to DrProject and has built the foundation for the new tool discussed here, *WikiDev 2.0*.

There are two fundamental hypotheses underlying the design of *WikiDev 2.0*. First, a wiki is an easy to learn and easy to use collaboration platform, that enables transparent sharing of information while keeping track of its evolution history and its various contributors. Second, there are many dependency relations among software artifacts that are not visible within the confines of any single tool. Thus, an extendible relation-extraction mechanism is necessary for maintaining awareness, as new tools are being adopted by the software team. This mechanism requires a systematic method for accessing information from one tool and cross referencing it with information originating from other tools. In developing *WikiDev 2.0*, we have decided to build such a mechanism for integrating information from different sources in our own extension of MediaWiki, in order to take advantage of the MediaWiki collaboration metaphor.

The rest of this paper is organized as follows. Section 2 reviews related research and tools. Section 3 describes the *WikiDev 2.0* architecture. Section 4 discusses our preliminary results, and Section 5 draws some initial conclusions.

2 Related Work

There have recently been several projects and tools aimed at extracting interesting information from the data captured in a CVS repository. On one hand, tools like softChange [9], Cvsplot [24], Bloof [16], CVSAnalY [19], Beagle [10], ROSE [30], JDEvAn [28], as well as work by Gall et al. [8], Hassan and Holt [11] and Bevan [3] are designed to analyze the evolution history of the software residing in a version-control repository. These tools have been applied to a variety of software projects to analyze (a) the volatility (or entropy) of the project as an indication of the overall amount of changes occurring in its modules [3, 11], (b) the various distinct phases of the software evolution history [9, 28], (c) the implicit coupling among software modules that coevolve [8, 28, 30] and (d) the internal code restructurings that may have occurred during the project lifecycle [10, 28]. On the other hand, CVSMonitor [4] and CodeStriker [23] are designed to support goal-oriented navigation of the source code.

Yet other tools focus on recommending useful software artifacts to developers. CodeBroker [29] relies heavily on consistent code documentation since it recommends a component (method) based on developer comments and method signature. Hipikat [25] integrates a large number of software artifacts (i.e., source code, documentation, bug reports, e-mail, newsgroup articles, and version information) and recommends them based on the task context of the developer when they request help from Hipikat. Hipikat and *WikiDev 2.0* are closely related in terms of the overall problem they are trying to address, in that they both aim to show developers software artifacts of interest. There are three interesting differences between them. First, the software architecture of *WikiDev 2.0* is meant to be easier to extend as we are developing standard APIs for providing information to the system that could be used from a variety of development tools. Second, the method we are using for artifact recommendation is based on clustering, whereas Hipikat relies on text information retrieval. Finally, by adopting Annoki, our own extension to MediaWiki, as the overall platform, we expect to enable more informal interactions to be part of the project wiki, thus enabling us to better understand the nature of the team's collaboration process. This last feature, namely the adoption of a wiki in support of software collaboration, is also shared by WikiDev [14][1] and DrProject [18]. All of these tools provide a web-based environment (in many cases based on wiki pages).

More recently, several tools have been proposed that employ social networks based on software artifacts in order to enhance the communication and the awareness within a team of developers. Codebook [2] is a social networking web service that allows developers to become "friends" with artifacts and other users of their interest. Using graphs based on the Bridge [26] framework, it includes information about developers and group of developers, code artifacts (source files, types, methods), bugs and messages. The graphs of the project activity are then presented to the users to facilitate them. The tool can also suggest new "friends" to the users based on their activity. Similarly to Facebook, Codebook relies a lot upon its users and their input. Unlike our method, it seldom captures any implicit relationships of significant interest. Tesseract [22] is a monitoring tool that employs visualizations and social networking techniques to present the activity of the project. It identifies relationship between developers based on their communications. Relationships between code artifacts are inferred based on the activity of the version-control repository. For example, if two files were submitted at the same time, it is assumed that there might be a correlation between them. Finally, the tool monitors the progress of open issues. The fundamental difference between Tesseract and *WikiDev 2.0* is that the latter identifies relationships between artifacts based on textual information while the former identifies relationships based on the users' activities.

[1] *WikiDev 2.0* is a conceptual descendant of WikiDev, although our current work does not share any actual implementation with the original WikiDev.

Finally, Wolf et al. [27] show that the communication structure with the development team impacts the success (or failure) of build results. They show that builds that involve developers that communicate a lot with each other are less likely to fail. Although the authors link the impact of communications artifacts across teams to the results of code artifacts, their goal differs from our goal of discovering patterns among all artifacts in the project and how that impacts the success of an individual team.

3 *WikiDev 2.0* Architecture and Features

WikiDev 2.0 is built on top of Annoki, our own extension to MediaWiki. Annoki provides a set of useful extensions to the original MediaWiki including the following (of interest to *WikiDev 2.0*):

- Annoki provides a namespace-based access-control mechanism, supporting the sharing of information openly or within a group or restricting it to a single individual. Individuals can be members of multiple groups and individual pages can be shared across groups, thus resulting in a quite flexible mechanism.

- In addition to the MediaWiki Diff mechanism, Annoki provides a contribution analysis and visualization based on sentence ownership and structure evolution.

- Annoki supports the development of templates, which users can adopt to make explicit the logical structure of their content. Template-based pages can be edited through a form-like interface, in addition to the regular wiki-style editing.

- Finally, Annoki has several built-in visualizations for communicating the contribution of users to the content of a page, the logical structure of several built-in templates, and the organization of the overall wiki server content.

Annoki provides the metaphor for the communication among the software team members through wiki pages, their evolution analysis and their visualizations. In addition it constitutes the central platform for *WikiDev 2.0*'s information integration mechanism.

Let us now review the *WikiDev 2.0* information-integration mechanism. If the tool of interest relies on its own database (or a specially structured file system), a data-level integration mechanism is applicable: active triggers are introduced in the external tool database (or daemon processes monitoring the file-system changes) to report information of interest to *WikiDev 2.0* as events occur to update the tool database (or changes happen in the filesystem). If the external tool is developed in a language with aspect-oriented support, a procedure-level integration is possible: aspects can be weaved though special workflows to report information of interest to *WikiDev 2.0*. These bottom-up information feeds populate the *WikiDev 2.0* database and through special-purpose templates are immediately accessible to the *WikiDev 2.0* users though the web.

The combination of Annoki and the information-integration mechanism provide a web-based tool for information sharing across the whole team. The core of *WikiDev 2.0*, however, is its relation-extraction mechanism, discussed in detail in the subsections below.

The first step in analysis involves parsing of all the textual information associated with the input information feeds to recognize mentions of team members (their names, nicknames, or IDs) and software artifacts (classes, methods, and interfaces). These recognized references introduce the explicit relations among existing work products. A subsequent step calculates the implicit relations based on triangular inequality thus providing us with some new insights about hidden dependencies. Using this information, we believe that the team members, as well as the project-management team, should be able to enhance their knowledge about the internal processes of the project they work on, by being able to answer questions like *who works on what artifact currently*, *who has discussed a specific artifact that should potentially be consulted about changes to it*, and *how might a member's own work affect other people's work?*

3.1 Dataset Preprocessing

The dataset-preprocessing step enables us to reason about our results and identify the intuition behind the relationships. Considering relationships between specific pairs of artifacts help us identify meaningful connections. Therefore, we divide our dataset of artifacts into two broad categories: (a) changesets, i.e., collections of files commited together in the repository and the associated comment, and (b) secondary (or communication) artifacts which include tickets, messages and wiki pages. The reason for this division is that we consider the source code to be the most important artifact of a project and almost every communication during the development usually refers to the source code. Thus, a pair consisting of a secondary artifact and a changeset indicates that the former has triggered or contributed to a source code change.

3.2 Text Analysis

The relationships are identified based on the text associated with each artifact. If a secondary artifact references the name of a source class, then a direct connection is identified between the artifact and the changeset that last changed the class. We also assign weights on the connections to pick out the strong ones. The weights are based on the TF/IDF (term frequency/inverse document frequency) [21] measure using the following formula:

$$TF/IDF_{i,j} = TF_{i,j} * \log \frac{|D|}{\{d_j : t_i \in d_j\}} \quad (1)$$

where i is the term, j is the artifact and $|D|$ is the total number of artifacts in the dataset.

This measure gives us a vector with the normalized frequencies of the terms in each artifact. This way we will be able to determine the predominant terms in each artifact and later using this information to determine the existence as well as the strength of similarities between artifacts.

We first parse the text of each artifact and produce a word list which is clean of any punctuation and whitespace. Then, we remove stop words and stem the rest of the list using Porter's stemmer [17]. Porter's algorithm is essentially a suffix stripping algorithm. It might not always give perfectly accurate results—for example it will group *university* and *universal* under *univers*—but we have found it generally sufficient. Finally, for each word, we first count its frequency in the artifact's text, then count the number of artifacts of the total dataset in which this word is mentioned, and calculate the TF/IDF measure. We cannot afford to remove the rare terms because in some artifacts the text consists of only a couple of sentences, meaning that the terms rarely occur more than once.

3.3 Clustering

The clustering algorithm we use is the hierarchical agglomerative clustering algorithm. First, each data point (in our case each artifact) is assigned to a single cluster. Then, the clusters are merged according to the linkage criterion. The linkage criterion is actually the distance between clusters of data objects as a function of the pairwise distances between data objects. Some commonly used linkage criteria are the *complete linkage* (based on the maximum distance between the data objects of two clusters), the *average linkage* (based on the average distance) and the *single linkage* (based on the minimum distance). We chose to use the last criterion in our implementation. The merging step of the algorithm is repeated until all the data objects are clustered in the same single cluster.

The final output of the algorithm is a hierarchy of clusters. To obtain the artifact clusters, one must define a distance threshold value as a single cut-off value. The final clusters are those contained in the last level of hierarchy just below the threshold value.

A problem with the agglomerative algorithm is that frequently a single distance threshold does not yield the optimal results. For this reason, our method is not restricted to only one value. We apply the algorithm for different threshold values ranging from 0.1 to 0.9 and the resulting clusters are presented to the user, who can use the best ones. However, our experimentations have shown that the most meaningful results are produced using a threshold value between 0.5 and 0.7. Another advantage of the distance threshold is that it can be conceptualized to more verbal discrete values (*e.g.* TIGHT/LESS TIGHT), so that the user can select the appropriate value.

We did not choose a partitioning algorithm, such as k-means, because it would be harder to select the number of clusters rather than a distance threshold as the indicator of how tight the clusters should be. This is because the spatial structure of our datasets is relatively unknown and and we cannot intuitively predefine a fixed number of clusters and neither can the users since they don't have any insight on this kind of information concerning the data. Moreover, the datasets that we examined came from student projects and contain approximately 500 to 1000 artifacts. Therefore, the calculation of a proper k value (*i.e.* the number of clusters) can dramatically deteriorate the performance of our method. Finally, partitioning algorithms are known not to be robust against noise. By noise, we mean data objects that are too far from the body of the dataset to be clustered to any of the clusters. On the other hand, choosing the appropriate distance threshold will help the agglomerative algorithm to deal with noise efficiently.

We chose an agglomerative algorithm over a density-based one because the former requires the definition of two threshold values. Density based algorithms, like DBSCAN [7] require the ε-neighborhood parameter, which defines a radius of a point around which a dense subgroup (not a cluster) can be defined, and the *MinPts* parameter that indicates the minimum number of data objects that need to reside in a subgroup so that it can be considered as a dense one. While we can calculate an ε-neighborhood value in a similar way to how we define the distance threshold, it is not as easy to predefine a *MinPts* value and there is no apparent reason to restrict our method that way.

The distance metric we use is based on the cosine distance [20]. Supposing that we need to calculate the cosine distance between artifacts A and B. The first artifact contains the terms $t1$, $t2$ and $t3$ and the second the terms $t1$, $t2$ and $t4$. The associated term frequency vectors would be:

$$a = \{f_{1a}, f_{2a}, f_{3a}, 0\} \quad (2)$$

$$b = \{f_{1b}, f_{2b}, 0, f_{4b}\} \quad (3)$$

where f_{ij} is the TF/IDF frequency of term i in artifact j. The cosine distance is then calculated as follows:

$$d_{ab} = 1 - \frac{a \cdot b}{||a||||b||} \quad (4)$$

As we have already discussed, direct relationships are primarily identified between secondary or communication artifacts (tickets, wikis and messages) and changesets. If the secondary artifact contains a term that matches the name of a class that was changed in the changeset then a relationship is identified and the distance between the artifact and the changeset is the cosine distance of the respective term frequency vectors.

Having identified the direct relationships, the next step is to capture the implicit relationships. A way to achieve that is by connecting all the entities with each other creating, as a result, a complete graph. To that end, we employ the triangular inequality, which holds since cosine distance is a metric. According to this concept, the distance between two entities cannot be greater than the sum of distances between the two entities and a common neighbor of theirs. Therefore, we try to find common neighbors (*i.e.,* directly related artifacts) for every pair of artifacts. If the sum of distances between the artifacts and their common neighbor is less than the distance between the artifacts, then the sum substitutes the former distance. In the end, the distance of the two artifacts is the sum of distances between them and their closest common neighbor.

3.4 Reasoning

The desire here is that a cluster defines a set of closely related artifacts across the different types of artifacts (currently code changesets, wiki changes, tickets and messages) within the dataset under evaluation. Smaller clusters of closely related artifacts will allow us to guide developers by making them aware how changes to artifacts impact other artifacts in the system under development. We hope to have a small collection of closely related artifacts to each artifact in the system. Too many artifacts being related would just create overload while too few may only include the obviously related artifacts.

This leads us to our choice of slicing the dataset by development week before applying the clustering. At first we attempted clustering across the entire project for each team and our results ended with only a few clusters of large size. By slicing the dataset into weeks, and choosing an appropriate threshold value by examination of the clustering using threshold values from 0.1 to 0.9, we came up with the clusters discussed in the following section.

4 Results and Discussion

In our evaluation of our artifact clustering algorithm, we have been considering the following questions.

1. Does a cluster really contain data that is "close", i.e., related to some specific development task?
2. Are artifacts that should be in a cluster missing from the cluster? This question refers to the "recall" quality of the algorithm.
3. Are there artifacts in a cluster that shouldn't be there? Conversely, this question refers to the algorithm's "precision" in retrieving relevant data.

For this paper, we chose to analyze the data from a team of four developers from our undergraduate software engineering class. The data includes subversion changesets, tickets, wiki edits and email messages from each team. A total of nine weeks of development was involved although the initial weeks produced little code and hence few or no clusters.

As discussed in Section 3, we ran our clustering algorithm with thresholds in increments of 0.1 from 0.1 to 0.9. We found that a threshold of 0.7 is the optimal in terms of finding enough clusters while avoiding false positives. The results of the clustering are presented in Tables 1, 2, and 3. Once the clusters were produced, the course instructor (and one of the paper's coauthors) analyzed the contents of each cluster to answer the three questions above for each cluster. The table columns labeled "Close", "Missing", and "Extra" correspond to these three questions. A checkmark in the "Close" column signifies that the artifacts in the cluster are related to some specific development task, a checkmark in "Missing" signals that the instructor felt some other artifact should have been included in the cluster, and a checkmark in "Extra" means the instructor determined there were artifacts included in the cluster that are not related to the others. The anonymized author of each artifact and notes from the instructor are included for each cluster.

We were also interested in the number of elements that were part of a cluster in terms of percentage of total artifacts in each week under analysis. This is presented in Table 4. Note that there are no clusters identified in weeks one and two, as very few teams started actually committing code before the third week of the project.

Week	#Artifacts in clusters	#Artifacts total	%
1	0	17	0
2	0	21	0
3	11	140	7.9
4	12	65	18.5
5	12	149	8.0
6	10	156	6.4
7	12	128	9.4
8	4	92	4.3
9	28	477	5.9
Total	**89**	**1245**	**7.1**

Table 4: Proportion of Artifacts in Clusters

We see an initial cluster in the first week of actual development (Week 3). This cluster of eleven artifacts consists of four subversion changesets, six tickets and one wiki page edit. As noted on Table 1, all of these artifacts are related due to the fact they are about the unit testing of the project. Note also that all items were associated with the same developer.

We can see an example of a false positive match in (Week4, Cluster1) involving a subversion changeset (CS26) and a ticket (Ticket41). The cluster was detected here solely on the matching of the string "details" in both artifacts. The subversion commit comment was "See ticket 37 for details" while the ticket description was "See UI diagram for details". Note that the subversion commit has a developer defined link to another ticket that is not included in this cluster, because it was introduced in the week prior; thus, our method slicing the data set in one-week increments introduces artificial boundaries which may disconnect relevant artifacts from each other. In fact, we have identified that all clusters with a checkmark in "Missing" are a result of this slicing. The consequences of this slicing are discussed below.

An example of an interesting cluster is shown in (Week9, Cluster10). The text of an email to the team mailing list (Message82) describes a bug that was found but the developer had difficulty replicating. The two subversion changesets (CS511 and CS587) committed by a different developer are both related to tickets that were created by this second developer. Again these tickets are not in the cluster due to the slicing by dates. What is interesting is that the email was sent by a developer that did not realize the bug was already part of tickets in the system.

Of the thirty-two clusters identified, only two were not considered to be "close" in terms of being related to a specific development task. Thirteen of the clusters have been identifed as having artifacts "missing". Finally, three of the clusters were identified as having "extra" artifacts meaning the algorithm was false identifying some artifacts as being related to others when the instructor determined they were not in fact related. Although in this paper we are discussing the results from a single team we are encouraged by the outcomes of our clustering algorithm and we are in the process of analyzing the artifact clusters of the other ten teams of the same course. As we mentioned before, we previously were attempting to cluster across the entire timeframe of the project and this lead to very few clusters of large size. The move to do an initial slicing of the data into weekly sets generated the results we see in this paper. A total of thirty-two clusters were identified across nine weeks of which two weeks have no clusters due to project spin-up.

From the thirty-two clusters, only two had no real relation among the artifacts due to mismatches of common text. Cluster one in week four consisted of a changeset and ticket by the same author that included the word “details”. Cluster one in week eight was also a changeset and ticket pair by the same author that had a false match on the word “node”. We expect to encounter some false positive matches in our clustering and are looking for solutions to avoid these matches or include tagging support at a user interface level in the Annoki presentation layer to mark artifacts as not being related in order to preclude them from matching in future.

Of the three cases of “extra” artifacts being included in clusters, two are in fact already identified by the preceeding discussion where no close relationship was found. The third case is cluster one in week seven, where two (CS286 and Ticket111) of the three artifacts in the cluster are related and a third artifact (CS228) was included in the cluster due to a false match on the text “search” and both changesets include changes to the file SearchResults.java.

Of more concern are the thirteen cases of “missing” artifacts from clusters. We found there are two factors causing this problem. All cases of missing artifacts stem from the developer noting the ticket number in the subversion commit message. Normally this would cause the ticket to be included in the cluster but the slicing of the dataset into weeks eliminates these tickets from consideration when the ticket was created (opened) previous to the week being examined. There are, at least, three approaches to this problem. The first involves including the changes to tickets over time in the clustering algorithm. Currently only the creation of the ticket is analyzed which includes the text in the subject and description part of the ticket which are dated on the timestamp of ticket creation. We now have the full history of tickets in the database and including those artifacts (each ticket change being labeled an artifact) should push ticket changes related to other artifacts into the cluster based on their time proximity. The second approach we examine is to force inclusion of these artifacts that are explicitly linked in the text of artifacts as tickets tend to be referenced in subversion commits. Finally, another method would be to employ a sliding window, as opposed to a calendar-week window, and reject all clusters that are subsumed by another cluster found in a neighboring week.

Part of the training of the undergraduates in the project course is to encourage “good” practices such as “never writing code unless the task has been already defined as a ticket” and “always referencing the related ticket when committing code”. As can be expected, some students follow these rules better than others. More analysis could be done to see how following these rules affects the ability of our clustering algorithm to find relations between artifacts. Despite the fact that subversion commits are pretty equal across the members of the team (10.1%, 33.7%, 27.8% and 28.4% for User1, User2, User3 and User4 respectively) we notice that User2 and User3 appear more often than User1 and User4 in the artifacts of the clusters (3, 39, 31 and 16 times for User1, User2, User3 and User4 respectively).

Other effects on our clustering that we are in the process of examining are the styles of using the tools that generate the artifacts. We witnessed that some teams used the ticketing system to organize the team while others tended to use wiki pages more often for the same goal.

5 Conclusion

In this paper, we presented a method for retrieving relationships between software engineering artifacts. The method is part of the *WikiDev 2.0* framework. It is based on TF/IDF similarity and cosine distance between artifacts.

Our tool identified a significant number of clusters of artifacts from various sources in a typical software engineering undergraduate project. We believe the identification and visualization of these clustered relations can give stakeholders (be they the students, teaching assistants or the instructor) valuable insight about the status of the project at any given point in time. To that end, we have developed several visualizations in *WikiDev 2.0* and as they are all built as extensions of wiki pages, they can be discussed by the team managers and developers. In this paper, we have focused our discussion to the relation-extraction mechanism only, since it is essential in identifying depenendencies between the artifacts "imported" in *WikiDev 2.0* from the tools used by the software-team members. However, our choice of Annoki – essentially MediaWiki – as the underlying platform enables the straightforward access, visualization and editing of this information.

There is still substantial room for improvement in our relation-extraction method, such as including the history of ticket changes and including analysis of data from other sources. We recently started collecting logs of internet relay chat channels (IRC) for each team and have yet to bring that data into our analysis. We also intend to measure the effects of having (visualizations of) this information on the performance of the teams.

In the technical part, our intentions include the substitution of the text analysis part of our method with TAPoR [1] in order to employ a more established, efficient and rich toolkit. Furthermore, we are testing our algorithm using different time windows (*e.g.* every two or three weeks) to observe their impact on our results. Finally, we are considering trying a fuzzy partitioning algorithm like FCM (Fuzzy C-means) [6] to study the effect of overlapping clusters (*i.e.* an artifact belonging to more than one cluster) on our results.

Acknowledgements

The authors would like to acknowledge the generous support of iCORE, NSERC, and IBM. Moreover, we would also like to thank the instructors, teaching assistants and students of CMPUT301 for their support of our research efforts. This research would not have been possible without their support.

About the Authors

Ken Bauer is a Ph.D. candidate at the University of Alberta. His research interests include distributed collaborative software development and software engineering education. He is also a professor in Computing Science at ITESM in Guadalajara.

Marios Fokaefs is a M.Sc. student at the University of Alberta. His research interests include software reengineering of object-oriented and service-oriented systems, in particular identification of patterns and refactoring opportunities. He has also worked on projects on the application of data mining techniques on software systems and social networks.

Brendan Tansey is a research associate and recent M.Sc. graduate from the Department of Computing Science at the University of Alberta. His research interests include data mining information from team-based interaction repositories, wiki-based collaboration, the economics of software services, and modeling business concerns of offshore software development.

Eleni Stroulia is a Professor in Computing Science at the University of Alberta, as well as the iCORE Industry Research Chair in Service Systems Management, General Chair of the 2009 International Conference on Software Maintenance, and an editor for the Computational Intelligence Journal. Her reserch interests span the areas of software design and analysis, web-based system development, and service-oriented systems.

References

[1] Tapor. `http://portal.tapor.ca/`.

[2] Andrew Begel and Robert DeLine. Codebook: Social networking over code. In *ICSE COMPANION '07: Companion to the proceedings of the 31st International Conference on Software Engineering*, Vancouver, BC, Canada, 2009. IEEE Computer Society.

[3] Jennifer Bevan and E. James Whitehead, Jr. Identification of software instabilities. In *WCRE '03: Proceedings of the 10th Working Conference on Reverse Engineering*, page 134, Washington, DC, USA, 2003. IEEE Computer Society.

[4] Cvs monitor. `http://sourceforge.net/projects/cvsmonitor/`.

[5] S.A. Drummond and M. Devlin. Software Engineering Students' Cross-Site Collaboration: An Experience Report. *Proceedings of The 7th Annual Conference of the ICS HE Academy Conference, Trinity College Dublin*, August 2006.

[6] J. C. Dunn. A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. *Journal of Cybernetics*, 3(3), 1973.

[7] M. Ester, H.P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial database with noise. *International Conference on Knowledge Discovery in Databases and Data Mining*, August 1996.

[8] Harald Gall, Mehdi Jazayeri, and Jacek Krajewski. Cvs release history data for detecting logical couplings. In *IWPSE '03: Proceedings of the 6th International Workshop on Principles of Software Evolution*, page 13, Washington, DC, USA, 2003. IEEE Computer Society.

[9] D. German and A. Mockus. Automating the Measurement of Open Source Projects. *Proceedings of the ICSE '03 Workshop on Open Source Software Engineering, page Automating the Measurement of Open Source Projects*, 2003.

[10] Michael Godfrey and Qiang Tu. Growth, evolution, and structural change in open source software. In *IWPSE '01: Proceedings of the 4th International Workshop on Principles of Software Evolution*, pages 103–106, New York, NY, USA, 2001. ACM.

[11] Ahmed E. Hassan and Richard C. Holt. Studying the chaos of code development. *Reverse Engineering, Working Conference on*, 0:123, 2003.

[12] Jazz overview, innovation through collaboration. `http://www-01.ibm.com/software/rational/jazz/`.

[13] P. Layzell, O.P. Brereton, and A. French. Supporting Collaboration in Distributed Software Engineering Teams. *Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific*, pages 38–45, 2000.

[14] Y. Liu and E. Stroulia. A Lightweight Project-Management Environment for Small Novice Teams. *Proceedings of the 3rd International Workshop on Adoption-Centric Software Engineering*, pages 42–48, 2003.

[15] Y. Liu, E. Stroulia, and H. Erdogmus. Understanding the Open-Source Software Development Process: A Case Study with CVSChecker. *Proceedings of the 1st International Conference on Open Source Systems*, pages 154–161, July 2005.

[16] L. Pekacki and D. Draheim. Bloof, an infrastructure for analytical processing of version control data. `http://bloof.sourceforge.net/`.

[17] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[18] K.L. Reid and G.V. Wilson. DrProject: A Software Project Management Portal to Meet Educational Needs. *Proceedings of the 38th SIGCSE technical symposium*, 39(1):317–321, September 2007.

[19] Gregorio Robles and Juan Carlos. Remote analysis and measurement of libre software systems by means of the cvsanaly tool. In *In Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS*, pages 51–55, 2004.

[20] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.

[21] G. Salton, A. Wong, and C. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[22] Anita Sarma, Larry Maccerone, Patrick Wagstrom, and James Herbsleb. Tesseract: Interactive visual exploration of socio-technical relationships in software development. In *ICSE '09: Proceedings of the 31st International Conference*

on Software Engineering, Vancouver, BC, Canada, 2009. IEEE Computer Society.

[23] David Sitsky. Code striker. `http://codestriker.sourceforge.net`.

[24] David Sitsky. Cvsplot. `http://cvsplot.sourceforge.net/`.

[25] Davor Čubranić and Gail C. Murphy. Hipikat: recommending pertinent software development artifacts. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 408–418, Washington, DC, USA, 2003. IEEE Computer Society.

[26] G. Venolia. Textual alusions to artifacts in software-related repositories. In *MSR '06: Proceedings of the 2006 International Workshop on Mining Software Repositories*, pages 151–154, New York, NY, USA, 2006. ACM.

[27] Timo Wolf, Adrian Schrter, Thanh Nguyen, and Daniela Damian. Predicting build failures using social network analysis on developer communication. In *ICSE '09: Proceedings of the 31st International Conference on Software Engineering*, Vancouver, BC, Canada, 2009. IEEE Computer Society.

[28] Zhenchang Xing and Eleni Stroulia. Api-evolution support with diff-catchup. *IEEE Trans. Softw. Eng.*, 33(12):818–836, 2007.

[29] Yunwen Ye. Codebroker. `http://l3d.cs.colorado.edu/~yunwen/codebroker/`.

[30] Thomas Zimmermann, Peter Weisgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 563–572, Washington, DC, USA, 2004. IEEE Computer Society.

Week	Cluster	Close	Missing	Extra	Artifact	User	Notes
3	1	✓			CS16	User4	First week of real development, these are all related to testing. Matched on words “person”, “familycollect”, “test”, “marriag”, and “job”.
					Ticket20	User4	
					Ticket26	User4	
					CS8	User4	
					Wiki1590	User4	
					Ticket21	User4	
					CS17	User4	
					Ticket27	User4	
					Ticket22	User4	
					Ticket19	User4	
					CS7	User4	
4	1			✓	CS26	User2	False match on text “details”.
					Ticket41	User2	
	2	✓			CS27	User2	Match on text “MenuBarAndToolBar”.
					Ticket37	User2	
	3	✓			CS36	User3	Match on text “edit” and “menu”.
					Ticket45	User3	
	4	✓			CS41	User3	Match on text “redundant” and “information”.
					Ticket46	User3	
	5	✓			CS44	User3	Match on text “tabbed” and “UI”. Two users.
					Ticket40	User2	
					Ticket50	User3	
					CS46	User3	
5	1	✓			CS115	User2	Match “GraphicalComponent”.
					Ticket68	User2	
	2	✓			CS47	User3	Match on text “edit” and “menu”.
					Message12	User3	
	3	✓	✓		CS53	User3	Match on text “help” and “menu”. Two users. Missing Ticket54.
					Message14	User1	
	4	✓	✓		CS63	User2	Match on text “info” and “bar”. Missing Ticket41. Two users.
					CS71	User2	
					Ticket67	User2	
					Ticket81	User3	
	5	✓	✓		CS64	User3	Match on text “tab” and “pane”. Missing Wiki2180. Two users.
					Ticket78	User1	

Table 1: Detected Clusters for Weeks 3 to 5

Week	Cluster	Close	Missing	Extra	Artifact	User	Notes
6	1	✓			CS145	User3	Match on text “message” and “InfoBar”.
					Ticket91	User3	
	2	✓			CS152	User2	Match on text “refactoring”.
					Ticket83	User2	
					Ticket84	User2	
	3	✓			CS160	User2	Match on text “add”, “popup” and “menu”.
					Ticket86	User2	
	4	✓	✓		CS167	User2	Match on text “functionality”. Missing Ticket87.
					Ticket85	User2	
					CS170	User2	
7	1	✓	✓	✓	CS228	User2	Match “search”, “results” and “display”. False match on CS228. Missing Ticket61.
					CS286	User2	
					Ticket111	User2	
	2	✓			CS235	User2	Match on text “code” and “cleanup”.
					Ticket102	User2	
	3	✓			CS236	User2	Match on text “exceptions”.
					Ticket103	User2	
	4	✓	✓		CS261	User1	Match on text “test”, “cases” and “family”. Missing Ticket27. Two users.
					Wiki2522	User4	
	5	✓			CS284	User3	Match on text “image” and “upload”.
					CS294	User3	
					Ticket114	User3	
8	1		✓	✓	CS321	User2	False match on “node”, “move” and “Panel”. Missing Ticket128.
					Ticket129	User2	
	2	✓			CS325	User2	Match on text “view”, “edit” and “person”.
					Ticket130	User2	

Table 2: Detected Clusters for Weeks 6 to 8

Week	Cluster	Close	Missing	Extra	Artifact	User	Notes
9	1		✓	✓	CS372	User2	False match on “refactoring”. Missing Ticket127.
					Wiki2887	User2	
	2	✓			CS373	User3	Match on text “edit”, “marriage” and “infopanel”.
					Ticket145	User3	
	3	✓	✓		CS390	User2	Match on text “nodepanel” and “singleton”. Missing Ticket95. Two users.
					Message49	User2	
					Message51	User3	
	4	✓	✓		CS404	User3	Match on text “fix”, “picture” and “select”. Missing Ticket123.
					Ticket173	User3	
					CS500	User3	
	5	✓	✓		CS409	User3	Match on text “editpanel”. Missing Ticket170.
					Ticket151	User3	
					CS558	User3	
	6	✓	✓		CS435	User2	Match on text “node” . Missing Ticket35 and Ticket193. Multiple versions of same wiki page. Two users.
					CS614	User2	
					Wiki2904	User4	
					Wiki2911	User4	
					Wiki2910	User4	
					Wiki2905	User4	
	7	✓			CS449	User2	Match on text “tab”, “button” and “delete”.
					Ticket130	User2	
	8	✓			CS496	User3	Match on text “edit” and “load”.
					Ticket172	User3	
	9	✓			CS498	User3	Match on text “person”, “document” and “editpanel”.
					Ticket170	User3	
	10	✓	✓		CS511	User3	Match on text “fixed”, “bug” and “editpanel”. Missing Ticket174. Two users.
					Ticket130	User2	
					Ticket130	User2	

Table 3: Detected Clusters for Week 9