Mining the Jazz Repository: Challenges and Opportunities

Kim Herzig
Saarland University
Saarbrücken, Germany
kim@cs.uni-saarland.de

Andreas Zeller
Saarland University
Saarbrücken, Germany
zeller@acm.org

Abstract

By integrating various development and collaboration tools into one single platform, the Jazz environment offers several opportunities for software repository miners. In particular, Jazz offers full traceability from the initial requirements via work packages and work assignments to the final changes and tests; all these features can be easily accessed and leveraged for better prediction and recommendation systems. In this paper, we share our initial experiences from mining the Jazz repository. We also give a short overview of the retrieved data sets and discuss possible problems of the Jazz repository and the platform itself.

1. Introduction

Mining software repositories for various purposes, such as bug prediction, has a long history. In recent years, projects like Eclipse and many Apache projects were among the most common data sources used by the mining community. Nearly all of these projects store their bug data separated from their version archive. This required the mining community to develop algorithms and techniques to detect fix locations subsequently.

With Jazz [1], IBM has implemented and published a brand new collaborative software engineering tool which advances common available repositories. Jazz integrates software archive and bug database by linking bug reports and source code changes to each other at the moment the change is delivered to the version archive. This not only allows software developers to improve their productivity but also makes it easy to organize development process data. This data can be mined and provide valuable insights into the development process of software projects.

As part of the *IBM Academic Initiative*, our group is able to access the development repository of Jazz through Jazz (Jazz was developed within itself). Mining this first available Jazz repository is a great opportunity to determine the usefulness, power and value of Jazz repositories.

In this paper, we describe our experience in mining the Jazz repository, provide insight into our extracted data sets before discussing various issues and possible conclusions for Jazz. In the next section, we explain the new features Jazz provides and how they impact the content of the repository. In Section 4 we briefly describe the structure of the Jazz repository and give an example data set, which we are able to extract. Before concluding, we discuss multiple issues of the repository and the Jazz system itself.

2. Why Jazz?

Jazz is a newly developed team collaboration software based on the highly accepted software development platform Eclipse. The focus of Jazz is not software development itself—Eclipse does it already—but enabling *team collaboration* as well as defining, administrating and integrating development processes.

Jazz provides huge opportunities for software mining and defect prediction. Software projects developed using Jazz provide more detailed data sets in which all artifacts (e.g. bug reports and specification items) are linked to each other. This traceability reduces the effort to rebuild such dependencies while mining the repository. The development data of Jazz provides a preview of future development data sets while its own repository serves the mining community with a new and extended data source.

By linking all artifacts to each other, Jazz allows software miners to track specification items to bug reports and vice versa. It is possible to map defects to those specifications that requested code changes introducing the defect. Such detailed insights in development and specification processes can rarely be found in any other repositories. Heuristics that rebuild such artifact dependencies, as used by Zimmermann et al. [5], are no longer required; instead, recorded dependencies will raise the accuracy and detail level of extracted data sets.

Additionally, Jazz stores discussion artifacts and information on development teams that can be used to mine collaboration data as illustrated by Nguyen et al. [3]. As Jazz links such data to development process information and defect data, the Jazz repository may serve as a platform to combine many existing approaches to mining, prediction, and recommendation. The resulting traceability makes Jazz and its development repository one of the most valuable data sets for any kind of software mining.

3. Mining Jazz

The opportunities and features discussed in Section 2 come at a price. Accessing and analyzing a Jazz repository is far more difficult than accessing and analyzing separate version or defect repositories (like Subversion or Bugzilla). The reason is the *high complexity of the database* and the *huge amount of data* that needs to be stored in order to offer high artifact traceability.

Nguyen et al. [3] describe the four different methods available to access a Jazz repository and their drawbacks: *Report*, *Client API*, *Server API* and the direct connection to the *Database*. Accessing the database directly, without any API or tool between miner and data, seems the easiest alternative, but also raises risks. The Jazz database contains more than 200 database tables. Fetching data sets via plain SQL commands implies a high risk of retrieving wrong or incomplete results and makes the approach very fragile regarding future changes applied to Jazz.

For mining Jazz, we therefore decided to use the *client API*. This enabled us to rely on existing functionality provided by Jazz developers. Our mining tool uses Jazz services only and runs as a Jazz plugin. This allows us to hook into any existing and future Jazz repository by being as extensible as possible.

The high artifact traceability of Jazz mentioned in Section 2 is the key feature to be used when mining the Jazz repository. The concept behind Jazz artifacts and its dependencies are so called *work items*. Ying et al. [4] and Cheng et al. [2] compare work items with Bugzilla reports and explain the connections between work items and other collaboration features. A work item represents a bug report, an enhancement, a plan item (used for specification) or any kind of task. In its most simple occurrence, a work item contains all information a normal bug report does.

The main advantage of Jazz is based on *work item edges* that connect work items, discussions and other artifacts such as screenshots and *change sets* (a semantic related set of changes on files made to the Jazz repository) to one big *work items graph*. All edges of this graph are labeled by the type of the work item dependency (see Figure 1). Figure 2 shows an excerpt from the Jazz work item dependency graph. This connectivity brings a number of benefits:

Bug reports are automatically linked to change sets. Among other dependencies, the work item graph connects work items and corresponding change sets by requesting the developer to attach one work item to each source code change he wants to check in. For bug reports the attached change set identifies the bug fix locations. Heuristics such as the one used by Zimmermann et al [5] to identify bug fix locations in source code entities are no longer needed.

Defects can be tracked back to specifications. The relations between different work items across and types allow us to identify dependencies between various kinds of work items. Bug reports can be mapped to specification items and thus identify the cause for the bug. Inter work item dependencies may identify change set dependencies indicating change impact.

The work item graph contains collaboration data. User discussions are directly assignable to work items, change sets and problems. This collaboration data can be used to gain detailed information on developer activities and interaction. Jazz research projects like that of Nguyen et al. [3] are able to combine their results with bug reports and other process structure measurements.

4. The Data Set

The only Jazz repository data set available so far is the Jazz development repository itself (see Table 1). It does not reflect the complete development process starting from day zero but it covers most of the development activity including early stages.

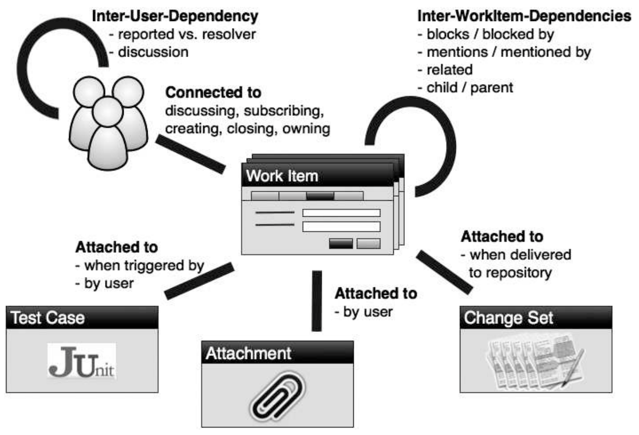

Figure 1. Components of the Jazz work item graph.

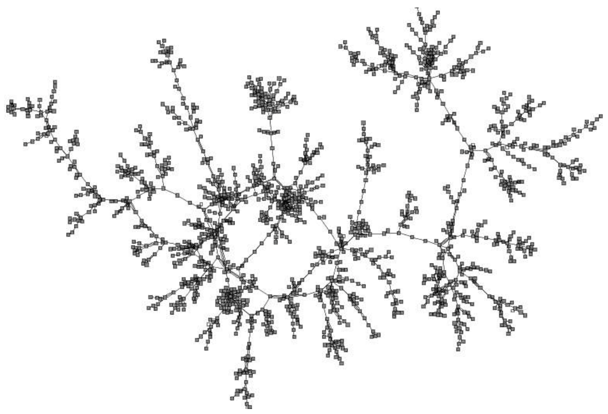

Figure 2. Partial work item dependency graph. Each work item is represented by a node while edges refer to work item dependencies.

Release of Version 1.0.1	Oct 2008
Data set from	Jun 2008
Included work items	Jun 2005 to Jun 2008
Included change Sets	Jun 2007 to Jun 2008
# work items	$\sim 47,000$
# bug reports	$\sim 34,000$
# resolved bug reports	$\sim 21,000$
# bug report with change set	$\sim 16,000$
# classes	$\sim 16,000$
# plan items (specification)	~ 150

Table 1. Jazz data set overview

For each work item in the repository, we can extract a set of metrics (see Table 2) that describe the dependency of each work item to other work items, discussions and artifacts. Of course, the metrics also include all direct properties of a work item (e.g. severity, category).

5. First Results

With its first release late 2008, Jazz is a very young software project. This implies that the available data set (last updated in June 2008) includes pre-release data only. In comparison to Zimmermann et al. [5], we can only work on early development data. This has multiple effects on the usefulness of the data set and its content.

Since Jazz is developed within Jazz, we have to consider that many features provided by Jazz today were not implemented in the early stages of the development process. Table 1 shows that the time window of available change sets is about two years shorter than the available work item history. This time difference and the fact that not all features of Jazz were available at all times implies that the Jazz repository contains multiple holes and misleading fragments that cannot be eliminated or identified easily. Tasks in early stages of the development process could not be created as work items, many work items (also bug reports) are not linked to change sets and there might be multiple other noise sources.

Furthermore, we have to consider the nature of young software projects and the development processes within such projects. Fixing early software defects often requires large changes as well as fixing multiple defects in parallel. Thus, to apply only compilable source code to the version archive, the developer has to commit multiple fixes at once. However, Jazz allows only one work item attached to each change set. This leads to the fact that many bug reports will be closed without any change set attached. Even worse, this also implies that change sets linked to work items may contain source code changes that were not necessary to fix the linked bug but could not be applied to the version archive separately.

Unfortunately, the number of specification items in Jazz is very low. When planning and specifying major parts of the system, the tools needed to create plan items were not available. We believe, though, that plan items and their dependencies to other work items could provide additional

Details directly extracted from work item
Type of work item
Severity of work item
Category the work item was filed in
User that opened the work item
User that closed the work item
Iteration Plan the work items was applied to
Priority of work item
Effort estimated by developer to complete work item
Time Spent (not yet implemented in Jazz)
Number of days open
Metrics computed on work item graph
Number of connected work items
Number of connected specifications
Number of connected defects
Number of child work items
Number of parent work items
Number of comments in the discussion
Average size of comments
Number of distinct users contributing in discussion
Number of work items mentioning this work item
Number of defects per specification item
Number of test cases attached to work item
Number of approvals / code reviews
Number of classes touched by attached change set

Table 2. Work item metrics for Jazz

benefits for defect prediction and recommendation models. The current content of the Jazz repository does not allow an extended investigation of this particular work item subgraph.

At the current stage, we are able to extract complete work item graphs from Jazz and are now mining them for interesting program and process patterns related to defects or maintenance issues. For example, we have been investigating the question whether there would be correlations between the time a defect report was open and the number of discussions or the number of users participating in the discussion. Figure 3 shows a scatter plot showing the days a work item has been active in relation to its number of comments. As one can see, the correlation between the two values is very low (0.26), but it gives promising insights into the data sets we are able to extract and analyze.

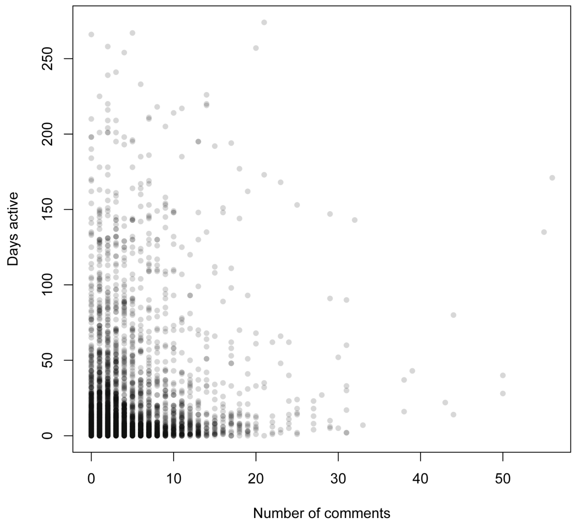

Figure 3. Defect reports: Days active (gray line) and the corresponding number of comments (black dots)

6. Conclusion

Mining the Jazz repository is not a trivial task and requires the miner to deeply dive into the Jazz code and its architecture when using the client API to complete the task. However, the resulting data set is worth the effort and contains many new insights into development processes and artifact dependencies that cannot be retrieved using other project repositories.

Even if the current data content of the Jazz repository is incomplete and may contain a lot of noise, it can be used to develop new ideas and techniques that will make new contributions to software mining and defect prediction communities. In the long run, these initial limitations will be overcome and Jazz and its concepts of integration and data traceability will gain more and more importance and benefits to research.

Acknowledgments. We thank the Jazz team for providing and approving the Jazz data and constructing the legal agreement. This work is funded by a IBM Jazz Faculty Grant.

References

[1] IBM rational jazz. http://jazz.net/.

[2] P. Cheng, S. Chulani, Y. B. Ding, R. Delmonico, Y. Dubinsky, K. Ehrlich, M. Helander, T. Klinger, A. Kofman, P. Matchen, V. T. Rajan, G. Saadoun, A. Sempere, P. Tarr, C. Williams, P. F. Xiang, A. Yaeli, S. X. Yang, and A. Ying. Jazz as a research platform: experience from the software development governance group at IBM research. In *Proceedings of the 1st International Workshop on Infrastructure for Research in Collaborative Software Engineering (iReCoSE)*, November 2008.

[3] T. H. Nguyen, A. Schröter, and D. Damian. Mining Jazz: An experience report. In *Proceedings of the 1st International Workshop on Infrastructure for Research in Collaborative Software Engineering (iReCoSE)*, November 2008.

[4] A. T. Ying, K. Ehrlich, L.-T. Cheng, H. L. Ossher, T. V. Frauenhofer, and F. van Ham. Jazz development data: a community perspective. In *Proceedings of the 1st International Workshop on Infrastructure for Research in Collaborative Software Engineering (iReCoSE)*, November 2008.

[5] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for Eclipse. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, May 2007.