Using Change Entries to Collect Software Project Information

Hazeline U. Asuncion[1], Macneil Shonle[1], Robert Porter[2], Karen Potts[1], Nathan Duncan[1], William Joseph Matthies Jr.[1]

[1] Computing and Software Systems
University of Washington, Bothell
Bothell, WA 98011 USA
{hazeline, mshonle, pottsk2, njd91, wjoem}
@ u.washington.edu

[2] Department of Computer Science
The University of Texas at San Antonio
San Antonio, TX 78249 USA
robertportercs@gmail.com

***Abstract*—Confronted with tight project deadlines, a development team is often under pressure to make decisions regarding the project (e.g., Which features can be included in the next release? Is the software product ready for release?). In order to make these decisions, it is necessary to obtain information from multiple sources, including source code in different languages and documentation in different formats. In this paper, we present a technique that uses change entries to obtain relevant project information. Our technique, FACTS PT, automatically extracts, traces, aggregates, and visualizes change entries along with other software metrics to provide project information. Results from our case study at the ABC Organization* suggest that the information provided by the FACTS PT is useful to project managers and developers. We also offer lessons learned regarding collecting and presenting information to a team in a proprietary and regulated software development context.**

Keywords-Software traceability; Software analytics; information management

I. INTRODUCTION

Connecting related information during the course of software development, referred as software traceability, is necessary for various software lifecycle tasks such as determining conformance to requirements or assessing the quality of design [7, 14]. However, connecting related information found in different artifacts to answer project management questions has received little attention [9]. (We define an artifact as any human-produced file during the software lifecycle.) Ideally, a team should be able to answer questions related to project status, quality of software artifacts, and compliance to requirements. To answer these types of questions and to make informed decisions, information from various sources must be collected.

Collecting project information has been referred to as software metrics [16], or software telemetry [18]. Techniques have been developed to support the automated collection of both process and product metrics [13, 18, 22, 26]. These techniques, however, are generally focused on collecting process information to aid developers assess their own skills and productivity. Project managers, meanwhile, require process and product information that provide them an understanding of the overall state of the code, the progress toward a milestone, and the skills and productivity of each member of the team. Buse and Zimmerman recently used the term software analytics to refer to the type of data and data analysis necessary to support managers in their decision-making tasks [9]. This paper provides a technique to support software analytics for project managers and developers.

Agile software development methods also have techniques for collecting project information. Burn down charts are used to track the velocity of a team over several iterations [11]. These techniques are built-in to the agile process and are difficult to adapt to teams which use other lifecycle models, such as the waterfall or spiral models. Incidentally, the concept of software analytics is also being used in agile projects [25].

We present a technique for tracing related information from various sources. Our technique, Flexible Artifact Change and Traceability Support for Project Team (FACTS PT), uses change entries as a level of abstraction by which changes across various artifacts can be uniformly represented and monitored. Previous approaches to collecting project information include tracking number of errors detected or number of lines of code added [16]. We posit that change entries do not only serve as a useful metric for project progress, but they also provide more meaningful information regarding the *reason* behind specific changes. Our contributions are: 1) change entries as a means of gathering scattered project information, 2) a set of tools for extracting, tracing, aggregating, and visualizing change entries and other metrics, 3) evaluation in a real-world setting which suggests the utility of our approach, and 4) lessons learned from the study.

This paper is organized as follows. The next section provides a motivation behind our technique. Section 3 presents our FACTS PT technique, followed by tool support in Section 4. Section 5 covers evaluation of our technique with lessons learned. Section 6 discusses related work. We close the paper with avenues of future work.

II. MOTIVATION

We now provide an overview of the challenges faced by a development team working in a proprietary and regulated software development context, such as the ABC Organization* where we conducted a case study of our technique.

This work is supported by the US National Science Foundation under Grant No. 1218266.

* Kept anonymous here due to a non-disclosure agreement.

The ABC Organization is a research institute which engages in scientific research and develops software for various applied science domains. The organization has several thousand employees with branches in the United States and around the world. Because the organization works on software projects that must adhere to government standards and regulations, and because these software products may be deployed on critical systems, the software must pass a rigorous quality assurance standard. The organization also uses the Capability Maturity Model Integration (CMMI) level as a means of demonstrating its maturity level [2].

The software development team which we studied uses a hybrid waterfall and iterative software development lifecycle. The team comprises of a project manager, a lead software engineer, 3-4 developers, 1-2 test engineers, and a documentation engineer. The team releases software in roughly 10-month cycles. Not only must the organization be able to assess the quality of their code, they must also demonstrate process maturity to reach the next CMMI level.

Since the group uses an iterative development, it is important for the development group to continually monitor changes performed on the code and accompanying artifacts for each release cycle. In addition, the team must be able to quickly obtain information from various artifacts, which are often scattered among the various tools used by the team. Finally, the team must be able to answer questions such as "What is the status of the project?" and "Are the source code and documentation meeting quality standards?"

III. FACTS PT TECHNIQUE

To address the challenges faced by a development team in tracing relevant information from heterogeneous sources, we used the FACTS PT technique. This technique consists of the following steps: select a tracing unit to connect heterogeneous sources of information, embed concepts within change entries, extract change entries and other project metrics, aggregate and visualize extracted data. We now discuss each step in detail.

A. Select a Tracing Unit to Connect Heterogeneous Sources of Information: Change Entry

Since a software team is confronted with large amounts of information from multiple sources, it is necessary to only trace the relevant information. One of the challenges with many traceability recovery techniques is the generation of potentially large number of false positive links along with accurate links [19]. This can be addressed by identifying a tracing unit that can connect heterogeneous information across the project lifecycle. The tracing unit serves as a means of "marking" artifacts, or parts of the artifacts, to trace.

Previous techniques use requirements [7], architecture concepts [5], or events as tracing units [24]. We connect information via change entries, which are descriptions of changes made in the artifacts (e.g., source code, documentation) (see Figure 1). A change entry can be mapped to specific deletions or additions in a file. Since change entries can be uniformly represented across heterogeneous artifacts, we use change entries as our tracing unit.

B. When Possible, Embed Project Concepts within Change Entries

Once the tracing unit is selected, the next step is to embed it with concepts understandable by a team. In our context, an important higher level concept is a task. A task is a software update to be performed by a software engineer. A task may entail implementing a new feature, performing a bug fix, or carrying out a code maintenance activity.

Since artifacts are at different levels of abstractions (e.g., requirements document contain abstract concepts while source code contain concrete concepts), project concepts may only be embedded in some change entries. For example, the concept of task is a higher level concept by which source code changes can be aggregated. Meanwhile, requirements document changes may not be related to tasks since concepts at the requirements document are at a higher level of abstraction than tasks. It is certainly possible to embed requirement change entries with another set of concepts, such as requirement IDs, and then develop a mapping between requirements and tasks.

C. Extract Change Entries and other Project Metrics

The next step is to extract change entries and other relevant project metrics to gather project information. To address the challenge of extracting change entries from heterogeneous artifacts, we use artifact-specific extractors, such as an extractor for each source code language, an extractor for specification documents in PDF, an extractor for test documents in PDF, etc.

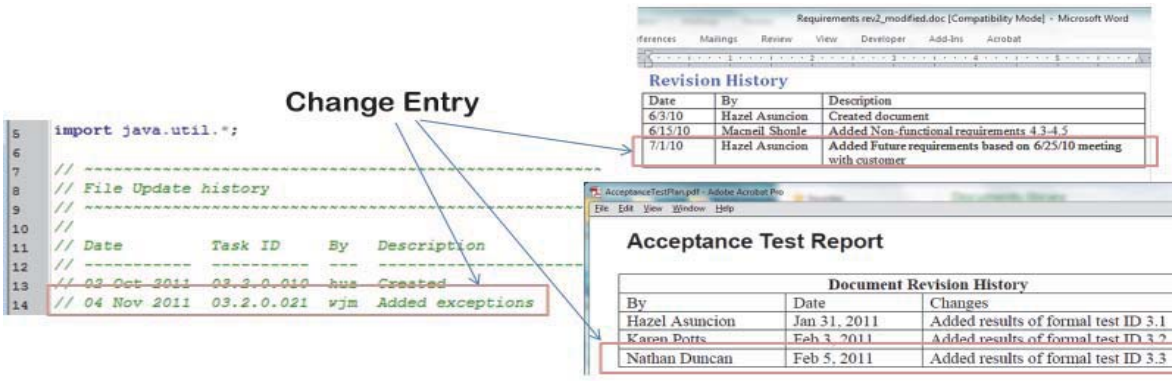

Figure 1. Examples of change entries on various software artifacts in different formats (source code, requirements documentation, and test report).

The extracted change entries are then represented as a uniform change model. We use XML to represent our change model, which contains the following information: a unique identifier, author, date of the change, task ID (optional), description, and path. All of these attributes, except the path, are obtained from change entries. The description is a free-form text that describes the change. To support access to the artifacts that were changed, it is also necessary to determine the location of the artifact–whether on the local machine, on the local network, or on the Internet. The path may also point to a specific location within the artifact to support accessibility at different levels of granularity [6]. The path is automatically determined relative to the project root folder. An XML file contains multiple change entries and multiple XML files may be used to encapsulate groups of change entries.

We also collect pertinent project metrics associated with managing the project. These include length of time spent by a developer on a task and number of lines changed for source code files. To minimize the overhead in collecting metrics, each developer simply reports the total number of hours spent on each task. The project manager compiles these self-reported hours into a Software Tasks spreadsheet, which includes additional task information. Once the information is in a spreadsheet, we can automatically extract the self-reported hours. Metrics regarding number of lines changed for source code files are obtained using a diff tool (e.g., SVN diff [12]).

D. Aggregate and Visualize the Extracted Data

Once the change entries, which contain task IDs, and other project metrics have been obtained, the data can then be aggregated and visualized at different levels of granularity. At a high level, an overview of project progress and potential problem areas in the code can be provided. At a detailed level, one can view how the aggregated information was derived, which parts of the documentation or source code has been changed, which parts of the code are non-compliant to coding standards, and which tests have passed or failed.

We generated the following types of visualizations: Change Lookup by Developer, Release Comparison Report, Change Distribution Chart, Timeline of Change Entries, Gantt Chart Actuals, and Task-Author-Artifact Report. The Change Lookup by Developer allows developers to search for their tasks and all the changes they performed for the current iteration. The Release Comparison Report shows a listing of all the source code files that have been changed between two specified releases. Within this report, additional information such as detailed diff and coding standard reports are provided for each file and are accessible via hyperlinks. The Change Distribution Report, meanwhile, visualizes changes along various dimensions, by software engineer, by file type, by subsystem, by hours spent, or a combination of these dimensions. The Timeline of Change Entries provides a chronological ordering of change entries. The change entries

Figure 2. Gantt Chart Actuals using data from the jEdit open source project. Top figure shows zoomed out view of the chart. Bottom figure is a zoomed-in view of the UI Task with a comment indicating the authors who performed changes for the selected date along with the number of change entries.

are color-coded according to various dimensions (e.g., author, file type, subsystem). We discuss in detail the rest of the visualizations.

Gantt Charts are generally used for project planning to determine when a software product can be released [27]. We have adapted the Gantt Chart to show actual project information based on change entries (see Figure 2, top screenshot). Similar to a Gantt Chart, the Gantt Chart Actuals lists tasks (or task IDs) on the left side of the chart. Each Gantt row spans a period of n months and is color-coded based on the number of change entries. Upon hovering over a colored heat map cell, additional metadata such as list of authors that performed changes for that date and number of change entries in brackets, are shown (see Figure 2, bottom screenshot).

Change entries, along with its mapping to a more abstract concept (e.g., task) and more concrete concepts (e.g., lines added or deleted, time spent), are shown in the Task-Author-Artifact Report. In this report, a user can select a task ID. A list of software engineers who are working on the specified task is shown, along with summary data for the number of change entries and hours spent for the selected task. Within each developer, detailed information is shown, such as files that were changed, the percentage of the total lines that are comments, number of lines added and deleted. This report also allows a user to assess the complexity of a task by combining self-reported information (number of hours) along with automatically generated metrics, such as the number of files modified and number of lines added or deleted.

IV. Tool Support

We built artifact-specific change extractors using Perl and Python scripts to extract change entries from requirements and design specifications (in PDF and Word) and from source code written in Java and JSP. Since each artifact type follows company standard formatting, it is straightforward to locate the section of the file that contains the change entries. Task IDs are included with each source code change entry (see Figure 1), and are included in the extraction process. Once the change entries are found, the change extractors write each change entry to an XML file with the appropriate XML tags. All generated XML files are then combined into one file.

The change entries and the project metrics, both in XML formats, are then fed into the visualizations. The Timeline of Change Entries was built using Piccolo 2D. The spreadsheet visualizations were built on top of Microsoft Excel 2010 spreadsheet in the .NET 4.0 Windows environment using C# and the Excel API. Spreadsheet visualizations were used since the team was comfortable with analyzing project data in the spreadsheet environment. The Gantt Chart Actuals and timeline visualizations have been fully implemented, while the other spreadsheet visualizations are partially implemented. For the Release Comparison Report, we used an SVN diff [12] to identify the differences between two releases. We also built a script that analyzes high priority coding standard violations and outputs a report for each source code file. Alternatively, an off-the-shelf coding standard checker could also be used.

V. Evaluation

The FACTS PT technique and tool support was evaluated in the context of an ongoing software project that has over 200K total lines of code and implemented in five different languages and scripts. We analyzed the change entries from the Java and JSP codebase which covers about half of the total codebase. The project also has numerous artifacts including requirements specifications, design documents, test plans, test documents, checklists, tasks, and change requests. These artifacts are in different file formats (e.g., spreadsheets, documents, PDF files, diagrams). FACTS PT was used to extract change entries from a subset of these artifacts, to relate tasks and change entries to developers, and to support tracking project progress. The artifacts from three major releases were studied, with thousands of change entries. The change entries spanned the period of January 2009 to May 2011.

We were primarily focused on determining the utility of change entries and their visualizations to developers or project managers for their respective tasks. Thus, we sought answers to the following research questions:

Q1: Does the FACTS PT technique assist you in development or management tasks? If so, in what way?

Q2: How useful are the FACTS PT visualizations?

We solicited information from various members of the team: a project manager who has 15 years of experience as a software project manager and three programmers who have about 10-15 years of experience. The subjects were presented with the visualizations after the releases and were asked to provide feedback via questionnaires and semi-structured interviews. We conducted four iterations of the study (and improved the FACTS PT tool support after each iteration).

A. Results

Q1: In the early iterations of the study, both the project manager and developers concurred that the FACTS PT technique did not assist them in their tasks.

In later iterations, both project manager and developers stated that FACTS PT can assist them in their tasks. The developers stated that the FACTS PT technique allowed them to quickly identify which files have changed and to understand the source code changes. The project manager stated that in its current state, the FACTS PT technique can assist him with project management tasks by identifying areas of improvement from the generated visualizations and reports after a major release. These areas of improvement can then be addressed in the next software development iteration.

Q2: In early iterations of the study, the development team stated that the FACTS PT visualizations were not useful.

In later iterations, the visualizations were useful to the development team. The programmers were able to quickly locate the changes they made with the Change Lookup by Developer and were able to reflect upon their own productivity. According to the project manager, the Gantt Chart Actuals (Figure 2) and the Task-Author-Artifact Report were rated as providing highly useful information because they provide

summary information. The other visualizations require further changes to be rated as highly useful.

B. Discussion

Q1: Throughout the four iterations, we followed the same general steps of extracting, aggregating, and visualizing change entries, except for the additional steps of selecting and embedding project concepts within change entries and including project metrics in later iterations. It turned out that these additional steps were keys in transforming the FACTS PT into a technique that can assist project managers and developers with their tasks. Viewing changes at the granularity of change entries was acceptable to all the subjects.

Q2: Throughout the study, we visualized change entries. It was interesting to learn that the type of visualization can largely affect the perceived usefulness of the change entries. Although the Timeline of Change Entries we initially used provided some insight into the project, all subjects had difficulties navigating it and was unable to quickly obtain aggregate information across different dimensions (e.g., by developer). Meanwhile, all subjects found the tabular visualization format as most useful, especially when it contained information extracted from various sources, as in the Task-Author-Artifact Report or the Change Lookup by Developer. The project manager added that when the FACTS PT tool support becomes more mature, it can also be used during a project iteration, as opposed to simply being used at the end of an iteration as part of a post-release analysis. The developers also expressed interest in visualizing their changes from other projects. Doing so would allow them to cross-reference the changes they make across different projects.

C. Limitations

Our approach makes the following assumptions:

Change entries are present in the files to trace. Development teams which produce formal specifications often have a history log as part of the document template (see IEEE Std 830-1998 [1]). In addition, many development teams also use a configuration management (CM) system which contains commit records. These commit records can be used as a source of change entries if history logs are not used. Open source projects also maintain a change log [10]. Time spent on tasks may be estimated from CM check-out/check-in timestamps.

It is possible that the developer-entered information, such as hours spent on tasks and change entries may contain incorrect information, or even missing information [10]. This inaccuracy would be fairly straightforward to detect since the developer-entered information is presented with the automatically generated metrics. In addition, if developers are incentivized for demonstrating a higher level of activity, via the change entries, it is less likely they will neglect providing change entries each time they make a change to a file.

With regards to limitations with our evaluation, we focused on whether tracing, aggregating, and visualizing change entries with other project metrics is useful to project managers and developers. We did not examine the overhead involved with processing the artifacts. This is a subject of future work.

D. Lessons Learned

1. A software development team is not keen on using new tools or technology unless they have a direct benefit. This finding is consistent with the adoption of software traceability techniques in industry [7]. Thus, even though we also presented change entries in the earlier iterations, the team was not willing to use the tool because the information was not accessible to them. Later on, when we presented the change entries in both the aggregated and detailed level, along with other project metrics, the team was more willing to use our tool and technique.

2. Aesthetically pleasing visualizations do not necessarily provide usable information. Since a development team is constantly under time pressure, a visualization must enable them to obtain information quickly. Thus, support for easy navigation, filtering, searching, and data manipulation are key requirements for a usable visualization. This is one of the reasons why tabular formatted data is preferred by the subjects. The information can easily be aggregated (by invoking the sum function), filtered by an attribute, or searched by a keyword. Our timeline visualization, while aesthetically pleasing and classifies changes according to author or file type, does not provide capabilities for fast data manipulation, and thus, was not useful to them.

3. Using a combination of self-reported and automated metrics can lower the overhead for collecting metrics, while minimizing privacy issues that may be associated with fully automated data collection techniques. By leveraging existing company practices in extracting metrics, more applicable metrics can be obtained. Moreover, some of the fully automated techniques in metric collection may under-report the actual time spent on an activity. Since the automated techniques are based on engineer interaction with tools [18, 26], these techniques do not measure the time away from the computer (e.g., face-to-face meetings with teammates).

On the other hand, manually tracking time can be a time consuming process [16] and may potentially distract engineers from their task since they are required to context switch between tracking their time and performing development tasks [17]. In a company setting, a balance can be achieved by tracking course-grained activities and tracking time at 10 or 15 minute increments. Recording time spent on activities can be performed at the same time as engineers report their timesheets (e.g., once a day), to avoid the context switching problem. In our context, the engineers track their time at the task level and the reported times were generally accurate.

VI. Related Work

We now compare our work to related research areas.

Software Traceability: Software traceability research is concerned with identifying relationships between various software artifacts [5]. Traceability techniques and approaches have generally been developed to support an analyst or a requirements engineer [15], an architect [5], or a developer [4], but not project managers. One case study describes the use of bug tracking as a tracing unit to support developers [21]. If a tracing unit is not embedded into the artifacts to trace, then

other techniques like link recovery techniques [4, 15] can be used to identify possible connections between artifacts. Limited traceability support for project management tasks was described in another study [7]. Jazz is a tool that supports mapping of information across various artifacts that reside within the Jazz platform [3]. Our work, however, uses change entries as a tracing unit, and aggregates and visualizes them with other project metrics to support developers and managers. Our work is also not constrained to a specific tool or platform.

Process metrics: Several process metrics have been previously proposed to support project management, including time spent on activities and number of defects discovered during code inspection [16]. Goal-Question-Metric paradigm provides guidance on which metrics to collect [8], while CQMM is a technique that collects metrics from different static analyses tools to monitor and assess code quality [23]. In agile development contexts, story points are used to track the amount of work performed in each iteration [11]. Using change entries as a metric is complementary to these techniques.

Metric Collection: There are tools that collect process and product metrics. One particular category of tools, Software Project Control Centers (SPCCs) are used to collect, interpret, and visualize project metrics to provide context-, purpose-, and role-oriented information for various members of the development team [20]. Other tools use different techniques to collect metrics, such as using sensors attached to various tools (e.g., development editors, build tools) [18], tracking evolution of classes, methods, invocations [22], or tracking personal software process (PSP) data [26]. Another tool allows users to plug-in their custom metrics into a provided infrastructure [13]. Our technique, meanwhile, combines self-reported metrics from developers (i.e., time spent on tasks) with extracted change entries and product metrics.

VII. Conclusion

In this paper, we presented FACTS PT, a technique that traces change entries across heterogeneous artifacts to collect project information. We developed a set of tools that automatically extracts, traces, aggregates, and visualizes change entries along with other project metrics. Our case study at a proprietary and regulated software development context indicates that our approach is useful to project managers and developers. We also offered lessons learned regarding collecting and presenting accessible information to a software development team.

We plan to continue improving the FACTS PT tool support. We will also solicit feedback of other members of the development team, including QA engineers and documentation engineers, to determine how our technique can also assist them in their tasks. Finally, we plan to analyze the description of change entries to automatically group together related changes.

VIII. Acknowledgements

We thank the project manager and developers at the ABC Organization for valuable insights and feedback. Dang Nguyen at UTSA developed the initial timeline visualization.

References

[1] IEEE Recommended practice for software requirements specifications. *IEEE Std 830-1998*, 1998.

[2] CMMI Institute. http://cmmiinstitute.com/, Jan 2013.

[3] The Jazz Project. http://jazz.net, Jan 2013.

[4] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering (TSE)*, 28(10):970–983, 2002.

[5] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor. Software traceability with topic modeling. In *Proc of 32nd Int'l Conference on Software Engineering (ICSE)*, 2010.

[6] H. U. Asuncion and R. N. Taylor. *Software and Systems Traceability*, chapter Automated Techniques for Capturing Custom Traceability Links across Heterogeneous Artifacts, pages 129–146. Springer London, 2012.

[7] H.U. Asuncion, F. François, and R. N. Taylor. An end-to-end industrial software traceability tool. In *Proc of the 6th Joint Meeting of the European Software Eng Conf and the ACM SIGSOFT Int'l Symp on the Foundations of Software Engineering (ESEC/FSE)*, 2007.

[8] V. Basili and S. Green. Software process evolution at the SEL. *IEEE Software*, 11(4):58 –66, 1994.

[9] R.P.L. Buse and T. Zimmermann. Information needs for software development analytics. In *Proc of ICSE*, 2012.

[10] K. Chen, S. R. Schach, L. Yu, J. Offutt, and G. Z. Heller. Open-source change logs. *Empirical Software Engineering*, 9(3):197–210, 2004.

[11] M. Cohn. *Agile Estimating and Planning*. Prentice Hall, 2006.

[12] CollabNet. TortoiseSVN. http://tortoisesvn.tigris.org/, Jan 2013.

[13] G. Gousios and D. Spinellis. A platform for software engineering research. In *Proc of Int'l Working Conf on Mining Software Repositories*, 2009.

[14] V. L. Hamilton and M. L. Beeby. Issues of traceability in integrating tools. In *IEE Colloquium on Tools and Techniques for Maintaining Traceability During Design*, 1991.

[15] J.H. Hayes, A. Dekhtyar, and S.K. Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *TSE*, 32(1):4–19, 2006.

[16] W. Humphrey. *A Discipline for Software Engineering*. Addison-Wesley, 1995.

[17] P. M. Johnson and A. M. Disney. A critical analysis of PSP data quality: Results from a case study. *Emp Software Engr*, 4(4):317–349, 1999.

[18] P. M. Johnson, H. Kou, M. Paulding, Q. Zhang, A. Kagawa, and T. Yamashita. Improving software development management through software project telemetry. *IEEE Software*, 22(4):76 – 85, 2005.

[19] J. Leuser. Challenges for semi-automatic trace recovery in the automotive domain. In *Proc of the 5th Int'l Workshop on Traceability in Emerging Forms of Software Engineering*, 2009.

[20] J. Münch and J. Heidrich. Software project control centers: concepts and approaches. *Journal of Systems and Software*, 70(1–2):3 – 19, 2004.

[21] C. Neumüller and P. Grünbacher. Automating software traceability in very small companies - a case study and lessons learned. In *Proc of the 21st Int'l Conference on Automated Software Engineering*, 2006.

[22] J. Oosterman, W. Irwin, and N. Churcher. EvoJava: A tool for measuring evolving software. In *Proc of the Australasian Computer Science Conference*, 2011.

[23] R. Plösch, H. Gruber, C. Körner, and M. Saft. A method for continuous code quality management using static analysis. In *Proc of the Int'l Conf on Quality of Information and Communications Technology*, 2010.

[24] W. Poncin, A. Serebrenik, and M. van den Brand. Process mining software repositories. In *Proc of the European Conference on Software Maintenance and Reengineering*, 2011.

[25] Rally Software. Advanced analytics. http://www.rallydev.com/platform-products/advanced-analytics, Jan 2013.

[26] A. Sillitti, A. Janes, G. Succi, and T. Vernazza. Collecting, integrating and analyzing software metrics and personal software process data. In *Proc of the Euromicro Conference*, 2003.

[27] D. White and J. Fortune. Current practice in project management — an empirical study. *Int'l Journal of Project Mgmt*, 20(1):1 – 11, 2002.