Software Engineering in the Age of Data Privacy: What and How the Global IT Community Can Share and Learn

Mark Grechanik
The University of Illinois at Chicago
Chicago, Illinois, USA
drmark@uic.edu

Fayola Peters
Department of CSEE,
West Virginia University,
West Virginia, USA
fpeters@gmail.com

Denys Poshyvanyk
Dept. of Computer Science
College of William & Mary
Williamsburg, VA, USA
denys@cs.wm.edu

Tim Menzies
Department of CSEE,
West Virginia University,
West Virginia, USA
tim@menzies.us

***Abstract*— It is notoriously difficult to both protect the confidentiality of private data and make the data available for different software engineering and maintenance tasks. While there are often several options that achieve an equivalent level of data protection, each option may have a unique impact on software engineering tasks. The goal of this technical briefing is to provide state-of-the-art overview of the research agenda, solutions, and questions related to data privacy issues in software engineering. One example of such a fundamental question of software engineering is how can a data owner protect private information so that the data subjects (e.g., persons, equipment) cannot be re-identified while the data retains their efficacy for software engineering tasks?**

***Index Terms*—Privacy, obfuscation, software engineering.**

I. I. TUTORIAL STRUCTURE AND CONTENT

Target audience: This tutorial is for software developers and managers who wish to learn how to build effective data sharing mechanisms for their organizations.

Content: In the age of the Internet and Big Data, it is often assumed that the data we need to solve any problem is readily available. However, this is often not the case. While there is much that we can learn from each other, there is often little we dare to share. There are many reasons why data is in short supply including issues of confidentiality and a very real desire not to compromise an organization's competitive advantage. For these reasons, it is standard practice to privatize data prior to its release. Until very recently, the usual effect of such privatization was to damage the data in such a way as to make data mining more difficult. However, recent results show that it is possible to anonymize the data without damaging the usefulness of this data [2-4]. This tutorial will present solutions and algorithms to a number of software engineering problems. Also we will present examples of real-world applications of these techniques to tasks such as defect prediction, cross-company repository mining and learning, software testing and program comprehension.

Pre-requisites: Since this tutorial does not present complex mathematical details or data structures, it has no prerequisites and would be suitable for non-technical managers and developers alike.

Proposed time: 3.5 hours.

II. TUTORIAL DETAILS

Creating and maintaining software is beset by many challenges, which include protecting sensitive information. Not only do recent data protection laws and regulations around the world prohibit organizations from disclosing confidential data, but they also impose stiff consequences for these organizations should they accidentally release sensitive information in software artifacts. Also, extracting project data from organizations is often very difficult due to business sensitivity associated with the data. Because of this sensitivity, data owners who want to share limited amounts of useful data (say, to advance scientific research leading to improved software) need to do so without breaching any data privacy laws or company privacy policies. Unless we can address these concerns, continued progress in Big Data field will be stalled. For these reasons, many researchers question the practicality of sharing the data for different research purposes. In a personal communication, Barry Boehm reports he can release none of the data that his COCOMO team collected after 1981. Similarly, at a recent keynote address at ESEM'11, Elaine Weyuker doubted that she will ever be able to release the AT&T data she used to build defect predictors.

For this reason, many researchers explore methods to anonymize the data. Unfortunately, a repeated result is that the *more* we anonymize the data, the *more* useless it becomes for certain utilities of certain tasks, for example, *classification*. Grechanik et al. [2] and Brickell et al. [1] report that the application of standard privacy methods such as *k*-anonymity or *l*-diversity or *t*-closeness damages inference power and offers little overall improvements in privacy.

Two presenters of this tutorial (Grechanik and Peters) are the authors of the two known algorithms that support privacy while preserving our ability to mine that data for different software engineering tasks. This tutorial will present those algorithms, after a careful review of the privacy literature. That is, this tutorial will give attendees a summary of the state of the art as well as insight and implementation details needed to deploy those methods within their organization or apply them in their research.

In particular, the tutorial will cover the background material and start with the business case for sharing the data (e.g., cross-company results where we can learn effectively from other organizations). Then the tutorial will overview the theory of privacy: types of privacy (re-identification, sensitive attribute disclosure) and measures (entropy, increased privacy ratio, sensitive attribute disclosure). Then, we will cover well-known algorithms such as k-anonymity, t-closeness, feature and instance selection, classification boundaries, suppressing and swapping. Finally, we will present concrete applications of data privacy algorithms for specific software engineering tasks such as bug prediction, program comprehension, mining software repositories.

III. DETAILS FOR THE SAMPLE TASK - OBFUSCATING SENSITIVE INFORMATION IN SOFTWARE

In the past decade, there have been many publicized cases of leaked source code that contained sensitive information from well-known companies. Clearly, sensitive information should be redacted in source code and other software artifacts; however, doing this manually is difficult and time-consuming. More importantly, blindly removing sensitive information from software artifacts may severely reduce program comprehension, thereby thwarting different software maintenance and evolution tasks. *Finding a solution that balances the goals of privacy and program comprehension in the context of software maintenance tasks is one of the modern challenges of global software development theory and practice*.

Also, software engineers often do not even get access to confidential data because of internal security rules that are put in place by organizations that own these data and because of different laws that regulate data protection and privacy. This situation complicates basic software engineering tasks such as testing. To give access to required data, data owners typically use commercial tools to anonymize or “sanitize” the data. Unfortunately, none of the existing tools takes into account basic software engineering tasks such as testing, which leads to situations where the anonymized data is of little to no value for software engineers. Currently, software engineers operate with little or no meaningful data, and it is a great obstacle to creating good quality software. The goal of this tutorial is to provide state-of-the art overview of the research agenda and solutions to redact sensitive information while preserving program comprehension and testing coverage [2, 4] during vital software development and maintenance tasks.

Moreover, we will discuss our vision of the future development at the intersection of data privacy and different software engineering tasks, for example, performance management and secure code search engines that enable developers to share and reuse source code without exposing sensitive information. We will present multiple examples and show possible solutions for these problems.

IV. DETAILS FOR THE SAMPLE TASK - MINING SOFTWARE REPOSITORIES

Different software repositories mushroomed in the past decade, many of which are closed and owned by large organizations and companies. Stakeholders who are engaged in maintaining software benefit from mining software repositories for two main reasons. First, mining software applications allows stakeholders to decide what features (i.e., units of functionality) they should implement in their new applications that are similar to the applications from software repositories. Second, stakeholders can determine what problems or bugs are common to many applications, and in turn predict what problems or bugs other applications are likely to encounter. A common key assumption for all existing mining approaches is that the source code of open-source applications is always available. Unfortunately, it is often not true in case of commercial software development. Many organizations cannot release the source code of their applications for various legal and organizational reasons. Furthermore, consulting companies such as Accenture, IBM, and HP Global Services do not own the source code that they produce – their clients do. Unauthorized disclosure of source code is embarrassing, and the damage is multiplied when sensitive information is revealed in software artifacts. In the past decade, there have been many well-publicized cases of disclosing sensitive information in software artifacts from different well-known companies including Cisco, Facebook, and Microsoft, Symantec. Moreover, even within the same company or organization source code cannot be easily shared – it is reported that NASA contractors are reluctant to share data about mission critical errors, lest that data is used against them during the bidding process. Therefore, many external service providers use only aggregated software metrics in the approaches for mining software repositories. A fundamental problem of mining software repositories is how to release different software metrics that are obtained from the source code of applications to external service providers for different SE tasks (e.g., defect and resource prediction) without revealing sensitive information about software projects that these software metrics describe. This is a new problem, an instance of which for cross-company defect prediction was recently addressed by Peters and Menzies [3]. However, many other instances of this big and important problem have not been addressed or even precisely formulated.

V. BIOS OF THE PRESENTERS

Mark Grechanik is an Assistant Professor at the department of Computer Science of the University of Illinois at Chicago. He earned his Ph.D. in Computer Science from the department of Computer Sciences of the University of Texas at Austin. In parallel with his academic activities, Mark worked for over 20 years as a software consultant for startups and Fortune 500 companies. Mark's research produces new algorithms and techniques with which data owners can share sensitive data with external testing organizations while protecting data privacy and preserving testing efficacy. Some of his new approaches are described in his paper that won the Best Paper Award at the IEEE International Symposium on Software Reliability Engineering (ISSRE'10).

Tim Menzies is a Professor of AI and SE at the Lane Department of Computer Science and Electrical Engineering, West Virginia University. He is the author of over 200 refereed publications and one of the co-founders for the PROMISE repository for repeatable SE experiments. He serves on the steering committees of IEEE ASE and on the editorial boards of IEEE TSE, the Empirical Software Engineering journal, and the Automated Software Engineering Journal. He was PC co-chair for IEEE ASE 2012.

Denys Poshyvanyk is an Assistant Professor at the College of William and Mary in Virginia. He received his Ph.D. degree in Computer Science from Wayne State University in 2008. Since 2010, he has been serving on the steering committee of the International Conference on Program Comprehension (ICPC), chair elect in 2012. He serves as a program co-chair for the 18th and 19th International Working Conference on Reverse Engineering (WCRE 2011 and WCRE 2012). He also serves as a program co-chair for the 21st International Conference on Program Comprehension (ICPC 2013).

Fayola Peters is a Ph.D. candidate at the Lane Department of Computer Science and Electrical Engineering, West Virginia University. Along with Grechanik, she is the author of one of the two known algorithms (presented at ICSE'12 [3]) that can privatize algorithms while still preserving the data mining properties of that data.

References

[1] Brickell, J. and Shmatikov, V., "The cost of privacy: destruction of data-mining utility in anonymized data publishing", in Proceedings of 14th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'08), 2008, pp. 70-78.

[2] Grechanik, M., Csallner, C., Fu, C., and Xie, Q., "Is Data Privacy Always Good For Software Testing?" in Proceedings of 21st IEEE International Symposium on Software Reliability Engineering (ISSRE'10), San Jose, California, USA, November 1-4 2010, pp. 368-377.

[3] Peters, F. and Menzies, T., "Privacy and Utility for Defect Prediction: Experiments with MORPH", in Proceedings of 34th IEEE/ACM International Conference on Software Engineering (ICSE'12), Zurich, Switzerland, June 2-9 2012, pp. 189-199.

[4] Taneja, K., Grechanik, M., Ghani, R., and Xie, T., "Testing Software In Age Of Data Privacy: A Balancing Act", in *8th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'11)*. Szeged, Hungary, 2011, pp. 201-211.