Software Analytics as a Learning Case in Practice: Approaches and Experiences

Dongmei Zhang[1], Yingnong Dang[1], Jian-Guang Lou[1], Shi Han[1], Haidong Zhang[1], Tao Xie[2]
[1]Microsoft Research Asia, Beijing, China
[2]North Carolina State University, Raleigh, NC, USA
{dongmeiz,yidang,jlou,shihan,haizhang}@microsoft.com, xie@csc.ncsu.edu

ABSTRACT

Software analytics is to enable software practitioners to perform data exploration and analysis in order to obtain *insightful* and *actionable* information for data-driven tasks around software and services. In this position paper, we advocate that when applying analytic technologies in practice of software analytics, one should (1) incorporate a broad spectrum of domain knowledge and expertise, e.g., management, machine learning, large-scale data processing and computing, and information visualization; and (2) investigate how practitioners take actions on the produced information, and provide effective support for such information-based action taking. Our position is based on our experiences of successful technology transfer on software analytics at Microsoft Research Asia.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Debugging aids*, *diagnostics*; D.2.9 [**Software Engineering**]: Management—*Productivity, software quality assurance (SQA)*

General Terms

Experimentation, Management, Measurement

Keywords

Software analytics, machine learning, technology transfer

1. INTRODUCTION

A huge wealth of various data exists in the software development process, and hidden in the data is information about the quality of software and services as well as the dynamics of software development. With various analytic technologies (e.g., data mining, machine learning, and information visualization), *software analytics* is to enable software practitioners[1] to perform data exploration and analysis in order to obtain *insightful* and *actionable* information for data-driven tasks around software and services[2].

Insightful information is information that conveys meaningful and useful understanding or knowledge towards performing the target task. Typically insightful information is not easily attainable by directly investigating the raw data without 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.

Developing a software analytic project typically goes through iterations of the life cycle of four phases: task definition, data preparation, analytic-technology development, and deployment and feedback gathering. Task definition is to define the target task to be assisted by software analytics. Data preparation is to collect data to be analyzed. Analytic-technology development is to develop problem formulation, algorithms, and systems to explore, understand, and get insights from the data. Deployment and feedback gathering involves two typical scenarios. One is that, as researchers, we have obtained some insightful information from the data and we would like to ask domain experts to review and verify. The other is that we ask domain experts to use the analytic tools that we have developed to obtain insights by themselves. Most of the times it is the second scenario that we want to enable.

Among various analytic technologies, machine learning is a well-recognized technology for learning hidden patterns or predictive models from data. It plays an important role in software analytics. In this position paper, we argue that when applying analytic technologies in practice of software analytics, one should

- incorporate a broad spectrum of domain knowledge and expertise, e.g., management, machine learning, large-scale data processing and computing, and information visualization;
- investigate how practitioners take actions on the produced insightful and actionable information, and provide effective support for such information-based action taking.

Our position is based on a number of software analytic projects that have been conducted at the Software Analytics (SA) group[3] at Microsoft Research Asia (MSRA) in recent years (in the rest of this paper, we refer to members of the software analytic projects at MSRA as the SA project teams). These software analytic projects have undergone successful technology transfer within Microsoft for enabling informed decision making and improving quality of software and services. We expect that our position will provide useful

[1]Software practitioners typically include software developers, testers, usability engineers, and managers, etc.

[2]We coin this term *software analytics* to expand the scope of previous work [1, 6] on analytics for software development and previous work on software intelligence [5].

[3]`http://research.microsoft.com/groups/sa/`

MALETS '11, November 12, 2011, Lawrence, Kansas, USA

guidelines for other researchers and practitioners in successfully carrying out software analytics research for technology transfer.

2. PROJECT OBJECTIVES

The main objectives of MSRA software analytic projects are to advance the state of the art in software analytics and help improve the quality of Microsoft products as well as the productivity of Microsoft product development. The target customers (in short as customers) are primarily practitioners from Microsoft product teams. Such emphasis also benefits the broader research and industrial communities because Microsoft product development belongs to important representatives of modern software development, along dimensions such as the team scale, software complexity, and software types. Some of the MSRA project outcomes could also eventually reach out to broad communities through their integration with Microsoft development tools. For example, the code-clone detection tool [2] resulted from the the first example project discussed in Section 4 has been integrated in Microsoft Visual Studio vNext, benefiting broader communities.

3. CHALLENGES

There are two main categories of challenges to overcome in order to achieve the stated objectives. The first category is rooted from the characteristics of the data being analyzed with analytic technologies.

Data scale. Typical data in software analytics is of large scale, e.g., due to the large scale of software being developed and the large size of software development teams. Some tasks require to analyze division-wide or even company-wide code bases, which are far beyond the scope of a single code base (e.g., when conducting code-clone detection [2]). Some tasks require to analyze a large quantity of (likely noisy) data samples within or beyond a single code base (e.g., when conducting runtime-trace analysis [3]). Although lacking data samples may not be an issue in this context of machine learning, the large scale of data poses challenges for data processing and analysis, including learning-algorithm design and system building.

Data complexity. Typical data in software analytics is of high complexity, which is partly due to the high complexity of software being developed. For example, runtime traces from distributed systems [3] need to be correlated, while traces from multiple threads [7] need to be split. System logs [3] include unstructured textual information. There could be high dependencies across traces and noises among traces. In addition, real-world usage data produced from in-field operations offers substantial opportunities for various tasks such as debugging (e.g., those assisted by the Microsoft Error Reporting system [4]). In addition to high complexity, such data is typically distributed and often partial (e.g., collected with sampling-based techniques to reduce runtime overhead). All these characteristics pose challenges for analytic technologies such as machine learning.

The second category is rooted from the characteristics of the tasks being assisted by software analytics.

Focus on ultimate tasks being assisted. Among tasks assisted by software analytics, some tasks are intermediate tasks and some are ultimate tasks. Usually intermediate tasks produce information toward solving ultimate tasks. For example, code-clone detection is considered as an intermediate task, which produces information towards refactoring and defect detection that are ultimate tasks. Such focus on ultimate tasks requires the mandatory inclusion of the phase of deployment and feedback gathering in the life cycle of a software analytic project. Unlike most previous research on code-clone detection, we should not stop at measuring the precision and recall of detected clones; rather, we should push further to accomplish that the detected clones could effectively help address ultimate tasks such as refactoring and defect detection, and should measure such benefits in evaluations.

Engagement of customers during the development process of a software analytic project. It is well recognized that engaging customers is a challenging task especially in the context of software engineering tools. Customers may have resistance to proposed changes (due to analytic-tool adoption) on their existing way of carrying out a task. In addition, due to tight development schedule, they may not be able to invest time on gaining understanding of the best/worst scenarios for applying an analytic tool. However, developing a software analytic project typically needs the engagement of customers in iterations of the four phases in the project life cycle, e.g., to get better understanding on the tasks and domain knowledge. Among the phases, especially the phase of deployment and feedback gathering, it is crucial for the produced analytic tools to have good usability, e.g., providing effective visualization and manipulation of analysis results.

4. EXPERIENCES

We next discuss our experiences in different phases of developing two example MSRA analytic tools that have been successfully adopted at various Microsoft product teams. XIAO [2] is a tool for detecting code clones of source-code bases, targeting at tasks such as refactoring and defect detection. StackMine is a tool for learning performance bottlenecks from call-stack traces collected from real-world usage, targeting at tasks such as performance analysis.

4.1 Task Definition

Task definition is to define the target task to be assisted by software analytics. There are two models (or their mixture) of initiating a software analytic project at MSRA: the *pull* model and *push* model.

StackMine primarily follows the pull model. Before the StackMine project was started, during one of the meetings with the SA team, a member of a Microsoft product team talked about their state of practice in inspecting a single stream of stack traces[4] for performance analysis, as well as the challenges that they were facing on inspecting a large number of trace streams. Then the SA project team started the StackMine project to address the (most urgent) need of the target customers. Typically, projects of this pull model may more easily fit in the workflow of the target product team, facilitating the integration of the analytics tools in the product team's activities.

XIAO primarily follows the push model. Based on the research literature and initial investigations of some Microsoft code bases, the SA project team gained insights and realized the existence of defects related to code clones especially near-miss clones, but did not know their extent. Then the SA project team developed the XIAO prototype and demonstrated the prototype to various Microsoft product teams to "sell" the solution to them. Through iterations of interactions with product teams, the SA project team concretized the details of the target tasks of refactoring and defect detection.

4.2 Data Preparation

Data preparation is to collect data to be analyzed. For data preparation, there are two types of infrastructure supports: *existing ones*

[4]One stream of stack traces corresponds to one usage scenario that exhibits performance issues.

in industry and *in-house* ones.

The data preparation of StackMine primarily relies on the existing Microsoft infrastructure support. In particular, StackMine relies on the Event Tracing for Windows (ETW), which has been included in Microsoft Windows 2000 and later. Often the time, existing infrastructures are designed to collect data for various purposes. If these infrastructures provide insufficient support for a specific analytic task, it may be difficult or take relatively long time for the infrastructure development team to accommodate feature requests.

The data preparation of XIAO primarily relies on in-house code-analysis front end. For in-house infrastructure support, since the infrastructure development is under the control of the SA project team, it is relatively easy to improve infrastructure support to satisfy the need of software analytic projects. For example, parsers based on abstract syntax tree (AST) were initially considered for XIAO's analysis front end. In the end, to parse source code from heterogonous compilation environments, a token-based parser was developed as XIAO's analysis front end.

4.3 Analytic-Technology Development

Analytic-technology development is to develop problem formulation, algorithms, and systems to explore, understand, and get insights from the data. Due to the large scale of the data being analyzed, analytic technologies such as machine learning techniques need to be scalable. The realization of scalability includes both the design and implementation of analytic technologies.

The SA project team needs to acquire deep knowledge about the data (including its format and semantics) and often the time this acquirement process may be non-trivial. For example, for StackMine, the SA project team needs to learn the format and semantics of ETW traces via reading the ETW documentations and trace annotations as well as consulting with the customers.

The SA project team needs to acquire good understanding of target tasks. For example, for StackMine, it is important to understand how performance analysts (the customers) currently conduct performance analysis on individual ETW traces. Such understanding could help define what analysis results the technologies need to provide. Acquiring such understanding requires intensive interactions with performance analysts. For XIAO, it is important to understand what types of clones would be better for a specific target task. For example, for refactoring, exactly-match clones are preferable over near-miss clones, while for defect detection, near-miss clones are preferable over exactly-match clones.

The SA project team needs to acquire domain or task knowledge from customers and this acquirement process is often challenging. One main reason is that the customers may not be able to effectively identify, abstract, or articulate the domain or task knowledge required by the SA project team. The customers understand and use such knowledge in their daily work, and they could respond to judge given concrete cases but have difficulties to articulate the knowledge to others as general rules. For example, for StackMine, it is important for the SA project team to realize some function calls such as a system function call `SendMessage` are typically expensive and need to be filtered out from the analysis results to better assist performance analysis. For XIAO, it is important for the SA project team to realize that clones occurring at debugging statements such as long-printing and assertion statements are typically not useful for the target tasks. XIAO needs to have built-in filtering mechanisms to filter them out from the analysis results to better assist refactoring or defect detection.

Due to the complexity of the target tasks or the scale of the data being analyzed, often the time there may not be off-the-shelf learning algorithms or implementations that could work well with the scale of the data or the target tasks. In contrast to academic projects on software analytics (which often do not include effort for technology transfer), SA project teams may need to develop new learning algorithms or implementations. For example, for StackMine, no single existing learning algorithm is capable of providing the desired analysis results for the target tasks. For the target tasks, the SA project team composed a frequent sequence mining algorithm and a clustering algorithm. Even for each of these two algorithms being composed, no existing implementations are available to handle the scale of the data being analyzed. The SA project team implemented these two algorithms with desired scalability. For XIAO, no existing algorithm is sufficient for the target tasks. The SA project team designed and implemented home-grown matching algorithms for the target tasks.

4.4 Deployment and Feedback Gathering

Deployment and feedback gathering involves two typical scenarios. One is that, as researchers, we have obtained some insightful information from the data and we would like to ask domain experts to review and verify. The other is that we ask domain experts to use the analytic tools that we have developed to obtain insights by themselves. Most of the times it is the second scenario that we want to enable. These scenarios in the phase of deployment and feedback gathering require that the deployed tools have good usability as well as great data presentation and interaction mechanisms that are powered by information visualization techniques.

As an integrated part of the deployed tools, the SA project team needs to design a mechanism along with its user interface that allows customers to integrate their domain knowledge into the tool, or customize the tool based on their specific needs. In addition, sometimes the SA project team may need to educate customers that doing so may be necessary to achieve satisfactory task results (otherwise, some customers may have high expectation to over-optimistically consider the use of the tools to be just one mouse click). For example, the StackMine user interface allows customers to specify filtering scopes for traces, frequent function-call sequences, or sequence clusters. The XIAO user interface allows customers to set different similarity-threshold values for matching depending on their target tasks: lower threshold values for security-defect detection to reduce the chance of missing important defects, and higher threshold values for larger code bases or less allocated inspection time to reduce the required clone-inspection effort.

The SA project team needs to design tool user interfaces to allow customers to *incrementally* integrate their knowledge to tools over time of their use of the tools. In other words, the more the customers use the tools, the "smarter" the tools become. For example, the StackMine user interface allows customers to turn "good" learned frequent function-call patterns into the knowledge base of bottleneck signatures so that new traces could be matched against these signatures. When the matching is successful, interactions with the analysis results on new traces could be avoided to save inspection cost. The XIAO user interface allows customers to collaboratively tag the analysis results (e.g., uninteresting, problematic inconsistencies, and refactoring opportunities) of a code base so that interactions with the analysis results for later versions of the code base could be reduced to save inspection cost.

4.5 Domain Knowledge and Expertise

Crosscutting the experiences gained from the four phases of developing a software analytic project, the most important one is that various domain knowledge and expertise are strongly needed in successfully developing a software analytic project for technology

transfer. Besides collaboration between researchers and customers, virtual SA project teams could be formed via open and extensive collaboration among researchers in domains such as machine learning, visualization, system, and software engineering.

Some major types of domain knowledge are listed below.

Specific application domain knowledge. This type of knowledge is typically specific to the software application under analysis: it is difficult for the SA project team to pre-hardcode such knowledge into the analytic tools. Therefore, the customers are the ones to acquire such knowledge and the SA project team needs to design tools to allow the customers to integrate such knowledge into tools at the tool usage time (see Section 4.4).

Common application domain knowledge. This type of knowledge is typically common across a family of software applications under analysis (e.g., applications using system APIs from kernel32.dll). Therefore, the SA project team can pre-hardcode such knowledge into the analytic tools. As discussed in Section 4.3, transferring such knowledge from the customers to the SA project team could be challenging.

Data domain knowledge. The SA project team needs to acquire such knowledge to develop analytic tools (e.g., in the phase of data preparation). For some tasks, the customers may also need to acquire such knowledge. For example, when using the StackMine tool, the customers need to inspect raw stack traces guided by StackMine.

Some major types of expertise are listed below.

Task expertise. The customers need to have expertise to carry out the target tasks, with the assistance of analytic tools developed by the SA project team. The SA project team would also need to work with the customers to learn the workflow of the customers in carrying out the target tasks in order to develop analytic tools that could effectively help the customers to perform the workflow or even improve the workflow.

Management expertise. The SA project team needs to have members with good management and communication skills to interact with the customers and manage the team. Successful technology transfer heavily relies on gaining sustained trust of the customers on the SA project team, soliciting requirements from the customers, and managing software analytic projects to fulfil the requirements.

Machine learning expertise. The SA project team needs to have expertise to develop machine learning algorithms and tools. The SA project team needs to have good understanding of existing machine learning algorithms and their implementations (not just in a black-box way).

Large-scale data processing/computing expertise. The SA project team needs to have expertise to design and implement scalable data processing tools and learning tools. Such expertise is coupled with system-building expertise.

Information visualization expertise. The SA project team needs to have expertise to design and implement good user interfaces and visualization for presenting analysis results and allowing the customers to manipulate the final analysis results as well as raw data or intermediate results produced by analytic tools (see Section 4.4).

5. CONCLUSION

Based on our experiences of successful technology transfer on software analytics at Microsoft Research Asia, in this position paper, we have advocated that when applying analytic technologies in practice of software analytics, one should (1) incorporate a broad spectrum of domain knowledge and expertise, e.g., management, machine learning, large-scale data processing and computing, and information visualization; and (2) investigate how practitioners take actions on the produced insightful and actionable information, and provide effective support for such information-based action taking.

Recently, Zeller et al. [8] discuss some pitfalls in conducting research on empirical software engineering. One of their suggestions as quoted below supports the second part of what we have advocated when developing a software analytic project: *"**Get real**. ... Far too frequently though... do we rely on data results alone and declare improvements on benchmarks as "successes". What is missing is grounding in practice: What do developers think about your result? Is it applicable in their context? How much would it help them in their daily work?"* [8] In this position paper, we have provided successful concrete examples on how to "get real" in the domain of software analytics.

Acknowledgment

We thank the members of the Software Analytics group at Microsoft Research Asia for contributing to the example projects described in this position paper and helping shape the position. We thank Ira Baxter, Judith Bishop, Prem Devanbu, Harald Gall, Mark Harman, David Notkin, Sriram Rajamani, Wolfram Schulte, David Weiss, and Minghui Zhou for their feedback on an earlier version of this paper.

6. REFERENCES

[1] R. P. Buse and T. Zimmermann. Analytics for software development. In *Proc. FSE/SDP Workshop on Future of Software Engineering Research (FoSER 2010)*, pages 77–80, 2010.

[2] Y. Dang, S. Ge, R. Huang, and D. Zhang. Code clone detection experience at Microsoft. In *Proc. 5th International Workshop on Software Clones (IWSC 2011)*, pages 63–64, 2011.

[3] Q. Fu, J.-G. Lou, Y. Wang, and J. Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *Proc. 9th IEEE International Conference on Data Mining (ICDM 2009)*, pages 149–158, 2009.

[4] K. Glerum, K. Kinshumann, S. Greenberg, G. Aul, V. Orgovan, G. Nichols, D. Grant, G. Loihle, and G. Hunt. Debugging in the (very) large: ten years of implementation and experience. In *Proc. 22nd ACM SIGOPS Symposium on Operating Systems Principles (SOSP 2009)*, pages 103–116, 2009.

[5] A. E. Hassan and T. Xie. Software intelligence: Future of mining software engineering data. In *Proc. FSE/SDP Workshop on Future of Software Engineering Research (FoSER 2010)*, pages 161–166, 2010.

[6] K. Hullett, N. Nagappan, E. Schuh, and J. Hopson. Data analytics for game development: NIER track. In *Proc. 33rd International Conference on Software Engineering (ICSE 2011)*, pages 940–943, 2011.

[7] J.-G. Lou, Q. Fu, S. Yang, J. Li, and B. Wu. Mining program workflow from interleaved traces. In *Proc. 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2010)*, pages 613–622, 2010.

[8] A. Zeller, T. Zimmermann, and C. Bird. Failure is a four-letter word - a parody in empirical research. In *Proc. 7th International Conference on Predictive Models in Software Engineering (PROMISE 2011)*, pages 5:1–5:7, 2011.