Effective Mining of Software Repositories

Marco D'Ambros
REVEAL
Faculty of Informatics
University of Lugano, Switzerland
Email: marco.dambros@usi.ch

Romain Robbes
PLEIAD Lab
Computer Science Department (DCC)
University of Chile, Chile
Email: rrobbes@dcc.uchile.cl

I. Presenters

Marco D'Ambros earned his Ph.D. in October 2010 and is currently a postdoctoral researcher at the REVEAL Group, University of Lugano, Switzerland. He previously received MSc degrees from both Politecnico di Milano (Italy) and the University of Illinois at Chicago. His research interests lie in the domain of software engineering with a special focus on mining software repositories, software evolution, and software visualization. He authored more than 25 technical papers, and is the creator of several software visualization and program comprehension tools.

Contact:
Dr. Marco D'Ambros
Faculty of Informatics
University of Lugano
Via G. Buffi 13, 6904, Lugano, Switzerland
web: http://www.inf.usi.ch/phd/dambros/
email: marco.dambros@usi.ch

Romain Robbes is an Assistant Professor at the University of Chile. He earned his Ph.D. in December 2008, from the University of Lugano, Switzerland and received his Master's degree from the University of Caen, France. His research interests lie in Empirical Software Engineering and Mining Software Repositories. He authored more than 30 papers on these topics, including top software engineering venues (ICSE, ASE), and best paper awards at WCRE 2009 and MSR 2011. He is program co-chair of IWPSE-EVOL 2011, and the recipient of a Microsoft SEIF award 2011.

Contact:
Prof. Dr. Romain Robbes
PLEIAD Lab, DCC,
University of Chile
University of Lugano
Blanco Encalado 2120, Off. 308
Santiago, Chile
web: http://romain.robb.es
email: romain.robbes@dcc.uchile.cl

II. Goal and Objectives

The overall goal of this tutorial is threefold: (1) provide an overview of what can be learned from software repositories; (2) teach how to effectively mine such repositories, describing concrete techniques and tools; and (3) highlight the limitations of current software repositories and MSR approaches, presenting novel repositories and data structures that can be used instead. We expect a participant, after the tutorial, to be able to (1) know to which extent MSR can help her in learning about her software system, and for which tasks MSR techniques can be employed; (2) apply concrete MSR approaches to support software maintenance activities; and (3) be aware and take into account the limitations of current MSR approaches.

In particular, the tutorial aims to answer the following questions:

Overview

- Which repositories can be used to perform MSR?
- How are these repositories structured, and how much data processing is necessary?
- How can I link data from several repositories together?
- For what kind of tasks can I use MSR? For what kind of tasks should I refrain to use MSR techniques?

Techniques

- How can I leverage the information in software repositories to empirically validate or invalidate hypotheses on software engineering development?
- How can I evaluate the effectiveness of software engineering tools and approaches using software repositories?
- How can I analyze and understand the large quantity of data, in order to monitor the evolution of my system?
- How can I focus Quality Assurance efforts, in the presence of limited resources?

Limitations

- What are the known issues with software repository data that I need to take into account?
- Are there better data sources than conventional software repositories?
- What can I expect from these alternative software

repositories?

To tackle the above questions, we will survey and synthesize the large literature that MSR has spawned in the last decade (and more), starting with early work on software repositories, detailing state of the art techniques, and highlighting the next generation of data sources, and how it has the potential to improve the accuracy of current analyses, open the door to new ones, and to change the evaluation techniques commonly used in MSR.

III. Scope

A. Intended Audience

This tutorial is intended for a variety of people, ranging from project managers, practicitioners, to confirmed researchers and prospective students. All attendees will get an overview of MSR in research and practice, and learn about the current techniques and potential pitfalls of MSR, as well as upcoming techniques. In addition:

- *project managers* will learn how MSR can help in advancing and maintaining their software projects;
- *practitioners* will learn about concrete tools that they can use in their daily activities;
- *researchers* will learn about the methodology involved in empirical studies based on MSR; and
- *prospective students* will learn about open investigation problems that they may want to invest themselves in.

To further all of these goals, we will provide an extensive list of references that they can use to deepen certain topics.

B. Prerequisites

The participants should have a basic background in software engineering and software development. Nonetheless, we designed the tutorial so that no specific previous knowledge in MSR is required.

IV. Teaching Method

The tutorial will employ two *interleaved* teaching methodologies:

1) *Lecturing with slides*. We will present an overview of MSR, as well as detailing MSR techniques and examples, using a standard lecture methodology with slides. This methodology is still unrivalled to transmit a large amount of information in a short amount of time. Printed slides will be provided in advance to the participants, to facilitate following the lectures. In order to break the potential monotony of this style of lecturing, we will involve the audience at regular intervals, as the human attention spans is measured in minutes rather than hours. To achieve this, we will regularly open the floor for questions, and not hesitate to ask questions ourselves to the audience; having short ”quizzes” is a well-known technique to raise the attention of the participants, and let them gauge their performance.
2) *Demonstrating tools*. To give a more interactive and exciting tutorial we will showcase a number of MSR tools, providing concrete examples and application scenarios. This will be interleaved with the lecture, so as to—again—break the potential monotony of a lecture involving slides only. By presenting and demoing working tools, we will also demonstrate the taught MSR techniques in practice, providing the participants with practical skills.

V. Summary of Contents

A. *Mining Software Repositories*

A bit of history. At the first conference on software engineering in 1968 [1] software maintenance was considered a post production activity. The seventies were also the decade in which Software Configuration Management (SCM) emerged as a discipline. In 1975, Rochkind introduced the first SCM, called Source Code Control System (SCCS) [2].

In 1980, Manny Lehman in his seminal work [3], [4] introduced the laws of software evolution. In the meantime, SCM systems continued their growth. In 1982, Tichy introduced Revision Control System (RCS) [5], while four years later the Concurrent Versioning System (CVS) emerged.

It was in the nineties that SCM received widespread attention and usage. With the advent of the Internet and the improvement in network bandwidth, software development started to be distributed. The first bug tracking systems were created: GNATS being the first in 1992, followed by Debbugs in 1994, and Bugzilla in 1998.

In the second half of the nineties, SCMs became so used and popular that researchers started to mine source code repositories. The first approaches were proposed by Ball *et al.* in 1997 to find clusters of files frequently changed together [6], by Graves *et al.* in 1998 to compute the effort necessary for developers to make changes [7] and by Atkins *et al.* in 1999 to evaluate the impact of tools on software quality [8]. These are among the seminal research works where the field of mining software repositories has its roots.

In the first half of the current decade, software repositories received more and more attention by researchers and practitioners. In the meantime, software evolution started to be an active and well-respected research field in software engineering, and mining software repositories matured and started to be a research area on its own. In 2004, the first International Workshop on Mining Software Repositories (MSR) was held [9].

So, what is MSR? Mining Software repositories consists in gathering, modeling, and exploiting the data produced by developers and other stakeholders in the software development process as they create a software system. This data comes from various sources; first and foremost, we have version control archives, such as CVS, SVN and Git; then, issue tracking systems, such as Bugzilla, Jira, or Trac; finally, we also have free-form communication archives, such as development mailing lists, IRC chats, and blog posts.

After showing early MSR contributions as examples to concretize the discourse, we focus on data sources: We detail the format of the most common data sources, and how to process it in ways that allows to build subsequent analyses, in particular on the way the information from distinct repositories can be linked in a coherent whole. After that, we go on the next part of the tutorial, where we present an array of representative MSR techniques, explaining their purpose and how they use the information at hand.

B. *MSR Approaches: the State of the Art*

Based on the information available in software repositories, a variety of studies have been performed, and techniques have been proposed to assist the stakeholders of the development process. We present a selection of MSR approaches, grouped in categories. In each case, we take the stance of explaining both the results of the approaches, but also to explain in great detail *how* the evaluation was performed, as most, if not all, the evaluations performed below made use of software repository data.

Empirical Studies We demonstrate how one can use MSR data to perform empirical studies by way of examples; we then extract guidelines on performing empirical studies with software repositories. The case studies we examine are:

- The seminal study of Mockus *et al.* on open-source software development, investigating Apache and Mozilla [10];
- Tu and Godfrey's study of the evolution of Linux [11];
- The investigation by Bird *et al.* on the effect of distributed development on the defect rate in Windows Vista [12];
- The study by Battacharya and Neamtiu on the impact of programming languages on software defects;
- Callau *et al.*'s study on the usage of dynamic programming language features [13];
- Shihab *et al.*'s empirical study of reopened bugs [14];
- and the empirical study by Posnett *et al.* on the influence of pattern roles on change proneness [15].

Change prediction Change prediction tackles the problem of identifying entities in a software system that are likely to change next. Software repositories act as both a data source and an evaluation device for change prediction approaches; Hassan and Holt's Development Replay approach [16] allows to reliably and repeatably compare the the performance of several change prediction approaches on a large amount of historical data. Approaches using data mining techniques to look for development patterns in the history of the system [17], [18] were found to outperform approaches based on coupling metrics [19]. These approaches are based on change (or logical) coupling, the implicit dependency between two or more software artifacts that have been observed to frequently change together during the evolution of a system [20].

Defect prediction In a world were ressources are limited, managers have to focus QA efforts on parts of a system; they cannot afford to give equal attention to each and every source code file. Defect prediction approaches tackle this scenario. The role of these approaches is to classify files as potentially buggy or non-buggy by considering various attributes of the source code, such as its complexity or its propensity to change.

Software repository data can be used to evaluate the accuracy of these approaches, again by using historical reenactment techniques. The approach in a nutshell revolves in comparing the guesses of a given approach with the actual defects that were reported during a given period of the life of a software system.

Some of the best-performing approaches do make use of a system's history (age of a file, rate of change, number of developers involved) [21], its defect repository [22], or the conversations between developers [23]. Other approaches use static source code metrics only [24]

Moreover, there exist a variety of performance metrics, corresponding to different defect prediction scenarios. The tasks vary from binary classification, to effort-aware ranking [25]

Expertise and Bug assignment. The information in defect repositories can be used to model the knowledge of developers about the systems they work on. This can be used to recommend a specific expert when help is needed over a particular piece of code [26], [27].

A related problem is automatic bug assignment: using information from software repositories, one can choose the person to fix it whith greater accuracy, avoiding or reducing the "tossing effect", when a bug is reassigned to a chain of people before being finally solved [28], [**?**] . Yet another related approach is the automated identification of duplicated bug reports, either to suppress the duplicates, or to combine the information of both bug reports to increase the quality of the resulting report [29]

Code Search. With the massive amount of code freely accessible on the internet, there is a fair probability that a given problem has been solved already, and that an implementation is available somewhere. All that is needed, is to find it. Several source code search engines have been developed, such as CodeGenie [30], or Portfolio [31].

Reuse of source code freely available raises further problems, due to potential incompatibilities with licenses. Several approaches deal with these problematics, determining which license a piece of code possesses, and whether the license threatens its reuse or not [32].

Visual Evolution Analysis Sometimes, the best way to understand the large amount of data present in the history of a software system is to visualize it. We dedicate a part of the tutorial to present a selection of software visualization tools and approaches that handle repository data, and some of the case studies they were evaluated on. The Moose software analysis platform features several evolutionary visualizations based on the Hismo Metamodel such as Chronia [33], CodeCity [34], or SPO [35]. Visualization has been applied to industrial case studies with results, as shown by Telea and Voinea [36], based on their CVSGrab tool [37], [38].

Other visualization approaches focus on visualizing change coupling. These include the EvoGraph visualization approach [39], that combines release history data and source code changes to assess structural stability and recurring modifications, the Evolution Radar [40], and Kiviat diagrams [41].

Human Aspects Software is built by humans, and for humans. As such, taking the human element out of the loop is not advisable. If there is comparatively less work that makes use of developer communication artifacts such as e-mails and other free-form text communication (chats, wikis, blogs), there is still some research interest. We show some of the results that make use of this particular kind of repository [42], [43], [44], [23], [45].

C. Limitations of MSR

If the amount of data available for MSR studies is a boon for empirical research. However, this data comes with strings attached. In this last part, we document the common threats to the validity of empirical studies based on software repositories data. We also go further, and highlight the shortcomings of the current crop of software repositories for empirical research, and demonstrate some of the possible solutions [46].

With the increasing importance of software repositories, researchers observed that software development tools (*e.g.,*, SCM, bug tracking systems, integrated development environment, *etc.*) were not designed to support software evolution analysis and mining. The advent of MSR was a consequence of the widespread adoption of tools such as SCMs and defect tracking systems, which were created before researchers started to mine software repositories [47]

limitations of SCM systems. We argued that existing SCMs, such as CVS, SVN and SourceSafe, are not adequate for software evolution analysis for several reasons [48]: They are file-based instead of entity-based (classes, methods, attributes in object-oriented languages) and thus operations like renaming and refactorings have to be reconstructed with heuristics [49], [50]; they record changes only at commit time, and thus important pieces of information about the development process are lost [51].

limitations of defect repositories. Defect repositories are not immune to issues either; several researchers have cast doubt about the accuracy of the information recorded in them. Aranda and Venolia performed a study of coordination activities around bug fixing from three major product divisions at Microsoft [52]. They first queried the bug database—containing rich bug histories—to extract the people involved in the bug fixes, and then contacted and interviewed them. The authors showed that the information stored in bug repository only is not sufficient, and at times even misleading, to support developers coordination for bug fixing. Other studies have been focused on possible bias in bug fix datasets, and issues in the change to bug linking process [53], [54].

What's next? Considering versioning systems, the alternative we proposed [55] is to record the change activity in the IDE, instead of recovering it based on versioning system commits. We have shown how this improves MSR activities, allowing for instance the fine-grained evaluation of development tools, such as code completion tools [56], and change prediction tools [57]. Other tools use IDE activity data, among them Mylyn [58], [51], and Navtracks [59]. This reflects the opinion of several leaders of the field, that future MSR repositories will be based on tools recording fine-grained activity [46].

For defect tracking systems, researchers analyzed the efficiency of bug tracking systems in providing useful and relevant information that developers can use to understand and fix the reported bugs [60]. Based on these results, researchers proposed methodologies to improve bug tracking systems [61], [62], [63].

[64]

VI. Structure of Contents

We organized the tutorial as follows:

A. Overview of MSR research
   1) What is MSR: The history of MSR and its role in software development and evolution.
   2) A handful of early MSR approaches to animate the discourse.
   3) What are software repositories, and what format do they use. Description of CVS, SVN, Git, Bugzilla, Jira, Trac, and Mailing List.
   4) How to preprocess the data. Issues related to linking accross versions, and accross repositories (defects, and changes; emails and entities; bug-inducing changes and their fixes).

B. MSR Approaches (presented together with their tools, when applicable, and their evaluation methodologies).
   1) Empirical studies that were done using MSR data, and the methodology behind them.
   2) Change prediction approaches: motivation, evaluation methodology, and approaches.
   3) Defect prediction approaches: motivation, evaluation methodology*ies), and approaches.
   4) Expertise and bug assignment.
   5) Code search engines.
   6) Evolution visualization engines and case studies.
   7) Approaches considering human aspects, with a focus on emails.

C. The limitations of MSR, and their potential solutions
   1 Common threats to validity behind all MSR studies.
   2 Limitations of current software repository: noise in the data, loss of information, linking issues, biases, missing types of data.
   3 Proposed solutions to these issues: Alternative sources of information, change-based software evolution, improving defect archives.

D. Conclusions

VII. Constraints/Requirements

We have no specific requirements, save for the usual projector. An extra projector in order to display two screens at the same time could be a plus during the tool demonstrations.

References

[1] P. Naur and B. Randell, *Software Engineering*. NATO, Scientific Affairs Division, Brussels, 1969.

[2] M. J. Rochkind, "The source code control system," *IEEE Transactions on Software Engineering*, vol. 1, no. 4, pp. 364–370, 1975.

[3] M. M. Lehman, "On understanding laws, evolution and conservation in the large program life cycle," *Journal of Systems and Software*, vol. 1, no. 3, pp. 213–221, 1980.

[4] M. Lehman, "Programs, life cycles, and laws of software evolution," *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980.

[5] W. F. Tichy, "Design, implementation, and evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering (ICSE 1982)*. IEEE Computer Society Press, 1982, pp. 58–67.

[6] T. Ball, J.-M. K. Adam, A. P. Harvey, and P. Siy, "If your version control system could talk," in *Proceedings of the ICSE Workshop on Process Modeling and Empirical Studies of Software Engineering*, 1997.

[7] T. L. Graves and A. Mockus, "Inferring change effort from configuration management databases," in *Proceedings of the 5th International Symposium on Software Metrics (METRICS 1998)*. IEEE Computer Society, 1998, pp. 267–273.

[8] D. L. Atkins, T. Ball, T. L. Graves, and A. Mockus, "Using version control data to evaluate the impact of software tools," in *Proceedings of the 21st International Conference on Software Engineering (ICSE 1999)*. ACM, 1999, pp. 324–333.

[9] A. E. Hassan, R. C. Holt, and A. Mockus, "MSR 2004: International workshop on mining software repositories," in *Proceedings of the 26th International Conference on Software Engineering (ICSE 2004)*. IEEE Computer Society, 2004, pp. 770–771.

[10] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 3, pp. 309–346, 2002.

[11] M. W. Godfrey and Q. Tu, "Evolution in open source software: A case study," in *ICSM*, 2000, pp. 131–142.

[12] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does distributed development affect software quality? an empirical case study of windows vista," in *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 2009, pp. 518–528.

[13] O. Callaú, R. Robbes, É. Tanter, and D. Röthlisberger, "How developers use the dynamic features of programming languages: the case of smalltalk," in *MSR*, 2011, pp. 23–32.

[14] E. Shihab, A. Ihara, Y. Kamei, W. Ibrahim, M. Ohira, B. Adams, A. Hassan, and K. Matsumoto, "Predicting re-opened bugs: A case study on the eclipse project," in *Proceedings of the 17th Working Conference on Reverse Engineering*, 2010, pp. 249 –258.

[15] D. Posnett, C. Bird, and P. Devanbu, "An Empirical Study on the Influence of Pattern Roles on Change-Proneness," *Empirical Software Engineering, An International Journal*, pp. 1–28, 2010.

[16] A. E. Hassan and R. C. Holt, "Replaying development history to assess the effectiveness of change propagation tools," *Empirical Software Engineering*, vol. 11, no. 3, pp. 335–367, 2006.

[17] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *Proc. 26th International Conference on Software Engineering (ICSE 2004)*. Los Alamitos CA: IEEE Computer Society Press, 2004, pp. 563–572.

[18] A. Ying, G. Murphy, R. Ng, and M. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Transactions on Software Engineering*, vol. 30, no. 9, pp. 573–586, 2004.

[19] E. Arisholm, L. C. Briand, and A. Føyen, "Dynamic coupling measurement for object-oriented software," *IEEE Trans. Software Eng.*, vol. 30, no. 8, pp. 491–506, 2004.

[20] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *Proceedings of the 14th IEEE International Conference on Software Maintenance (ICSM 1998)*. IEEE Computer Society Press, 1998, pp. 190–198.

[21] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proceedings of the 30th International Conference on Software Engineering (ICSE 2008)*. ACM, 2008, pp. 181–190.

[22] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *MSR*, 2010, pp. 31–41.

[23] A. Bacchelli, M. D'Ambros, and M. Lanza, "Are popular classes more defect prone?" in *FASE*, 2010, pp. 59–73.

[24] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Software Eng.*, vol. 33, no. 1, pp. 2–13, 2007.

[25] T. Mende and R. Koschke, "Effort-aware defect prediction models," in *Proceeding of the 14th European Conference on Software Maintenance and Reengineering (CSMR 2010)*. IEEE Computer Society, 2010, pp. 109–118.

[26] A. Mockus and J. D. Herbsleb, "Expertise browser: a quantitative approach to identifying expertise," in *Proceedings of the 24th International Conference on Software Engineering*. ACM, 2002, pp. 503–512.

[27] T. Gîrba, A. Kuhn, M. Seeberger, and S. Ducasse, "How developers drive software evolution," in *Proceedings of the 8th International Workshop on Principles of Software Evolution (IWPSE 2005)*. IEEE CS Press, 2005, pp. 113–122.

[28] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th International Conference on Software Engineering (ICSE 2006)*. ACM Press, 2006, pp. 361–370.

[29] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proceedings of the 30th International Conference on Software Engineering (ICSE 2008)*. ACM, 2008, pp. 461–470.

[30] O. A. L. Lemos, S. K. Bajracharya, J. Ossher, R. S. Morla, P. C. Masiero, P. Baldi, and C. V. Lopes, "Codegenie: using test-cases to search and reuse source code," in *ASE*, 2007, pp. 525–526.

[31] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: finding relevant functions and their usage," in *ICSE*, 2011, pp. 111–120.

[32] M. D. Penta, D. M. Germán, and G. Antoniol, "Identifying licensing of jar archives using a code-search approach," in *MSR*, 2010, pp. 151–160.

[33] T. Gîrba, M. Lanza, and S. Ducasse, "Characterizing the evolution of class hierarchies," in *Proceedings of the 9th IEEE European Conference on Software Maintenance and Reengineering (CSMR 2005)*. IEEE CS Press, 2005, pp. 2–11.

[34] R. Wettel, M. Lanza, and R. Robbes, "Software systems as cities: a controlled experiment," in *ICSE*, 2011, pp. 551–560.

[35] M. Lungu, M. Lanza, T. Gîrba, and R. Robbes, "The small project observatory: Visualizing software ecosystems," *Sci. Comput. Program.*, vol. 75, no. 4, pp. 264–275, 2010.

[36] A. Telea and L. Voinea, "Case study: Visual analytics in software product assessments," in *VISSOFT*, 2009, pp. 65–72.

[37] L. Voinea and A. Telea, "An open framework for CVS repository querying, analysis and visualization," in *Proceedings of the 3rd International Workshop on Mining Software Repositories (MSR 2006)*. ACM, 2006, pp. 33–39.

[38] ——, "Multiscale and multivariate visualizations of software evolution," in *Proceedings of the 2006 ACM Symposium on Software Visualization (SoftVis 2006)*. ACM, 2006, pp. 115–124.

[39] M. Fischer and H. C. Gall, "Evograph: A lightweight approach to evolutionary and structural analysis of large software systems," in *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006)*. IEEE Computer Society, 2006, pp. 179–188.

[40] M. D'Ambros, M. Lanza, and M. Lungu, "Visualizing co-change information with the evolution radar," *IEEE Trans. Software Eng.*, vol. 35, no. 5, pp. 720–735, 2009.

[41] M. Pinzger, H. Gall, M. Fischer, and M. Lanza, "Visualizing multiple evolution metrics," in *Proceedings of the 2nd ACM Symposium on Software Visualization (SoftVis 2005)*. ACM, 2005, pp. 67–75.

[42] D. Pattison, C. Bird, and P. Devanbu, "Talk and work: A preliminary report," in *Proceedings of the 5th International Working Conference on Mining Software Repositories (MSR 2008)*. ACM, 2008, pp. 113–116.

[43] D. S. Pattison, C. Bird, and P. T. Devanbu, "Talk and work: a preliminary report," in *MSR*, 2008, pp. 113–116.

[44] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts," in *ICSE (1)*, 2010, pp. 375–384.

[45] A. Bacchelli, "Exploring, exposing, and exploiting emails to include human factors in software engineering," in *ICSE*, 2011, pp. 1074–1077.

[46] M. W. Godfrey, A. E. Hassan, J. D. Herbsleb, G. C. Murphy, M. P. Robillard, P. T. Devanbu, A. Mockus, D. E. Perry, and D. Notkin, "Future of mining software archives: A roundtable," *IEEE Software*, vol. 26, no. 1, pp. 67–70, 2009.

[47] J. Estublier, D. Leblang, A. v. d. Hoek, R. Conradi, G. Clemm, W. Tichy, and D. Wiborg-Weber, "Impact of software engineering research on the practice of software configuration management," *ACM Transactions on Software Engineering and Methodology*, vol. 14, no. 4, pp. 383–430, 2005.

[48] R. Robbes and M. Lanza, "Versioning systems for evolution research," in *Proceedings of IWPSE 2005 (8th International Workshop on Principles of Software Evolution)*. IEEE CS Press, 2005, pp. 155–164.

[49] C. Gorg and P. Weisgerber, "Detecting and visualizing refactorings from software archives," in *Proceedings of the 13th International Workshop on Program Comprehension (IWPC 2005)*. IEEE Computer Society, 2005, pp. 205–214.

[50] F. Van Rysselberghe, M. Rieger, and S. Demeyer, "Detecting move operations in versioning information," in *Proceedings of the 10th IEEE European Conference on Software Maintenance and Reengineering (CSMR 2006)*. IEEE Computer Society, 2006, pp. 271–278.

[51] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," in *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT 2006/FSE-14)*. ACM, 2006, pp. 1–11.

[52] J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *Proceedings of the 31st International Conference on Software Engineering (ICSE 2009)*. IEEE CS Press, 2009, pp. 298–308.

[53] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced?: Bias in bug-fix datasets," in *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2009)*. ACM, 2009, pp. 121–130.

[54] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein, "The missing links: bugs and bug-fix commits," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 2010, pp. 97–106.

[55] R. Robbes, "Of change and software," Ph.D. dissertation, 2008.

[56] R. Robbes and M. Lanza, "Improving code completion with program history," *Autom. Softw. Eng.*, vol. 17, no. 2, pp. 181–212, 2010.

[57] R. Robbes, D. Pollet, and M. Lanza, "Replaying ide interactions to evaluate and improve change prediction approaches," in *MSR*, 2010, pp. 161–170.

[58] M. Kersten and G. C. Murphy, "Mylar: A degree-of-interest model for IDEs," in *Proceedings of the 4th International Conference on Aspect-Oriented Software Development (AOSD 2005)*. ACM, 2005, pp. 159–168.

[59] J. Singer, R. Elves, and M.-A. D. Storey, "Navtracks: Supporting navigation in software maintenance," in *ICSM*, 2005, pp. 325–334.

[60] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th International Symposium on Foundations of Software Engineering (FSE 2008)*. ACM, November 2008, pp. 308–318.

[61] S. Just, R. Premraj, and T. Zimmermann, "Towards the next generation of bug tracking systems," in *Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC 2008)*. IEEE Computer Society, 2008, pp. 82–85.

[62] T. Zimmermann, R. Premraj, J. Sillito, and S. Breu, "Improving bug tracking systems," in *Companion to the 31th International Conference on Software Engineering (ICSE Companion 2009)*. IEEE Computer Society, 2009, pp. 247 –250.

[63] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: Improving cooperation between developers and users," in *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work (CSCW 2010)*. ACM, 2010, pp. 301–310.

[64] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429–445, 2005.