Software Analytics in Practice

Dongmei Zhang, Shi Han, Yingnong Dang, Jian-Guang Lou, Haidong Zhang, Microsoft Research Asia, China

Tao Xie, University of Illinois at Urbana-Champaign, USA

Abstract: Software analytics is to utilize data-driven approaches to enable software practitioners to perform data exploration and analysis in order to obtain insightful and actionable information for completing various tasks around software systems, software users, and software development process. In this article, we present how we worked on the StackMine project, a successful software analytic project that produced an analytic system used by and transferred to Microsoft product teams. We also discuss the lessons learned from StackMine on applying software analytics technologies to make practice impact – solving problems that practitioners care about, using domain knowledge for correct data understanding, building prototypes early to get practitioners' feedback, taking scalability and customizability into account, and evaluating analysis results using criteria related to real tasks.

Keywords: Software analytics, Mining software repositories, Technology transfer

Various types of data naturally exists in the software development process, including source code, bug reports, check-in histories, and test cases, etc. Analytics techniques were used on these data sources to help understand software quality in the last few decades. For example, there has been a plethora of work on predicting software reliability in terms of the defect count at different levels of software systems [9]. In recent years, examples of utilizing software artifacts to improve the software development process are software intelligence [8] and analytics for software development [7], both offering software developers pertinent information to support their decision making in the software development process.

In the past decade or so, Internet access has become widely available and software services have played indispensable roles in everyday life. Much richer types of data at much larger scales [10] have started to be available for better understanding how software and services perform in real-world settings, and how end users use them.

Considering the increasing abundance and importance of data in the software domain, we formed the Software Analytics group in Microsoft Research in 2009; and we proposed the concept "software analytics" to expand the previous scope on analyzing software artifacts. As defined in [1,2], software analytics is to utilize data-driven approaches to enable software practitioners to perform data exploration and analysis in order to obtain insightful and actionable information for completing various tasks around software systems, software users, and software development process.

Compared to the huge amount of research conducted in the past years, there have been a small number of reports [12] [13] on the practice impact that software analytics results have created on software development. There are different ways to creating practice impact [15][16][17], e.g., software practitioners could conduct software analytics by themselves; researchers from either the academia or industry could collaborate with software practitioners, either from software companies or the open source communities, and transfer their analytics technologies into real-world use; or some analytics technologies developed by researchers could become popular and organically get adopted by practitioners. No matter which of these different ways is pursued, it still remains a huge challenge today to make practice impact using software analytics technologies.

Being researchers in an industry research lab, we have worked on a number of research projects [3,4,5,6] of software analytics with successful technology transfer and with high impact on industrial practices. In this article, we use the StackMine project [3] as an example to share our lessons learned from these transfers, including (1) the target problems of software analytics should be what practitioners care about; (2) domain knowledge is important for correctly understanding the data under analysis; (3) prototypes should be built early in order to start the feedback loop with practitioners; (4) scalability and customizability are important when putting software analytics techniques into real use; (5) the results of software analytics need to be evaluated using criteria related to the real tasks that practitioners carry out. We hope that our lessons learned could help increase the awareness of both researchers and practitioners on the issues and practices when they try to put software analytics technologies into real use.

The rest of the article is organized as follows. In Section 1, we will introduce the opportunities and challenges of applying software analytics in practice. In Section 2, we will introduce the journey of the example project, StackMine, followed by our experiences and lessons learned from working on this project in Section 3. We conclude the article in Section 4.

1. Opportunities and Challenges

The objective of software analytics is to obtain *insightful* and *actionable* information from software artifacts to help software practitioners accomplish their target tasks around software systems, software users, and software development process [1,2]. Examples of such tasks are identifying performance problems in thousands of performance traces, understanding the usage patterns of a given feature in a software product, and selecting the most relevant test cases to run against a code check-in. *Insightful* information conveys meaningful and useful understanding or knowledge towards performing the target task. Typically, it cannot be easily obtained by direct investigation on the raw data without the aid of analytic technologies. *Actionable* information is information upon which software practitioners can come up with concrete solutions (better than existing solutions if any) towards completing the target task. For example, ranked performance bottlenecks represented by sequences of function calls narrow down the investigation scope of the underlying performance issues, and they also provide guidance on where the investigation effort should be spent. Therefore, the ranked performance bottlenecks are both insightful and actionable.

As shown in Figure 1, software analytics focuses on trinity of software systems, software users, and software development process, with the ultimate goal to help improve software quality, user experience, and development productivity. Note that software development process involves software engineers (including software developers), and analytics on software development process intends to improve development productivity. User experience is positioned from the end-users' perspective, whereas software quality focuses on issues such as reliability, performance, and security. In general, the primary technologies employed by software analytics include large-scale computing (to handle large-scale datasets), analysis algorithms in machine learning, data mining and pattern recognition, etc. (to analyze data), and information visualization (to help with data analysis and presenting insights).

The output of software analytics projects is usually in the form of insights from software artifacts, i.e., insightful and actionable information, or analytic systems. Such analytic systems not only enable software practitioners to obtain insights from software artifacts by themselves but also help software practitioners complete the target tasks. The target audience of software analytics are the broad range of software practitioners, including developers, testers, program managers, software management personnel, designers, usability engineers, service engineers, and support engineers, etc.

There are great opportunities for software analytics to make practice impact because of the following reasons. First, the data sources under study in software analytics come from real-world settings. For example, open-source communities naturally provide a huge data vault of source code, bug history, and check-in information, etc.; and better yet, the vault is active and evolving, which makes the data sources fresh and live. Second, as illustrated by the trinity view, software analytics spreads across the areas of system quality, user experience, and development productivity, indicating a wide scope and huge potential for creating practice impact.

Despite the aforementioned opportunities, there are significant challenges when putting software analytics technologies into real use. For example, how to ensure the output of the analysis results to be insightful and actionable? How do we know whether we are using the data to answer the questions that practitioners are concerned about? How do we evaluate our analysis techniques in real-world settings, etc.? In short, the challenge is how to get "real" when we want to make "real" impact using "real data"?

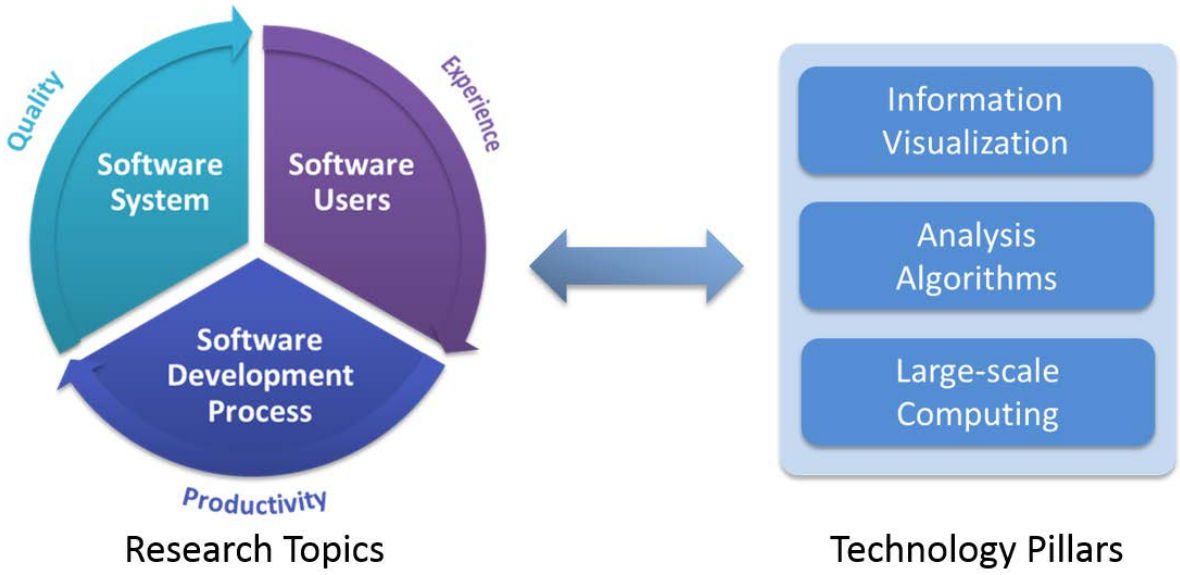


Figure 1. Trinity of software analytics

2. Example Software Analytics Project: StackMine

StackMine targets at mining large-scale performance traces generated by operating systems. In this section, we will first provide an overview of StackMine. Then we will discuss in detail the journey of StackMine starting from how we identified the research problem all the way to the impact it made on large-scale performance analysis.

2.1. Overview

Performance debugging in the large [3] has recently emerged due to available infrastructure support to collect execution traces with performance issues from a huge number of users at the deployment sites. One such example infrastructure support at Microsoft is PerfTrack[1], which measures system responsiveness to user actions on operating systems. For example, let's consider the scenario in which a user clicks a menu item of Windows Explorer to create a folder. PerfTrack measures how long it takes for the user to receive response from the system upon clicking. If the response time exceeds a pre-defined threshold, PerfTrack sends execution traces (containing callstacks collected during the preceding time interval back) to Microsoft for debugging. All such collected traces could contain more than 1 billion callstacks in total, which are far beyond the affordable investigation effort of performance analysts.

Performance debugging in the large can be reduced to a software analytic problem: given enormous callstacks collected at the deployment sites, how can analytic technologies help performance analysts effectively discover highly impactful performance bugs (e.g., bugs impacting many users with long response delay)? To tackle this software analytic problem, we developed StackMine [3], a novel approach and an implementation in the form of a scalable system for postmortem performance debugging. The approach includes a novel costly-pattern mining and clustering mechanism to reduce investigation effort of performance analysts.

2.2. Identifying the essential research problem

Solving essential problems is one of the key factors to the success of StackMine, and it should be among the most important guidelines for conducting successful software analytic projects. At the beginning of the StackMine project, however, we did not get this right.

[1] http://channel9.msdn.com/Blogs/Charles/Inside-Windows-7-Reliability-Performance-and-PerfTrack

Before the StackMine project was started, during one of the meetings with us (i.e., members from the Software Analytics group of Microsoft Research Asia), a member of a Microsoft product team introduced their existing tools and state of practice in inspecting a single stream of system-event traces for Windows performance analysis, as well as the challenges faced by them on inspecting a large number of trace streams. Then we went on the usual research route of looking into the research literature for existing work to understand the state-of-the-art techniques on OS-performance analysis. We found that OS-performance analysis based on a large set of real-world trace data was an emerging topic with little published literature. The finding was exciting because of the potential opportunity to pursue pioneer research. As researchers with machine learning background, we started the project the way we were familiar with for machine learning problems. Based on our reading of the existing analysis records documented by the Windows performance team and our discussion with one Windows performance analyst, we quickly decided to study the high-CPU-consumption symptoms exhibited in the traces. We then formulated the problem detection and the problem-correlation analysis into a classification problem and a clustering problem (both at the granularity of trace streams), respectively. We spent a few months along this direction, and we did get promising results from machine-learning perspectives, e.g., high accuracy for classification, high purity and completeness for clustering.

However, after we presented the promising results to the Windows performance analysts, besides some polite words such as “interesting” and “good”, we did not see much enthusiasm from the analysts to continue along this direction. Then from their further feedback, we learned that we were not solving the essential problems that they cared about for three reasons. First, we missed the most important problem category Wait representing delays due to synchronization, resource contention, and power state, etc. Second, automatic detection and localization of high-CPU-consumption intervals were of little help to solve their real problems. Such tasks of identifying Region-Of-Interest (ROI) should be and could be well handled by instrumentation together with reliable domain-specific heuristics. Third, trace stream was not the right granularity level for similarity modeling and clustering, because it is too high-level to help the performance analysts locate the root causes.

It was then when we decided to reset the project and focus on identifying the essential problems to solve. We worked closely with the Windows performance analysts on this; and we worked with them throughout the entire project lifecycle since then. In the end, we decided to target StackMine at identifying highly impactful program-execution patterns from a large number of trace streams. We selected stack patterns as the pattern representation, and we formulated the pattern-discovery problem as a mining and clustering problem against millions of stack traces.

2.3. Understanding the data with domain knowledge

Deep domain knowledge is embedded in the trace streams. There were 400+ different event types that could appear in a trace stream. It was neither affordable nor necessary for us to study all of them for StackMine. We consulted the analysts with our basic understanding of the performance analysis problem, and started with 10+ key event types. After we presented our initial algorithms based on these key event types, the analysts pointed out the missing ones that could compromise the correctness of the algorithms. We then started to learn those additional key event types and revised our algorithms accordingly. Overall, it took us more than one week to learn the corresponding domain knowledge behind these event definitions.

Domain knowledge can often help scope the relevant data for study, especially when the data scale is large. Given a time interval of a trace stream, we first mined and clustered stack patterns against all the CPU-sampling events and the thread-context-switch events. The computation was expensive and the result was not satisfactory. To address the issues, we later designed appropriate ROI-extraction algorithms [3] to properly scope the subsequent analysis while remarkably reducing the computational cost. We would not have come up with such algorithms without fully understanding how the data is related to the targeted problem. We gained such understanding from observing the existing practices of the analysts.

It is always important to remove noise in data. In StackMine, we used *coverage of performance bottlenecks* to measure the effectiveness of identified patterns. Higher coverage would indicate lower probability for high-impact bugs to remain undiscovered. The coverage was defined as the ratio of pattern-affected time periods to the total time periods of all ROIs. Our first coverage result was extremely low, regardless of how we improved the algorithms to discover more patterns. Then we found out that a number of abnormally-long-duration trace streams badly impacted

the coverage result. These trace streams were verified to be collected from computers that had experienced long sleeping, i.e., the trace streams started before a long sleep and ended after the wakeup. We got reasonable coverage result after filtering out these trace streams.

2.4. Building iterative feedback loop

The research and development of analytics technologies is usually an iterative process, in which getting timely feedback from the target audience is important. We certainly benefited from the feedback loop we constructed early on when StackMine started.

The first example is how to define the similarity model for clustering callstack patterns. A callstack pattern is a subsequence of function calls shared by a set of callstacks. We used callstack patterns to represent performance issues. Typically, one performance issue may exhibit multiple callstack patterns, which share identical parts of bottlenecks and vary in minor parts. Therefore, callstack patterns should be clustered in order to better prioritize performance bottlenecks with aggregated performance metrics. We first used the generic sequence-edit-distance model for clustering since it was a natural choice. Some of the resulted clusters were huge; and they included many different performance bottlenecks. Furthermore, some highly impactful performance bottlenecks were not clustered; and they were distributed across many small clusters. We reviewed the results with the analysts and we quickly identified the limitation in the similarity model: it could not reflect the true relationships of callstack patterns without appropriate domain knowledge. Then we started to design the similarity model incorporating the essential domain knowledge. After a few iterations of experiments and discussion, we obtained an appropriate similarity model with satisfactory clustering results.

The second example is how we selected the evaluation criteria for StackMine. When we presented the preliminary results (a ranked list of callstack-pattern clusters) to the analysts, we used purity and completeness metrics as the evaluation criteria due to common research practice. However, the analysts could not relate to these metrics; and they cared about the coverage of performance bottlenecks provided by the top pattern clusters. We learned that their primary interest was to quickly achieve high coverage against a given trace-stream set. High coverage would indicate that callstack patterns from top clusters could identify and explain most of the performance bottlenecks captured in the trace-stream set. Based on this feedback, we designed the performance-bottleneck-coverage metric to measure the callstack-pattern clusters [3].

The third example is how we used StackMine to help analysts and continuously improved the algorithms at the same time. When analyzing a new dataset, the analysts first used StackMine to obtain the most impactful performance bottlenecks from the top clusters. Then they quickly went through the top clusters to verify the performance bottlenecks, and noted down their comments or feedback. After using StackMine, they continued their analysis with their traditional workflow to study the uncovered parts of the trace streams. Leveraging the already-analyzed datasets and the comments and feedback as partial labels, we could improve our existing algorithms. This way, we as researchers would not be blocked or limited by lacking of labeled data. Meanwhile, the practitioners could also benefit from the improved solutions. This interactive and iterative model helped StackMine make steady progress along the way.

2.5. Making impact

StackMine was resulted from the two-year effort of continuous development and improvement. In December 2010, StackMine was first applied in performance-debugging activities at a Microsoft team for performance analysis. Performance analysts in the team utilized StackMine to analyze hundreds of millions of callstacks; and they got substantial benefits (90%) in terms of reducing human investigation effort in their analysis as stated in a statement from the Microsoft team: “We believe that the StackMine tool is highly valuable and much more efficient for mass trace streams (100+ trace streams) analysis. For 1000 trace streams, we believe the tool saves us 4-6 weeks of time to create new performance signatures (representations of performance issues), which is quite a significant productivity boost.”

3. Lessons Learned on Applying Software Analytics

In this section, we summarize the lessons learned from working on StackMine that can help address the challenges in applying software analytics in practice.

3.1. Identifying essential problems

Various types of data are incredibly rich in the software domain, and the scale of data is significantly large. It is often not difficult to grab some datasets, apply certain data analysis techniques, and obtain some observations. However, these observations, even with good evaluation results from the data analysis perspective, may not be useful for accomplishing the target tasks of practitioners. We made such a mistake at the beginning of StackMine. Similar to GQM [14], it is important to first identify essential problems for accomplishing the target task in practice, and then obtain the right data sets suitable to help solve the problems. These essential problems are those solving which would substantially improve the overall effectiveness of tackling the task, such as improving software quality, user experience, or practitioner productivity.

As demonstrated by StackMine, one way to identify essential problems is through close collaboration between researchers and practitioners, interactively and iteratively. Sometimes, practitioners may not be able to articulate the essential problems since they are emerged in various challenges in their daily work. Researchers would need to work in an agile way to construct a quick feedback loop in order to identify the essential problems at an early stage. Such quick feedback loop helps avoid detour or waste of time in the course of a software analytics project.

3.2. Domain semantics should be correctly understood for data preparation

Software artifacts often carry semantics specific to the software domain; therefore, they cannot be simply treated as generic data such as text and sequences. For example, callstacks are sequences with program execution logic, and bug reports contain relational data and free text describing software defects, etc. Understanding the semantics of software artifacts is a prerequisite for analyzing the data later on. StackMine may be an extreme example of this – there was a deep learning curve for us to understand the performance traces before we could conduct any analysis.

In practice, understanding data is three-fold: data interpretation, data selection, and data filtering. To conduct data interpretation, researchers need to understand basic definitions of domain-specific terminologies and concepts. To conduct data selection, researchers need to understand the connections between the data and the problem being solved. To conduct data filtering, researchers need to understand defects and limitations of existing data to avoid incorrect inference. We next explain the preceding three aspects in more details.

Data Interpretation. Researchers would need to be equipped with essential domain knowledge about data, including domain-specific terms, concepts, and principles, etc. Typically, researchers do not have such knowledge at the beginning of a software analytics project, and they usually ask practitioners for such information. It is common that such knowledge is scattered among practitioners and rarely documented; and few practitioners know which portion would be important for solving the problem. Therefore, based on our experience, we would suggest to learn or transfer the data knowledge using a demand-driven way. It could be driven by either researchers or practitioners in an interactive and iterative fashion.

Data Selection. Some data might be irrelevant with respect to solving the underlying problem. Researchers often need to select the appropriate subsets in order to conduct effective and efficient analysis. In addition to the knowledge about data, the experiences and skills of practitioners are also valuable for researchers to select to scope the data. Too large a scope might still incur high computational cost and introduce noise in the analysis results; while too small a scope might miss important information for getting the correct the result.

Data Filtering. Data might contain defects that could lead to incorrect analysis results. Example defects are data points with abnormal values some of which may indicate untrue situations. Such data defects might be difficult to detect. Researchers should review and examine the analysis results, and conduct data filtering to eliminate or reduce the impact of data defects.

3.3. A usable system should be built early to enable the feedback loop between researchers and practitioners

It is an iterative process to create software analytics solutions to solve essential problems in practice. Therefore, it is much more effective to build a usable system early on in order to start the feedback loop with the software practitioners. The feedback is often valuable for formulating research problems and researching appropriate analysis algorithms. In addition, software analytics projects can benefit from early feedback in terms of building trust between researchers and practitioners, as well as enabling the evaluation of the results in real-world settings.

In particular, software analytic tasks may have different kinds of outcomes. To provide end-to-end solutions, the outcomes typically include an analysis system, such as the StackMine system, with researchers as solution providers and practitioners as end users. Such systems should have at least two desirable characteristics: effective and easy-to-use. If it is not effective, i.e. it cannot help practitioners solve their problem, it will not be used. If it is difficult to use, it is unlikely to be adopted.

3.4. Scalability and customization are usually required for analytics solutions

Due to the scale of data in the real-world settings, scalable analytic solutions are often required to solve essential problems in practice. In fact, scalability may directly impact the underlying analysis algorithms used to solve the problems. Customization is another common requirement to incorporate domain knowledge due to the variations of software and services. The effectiveness of solution customization in analytics tasks can be summarized as (1) filtering noisy and irrelevant data, (2) specifying between data points their intrinsic relationships that cannot be derived from the data itself, (3) providing empirical and heuristic guidance to make the algorithms robust against biased data. The procedure of solution customization would be typically conducted in an iterative fashion via close collaboration between researchers and practitioners.

StackMine demonstrated the importance of scalability and customization. Without the distributed computing system, we would not have made it possible to handle the scale of callstacks Windows team analyzed. Without customization, it would not have been possible for the Windows analysts to incorporate their knowledge when using StackMine.

3.5. Evaluation criteria should be tied with real tasks in practice

Because of the natural connection with practice, software analytics projects should be (at least partly) evaluated using the real tasks that they are targeted to help with. Common evaluation criteria of data analysis, such as precision and recall, can be used to measure intermediate results. However, they are often not the only set of evaluation criteria when real tasks are involved. For example, in StackMine, we used the coverage of detected performance bottlenecks to evaluate our analysis results, which was directly related to the analysis tasks of Windows analysts. When conducting evaluation in practice with practitioners involved, researchers need to be aware of and cautious about the evaluation cost and benefit incurred on practitioners.

In particular, there are two aspects of a typical evaluation of a solution to the target problem: (1) what to evaluate – the evaluation criteria can be measurement of the effectiveness/efficiency of the solution, or comparison of the solutions to multiple alternatives; (2) how to evaluate - the evaluation activities to obtain the metric values of the evaluation criteria. For software analytics tasks, researchers need to select evaluation criteria based on practical needs, and design evaluation activities with little distraction from practitioners on their normal work.

Practical evaluation criteria are important for measuring the ultimate effectiveness/efficiency of a software analytics solution, particularly in helping practitioners with their daily work. These practical evaluation criteria might be quite different from traditional evaluation criteria of classic machine-learning/data-mining tasks. The latter are typically for measuring the effectiveness of intermediate steps, e.g., accuracy in classification tasks, precision and recall in information-retrieval tasks, purity and completeness in clustering tasks. In contrast, practical evaluation criteria are typically designed based on or originated from practitioners' existing practice in their daily work.

Evaluation criteria are typically in the form of a set of metrics calculated by corresponding objective functions. Based on the objective functions, optimal solutions are devised via optimizing the objective functions and comparing multiple alternative solutions. Among a set of objective functions, priority may exist, and the highest-

prioritized one should relate to the top interest. In other words, the highest-prioritized evaluation metric should reflect practitioners' biggest concern.

4. Conclusion

Software analytics is to utilize data-driven approaches to enable software practitioners to perform data exploration and analysis in order to obtain insightful and actionable information for completing various tasks around software systems, software users, and software development process. In this article, we have presented how we worked on the StackMine project, a successful software analytic project that produced an analytic system used by and transferred to Microsoft product teams. We have also discussed the lessons learned from StackMine on applying software analytics technologies to make practice impact – solving problems that practitioners care about, using domain knowledge for correct data understanding, building prototypes early to get practitioners' feedback, taking scalability and customizability into account, and evaluating analysis results using criteria related to real tasks.

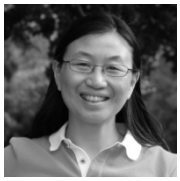
Acknowledgment. The authors would like to thank the engineers from the Microsoft product teams for the collaboration on the StackMine project and all other collaboration projects with the Software Analytics group of Microsoft Research Asia.

Reference

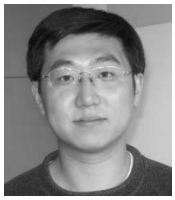
1. D. Zhang, Y. Dang, J.-G. Lou, S. Han, H. Zhang, and T. Xie. Software Analytics as a Learning Case in Practice: Approaches and Experiences. In *Proc. MALETS 2011*, 2011.
2. D. Zhang and T. Xie. Software Analytics in Practice: Mini Tutorial. In *Proc. ICSE 2012 SEIP Track*, Mini Tutorial, pages 997, 2012.
3. S. Han, Y. Dang, S. Ge, D. Zhang, and T. Xie. Performance Debugging in the Large via Mining Millions of Stack Traces. In *Proc. ICSE 2012*, pages 145-155, June 2012.
4. Y. Dang, D. Zhang, S. Ge, C. Chu, Y. Qiu, and T. Xie. XIAO: Tuning Code Clones at Hands of Engineers in Practice. In *Proc. ACSAC 2012*, 2012.
5. R. Ding, Q. Fu, J.-G. Lou, Q. Lin, D. Zhang, J. Shen, and T. Xie. Healing Online Service Systems via Mining Historical Issue Repositories. In *Proc. ASE 2012*, pages 318-321, 2012.
6. Q. Fu, J.-G. Lou, Q.-W. Lin, R. Ding, Z. Ye, D. Zhang, and T. Xie. Performance Issue Diagnosis for Online Service Systems. In *Proc. SRDS 2012*, 2012.
7. R. P.L. Buse, T. Zimmermann. Information Needs for Software Development Analytics. In *Proc. ICSE 2012 SEIP Track*, pages 987-996, 2012.
8. A. E. Hassan and T. Xie. Software Intelligence: Future of Mining Software Engineering Data. In *Proc. FSE/SDP FoSER 2010*, pages 161-166, 2010.
9. Todd L. Graves, Alan F. Karr, J. S. Marron, and Harvey Siy. Predicting Fault Incidence Using Software Change History. IEEE Trans. Softw. Eng. 26, 7, pages 653-661, 2000.
10. K. Glerum, K. Kinshumann, S. Greenberg, G. Aul, V. Orgovan , G. Nichols, D. Grant, G. Loihle, G. Hunt. Debugging in the (Very) Large: Ten Years of Implementation and Experience. In *Proc. SOSP*, pages 103-116, 2009.
11. D. Fisher, R. DeLine, M. Czerwinski, S. Drucker, Interactions with Big Data Analytics. *Interactions*, 19(3):50-59, May 2012.
12. J. Czerwonka, R. Das, N. Nagappan, A. Tarvo, A. Teterev. CRANE: Failure Prediction, Change Analysis and Test Prioritization in Practice - Experiences from Windows. In *Proc ICST*, pages 357-366, 2011.
13. Emad Shihab, Ahmed E. Hassan, Bram Adams, Zhen Ming Jiang: An industrial study on the risk of software changes. In Proc. SIGSOFT FSE 2012, pages 62.
14. Victor R. Basili, Gianluigi Caldiera, H. Dieter Rombach. The Goal Question Metric Approach, Chapter in *Encyclopedia of Software Engineering*, Wiley, 1994.

15. Tony Gorschek, Per Garre, Stig Larsson, and Claes Wohlin. A Model for Technology Transfer in Practice. *IEEE Softw*. 23, 6 (November 2006), pages 88-95, 2006.
16. Anna Sandberg, Lars Pareto, and Thomas Arts. Agile Collaborative Research: Action Principles for Industry-Academia Collaboration. *IEEE Softw*. 28, 4, 74-83, 2011.
17. C. Wohlin, A. Aurum, L. Angelis, L. Phillips, Y. Dittrich, T. Gorschek, H. Grahn, K. Henningsson, S. Kagstrom, G. Low, P. Rovegard, P. Tomaszewski, C. van Toorn, J. Winter. The Success Factors Powering Industry-Academia Collaboration. IEEE Software, 29, 2, pp. 67-73, 2012.

About the Authors:



Dongmei Zhang is a Principal Researcher in Microsoft Researcher Asia (MSRA), Beijing, China. She is also the founder and manager of the Software Analytics group of MSRA. Her research interests include data-driven software analysis, machine learning, information visualization and large-scale computing platform. Zhang received a PhD from the Robotics Institute, Carnegie Mellon University. Contact her at dongmeiz@microsoft.com.

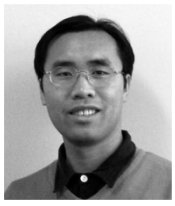


Shi Han is a Researcher in the Software Analytics group of Microsoft Researcher Asia, Beijing, China. His research interests include software analytics, particularly software performance analysis by leveraging machine learning and data mining technologies. Han received a MS in computer science from Zhejiang University, China. Contact him at shihan@microsoft.com.

Yingnong Dang is a Lead Researcher in the Software Analytics group of Microsoft Researcher Asia, Beijing, China. His research interests include software engineering, software analytics, data analytics, and human-computer interaction. Dang received a PhD in control science and engineering from the Xi'an Jiaotong University, Xi'an, China. He is a member of ACM, and a member of the Software Engineering Technical Committee of the China Computer Federation. Contact him at yidang@microsoft.com.

 **Jian-Guang Lou** is a Lead Researcher in the Software Analytics group at Microsoft Research Asia. His research interests include performance analysis and diagnosis of online services, data mining for software engineering. Lou received a PhD in pattern recognition from the National Laboratory of Pattern Recognition at the Institute of Automation of the Chinese Academy of Sciences. He is a member of IEEE and ACM. Contact him at jlou@microsoft.com.

 **Haidong Zhang** is a Principal Software Architect in the Software Analytics group of Microsoft Research Asia, Beijing, China. His research interests include software analytics and information visualization. Zhang received a PhD in computer science from Peking University, China. Contact him at haizhang@microsoft.com.

 **Tao Xie** is an Associate Professor in the Department of Computer Science at University of Illinois at Urbana-Champaign. His research interests include software engineering, particularly software testing, program analysis, and software analytics. Xie received a PhD in computer science from the University of Washington at Seattle. He's a senior member of IEEE and ACM. Contact him at taoxie@acm.org.

Complete contact information

Dongmei Zhang
T2, No.5 Danling Street, Haidian District
Beijing, P.R. China 10080
Email: dongmeiz@microsoft.com
Phone: +86 10-5917-5248

Shi Han
T2, No.5 Danling Street, Haidian District
Beijing, P.R. China 10080
Email: shihan@microsoft.com
Phone: +86 10-5917-5801

Yingnong Dang
T2, No.5 Danling Street, Haidian District
Beijing, P.R. China 10080
Email: yidang@microsoft.com
Phone: +86 10-5917-5683

Jian-Guang Lou

T2, No.5 Danling Street, Haidian District
Beijing, P.R. China 10080
Email: jlou@microsoft.com
Phone: +86 10-5917-3207

Haidong Zhang
T2, No.5 Danling Street, Haidian District
Beijing, P.R. China 10080
Email: haizhang@microsoft.com
Phone: +86 10-5917-5233

Tao Xie
Department of Computer Science
University of Illinois at Urbana-Champaign
3116 Siebel Center
201 N. Goodwin Ave.
Urbana, IL 61801, USA
Email: taoxie@acm.org
Phone: 919-655-8894