Decoding Technology Transfer through Experiences at Microsoft

Dongmei Zhang
Microsoft Research, Beijing, China
dongmeiz@microsoft.com

ABSTRACT

Technology transfer is an important form of collaboration between software engineering researchers and industrial practitioners. Despite benefits to both parties, it remains a huge challenge to carry out a technology transfer successfully. Based on the experiences at Microsoft, this talk discusses some key aspects in technology transfer, including technology readiness, partnership building, and one team as collaboration model.

KEYWORDS

Technology transfer, Software engineering

1 INTRODUCTION

Technology transfer is an important form of collaboration between software engineering researchers and industrial practitioners. For researchers, it is an opportunity to create practical impact using their research results, as well as get inspired on new research problems from real-world challenges. For practitioners, it is an opportunity to leverage advanced technologies to enhance products and improve productivity. While there have been successful examples of technology transfer on different topics in software engineering [1], [2], it still remains a huge challenge to carry out a technology transfer successfully. There has been discussion on the gaps between academic research and industrial practice, and lessons learned through technology transfer experiences [3], [5], [9].

In this talk, three key aspects of technology transfer are discussed, including *what* referring to the technology being transferred; *who* referring to both parties involved in the transfer, i.e. software engineering researcher, and industrial practitioner a.k.a. product team partner; and *how* referring to the technology transfer model. Specifically, using the examples in Section 2, challenges associated with each of the three aspects are addressed in detail - technology readiness, partnership building, and one-team model.

2 TECHNOLOGY TRANSFER EXPERIENCES AT MICROSOFT

Over the years, while conducting research on software analytics [10], the researchers in the Software Analytics Group at Microsoft Research collaborate extensively with product teams across the company. Through close partnership, the researchers have conducted a series of successful technology transfers, covering topics such as program analysis [11], OS performance, and reliability of distributed systems [7], etc. In addition, the researchers have extended technology transfer beyond the software engineering domain to other domains such as business intelligence [8]. The following are two examples of technology transfers.

Code clone detection has been studied by software engineering researchers for many years. Targeted at easy adoption in practice, a code clone detection technique named XIAO [4] was developed with high tunability, scalability, compatibility, and explorability. XIAO was transferred to different product groups, and its impact was highly recognized by the security group. XIAO helped security engineers effectively and efficiently locate potential security vulnerabilities caused by copy-and-paste code.

Debugging in the large is an important mechanism to improve software quality via leveraging the huge amount of telemetry data at the deployment sites such as performance counters, system logs, and stack traces. One challenge of debugging in the large is how to scale the analysis effort in order to fully utilize the power of data. StackMine [6] is a mining technique that helps developers find performance issues hidden in millions of stack traces. It was transferred to the Windows performance team and has been proven to effectively improve the productivity of performance analysts in terms of handling similar problems as well as finding new problems.

SER&IP'18, May 29, 2018, Gothenburg, Sweden
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5744-9/18/05.
https://doi.org/10.1145/3195546.3195553

REFERENCES

[1] Thomas Ball, Vladimir Levin, and Sriram K Rajamani. 2011. A decade of software model checking with SLAM. *Commun. ACM* 54, 7 (2011), 68–76.
[2] Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, and Dawson Engler. 2010. A few billion lines of code later: using static analysis to find bugs in the real world. *Commun. ACM* 53, 2 (2010), 66–75.
[3] Lionel Briand. 2012. *Embracing the Engineering Side of Software Engineering.* IEEE Computer Society Press. 96–96 pages.
[4] Yingnong Dang, Dongmei Zhang, Song Ge, Chengyun Chu, Yingjun Qiu, and Tao Xie. 2012. XIAO: Tuning code clones at hands of engineers in practice. In *Proceedings of the 28th Annual Computer Security Applications Conference.* ACM, 369–378.
[5] Carlo Ghezzi. 2009. Reflections on 40+ years of software engineering research and beyond: an insider‘s view. In *Keynote address in 31st International Conference on Software Engineering.*
[6] Shi Han, Yingnong Dang, Song Ge, and Dongmei Zhang. 2012. Performance debugging in the large via mining millions of stack traces. In *International Conference on Software Engineering.* 145–155.
[7] Qingwei Lin, Jian Guang Lou, Hongyu Zhang, and Dongmei Zhang. 2016. iDice: problem identification for emerging issues. In *International Conference on Software Engineering.* 214–224.
[8] Bo Tang, Shi Han, Lung Yiu Man, Rui Ding, and Dongmei Zhang. 2017. Extracting Top-K Insights from Multi-dimensional Data. In *ACM International Conference on Management of Data.* 1509–1524.
[9] Dongmei Zhang. 2012. Software analytics in practice: approaches and experiences. In *IEEE Working Conference on Mining Software Repositories.* 1–1.
[10] Dongmei Zhang, Yingnong Dang, Jian Guang Lou, Shi Han, Haidong Zhang, and Tao Xie. 2011. Software analytics as a learning case in practice:approaches and experiences. In *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering.* 55–58.
[11] Hongyu Zhang, Anuj Jain, Gaurav Khandelwal, Chandrashekhar Kaushik, Scott Ge, and Wenxiang Hu. 2016. Bing developer assistant: improving developer productivity by recommending sample code. In *ACM Sigsoft International Symposium on Foundations of Software Engineering.* 956–961.