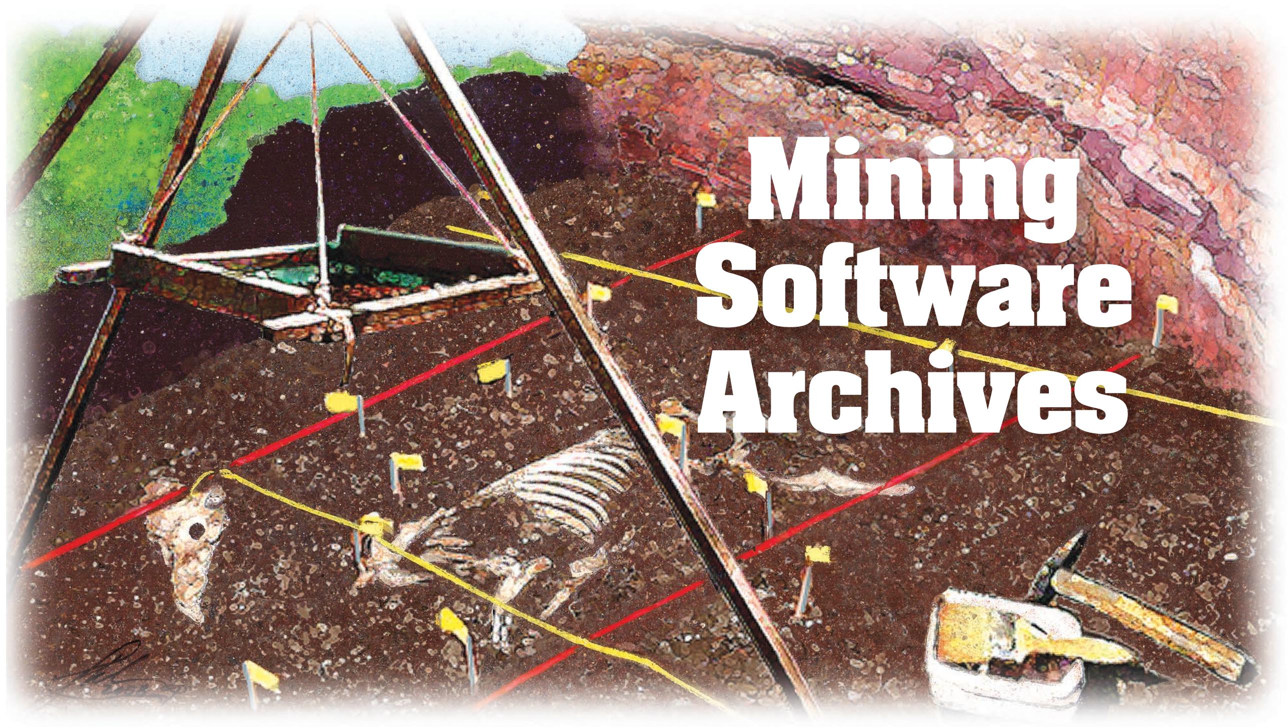
guest editors' introduction



Mining Software Archives

Nachiappan Nagappan and Thomas Zimmermann, *Microsoft Research*

Andreas Zeller, *Saarland University*

Corporations invest more than 300 billion US dollars annually in software production. Although new people are constantly entering the field, some of them aren't sufficiently trained and therefore aren't prepared to draw on the experience others have accumulated. This creates a situation in which every problem is perceived as new and unique, even though there's plenty of experience to learn from. Studying and collecting such experience is the goal of *empirical software engineering*, and its evidence finds its way into textbooks and magazines. Empirical studies tell us, for example, that the later a problem is discovered, the more effort it takes to fix it, and that 80 percent of the defects come from 20 percent of the code.

Such findings have long been common knowledge, but the consequences are very unspecific. How do we know where the most effort is spent? How do we know where the defects are? Which properties of the software or its development contribute to effort and quality? And, most important, how do we know whether some empirical or textbook finding applies to the project at hand?

To answer such questions, we need data—about the product, people, and process. However, collecting such data manually is expensive and can interfere with the development process and cost valuable developer time. If the data is collected from humans (for example, in surveys), there's a risk of bias, which we must estimate and deal with. Interpreting the data (again) requires considerable experience, time, and money.

A New Field

We have an alternative to manual collection, though. Modern programming environments and tools already collect data automatically. Configuration management tools (such as CVS) and bug-tracking systems (such as Bugzilla) are almost mandatory for systematic software development and are commonly integrated into modern programming environments, enabling automated, pervasive data collection. At the same time, modern program

0740-7459/09/$25.00 © 2009 IEEE

analysis techniques can derive more and more facts and abstractions from code, going much further than classical software metrics. All this allows for the exploration of far larger data bodies then ever before.

Such data isn't confined to industry alone. There are significant industrial projects (such as Mozilla, Apache, or the Eclipse project) that have gone open source, making plenty of industrial development data available for exploration and validation. If a technique is shown to be applicable to these projects, chances are that it will work in closed-source environments, too.

All this contributes to the rise of a new field, the *mining of software archives*, which is concerned with the automated extraction, collection, and abstraction of information from available software development data. In past years, mining software archives has become one of the fastest-rising areas in software development research. Its promise is not only to provide insights into actual development processes but also to provide tools and techniques that let anyone gather such insights with as little collection and modeling effort as possible.

The Special Issue

In this special issue, we're proud to present a selection of the exciting research that's going on in the field—a mix of contributions from industry and academia. In "Change Analysis with Evolizer and ChangeDistiller," Harald Gall, Beat Fluri, and Martin Pinzger describe a platform for mining software archives and how to answer essential questions about a project's evolution. "Mining Software History to Improve Software Maintenance Quality: A Case Study," by Alexander Tarvo, describes how to access the version history of Windows to predict the risk of changes. In "Analytics-Driven Dashboards Enable Leading Indicators for Requirements and Designs of Large-Scale Systems," Richard Selby shows how dashboards track and relate product and process metrics. The article "Mining Task-Based Social Networks to Explore Collaboration in Software Teams," by Timo Wolf, Adrian Schröter, Daniela Damian, Lucas D. Panjer, and Thanh H.D. Nguyen, shows how to mine social networks of developers, tracking patterns that are related to success or failure. In "Tracking Your Changes: a Language-Independent Approach," Gerardo Canfora, Luigi Cerulo, and Massimiliano Di Penta describe a tool that tracks the evolution of code fragments. They use their tool to answer common questions about code clones and vulnerabilities.

Finally, we've invited nine outstanding researchers in the field to share their thoughts on the future benefits of mining repositories—but also on possible pitfalls and limitations.

We hope these articles convey an idea about both the potential and the challenges of mining software archives. The sheer amount of data available, the diversity of sources, the semantic richness of both artifacts and natural language, and the overall goal of producing the most helpful insights will keep researchers busy for a long time.

Further Resources

The Working Conference on Mining Software Repositories (www.msrconf.org) is the main venue for researchers and practitioners to discuss ongoing research related to mining software archives. Each year, this conference hosts a mining challenge in which teams analyze a large open source project such as Mozilla, Eclipse, or Gnome (GNU Network Object Model Environment). The team with the best results wins.

The Bibliography on Mining Software Engineering Data (http://ase.csc.ncsu.edu/dmse) has numerous pointers to papers and other material, including a tutorial on mining software repositories.

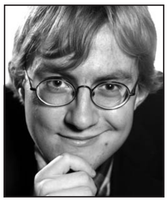
The IEEE Transactions on Software Engineering (www.computer.org/tse) published a special issue of seminal papers on mining software repositories in June 2005.

The PROMISE repository (http://promisedata.org) is a unique collection of free data sets related to defect prediction, effort estimation, and other software development activities.

About the Authors

Nachiappan Nagappan is a researcher at Microsoft Research, where he works in the Empirical Software Engineering and Measurement area. His research interests include software reliability, failure prediction, social network analysis, and empirical software engineering. Nagappan received his PhD in computer science from North Carolina State University. Contact him at nachin@microsoft.com.

Andreas Zeller is a computer science professor at Saarland University. He researches large programs and their history, and has developed methods to determine the causes of failures on open source programs as well as in industrial contexts, including at IBM, Microsoft, and SAP. His book *Why Programs Fail* received Software Development Magazine's productivity award in 2006. Zeller received his PhD in computer science from the Technical University of Braunschweig, Germany. Contact him at zeller@acm.org.



Thomas Zimmermann is a researcher at Microsoft Research. His work involves the evolution of large, complex software systems, conducting empirical studies and building tools that use data mining to support programmers. Zimmermann received his PhD in computer science from Saarland University. Contact him at tz@acm.org.