

Future of Mining Software Archives: A Roundtable

When mining software archives, we want to learn from the past to shape the future. But what does the research so far tell us about the future of the field itself? For this special issue, we invited and collected statements from nine research leaders in the field. These statements show opportunities for data collection and exploitation (Michael Godfrey, Ahmad Hassan, and James Herbsleb), enhancing programmer productivity (Gail Murphy and Martin Robillard), examining the role of social networking (Prem Devanbu), leveraging data for industry (Audris Mockus), and answering open research questions (Dewayne Perry). David Notkin, though, warns against too much enthusiasm: “Let us not mine for fool’s gold.” Enjoy!

—Nachiappan Nagappan, Andreas Zeller, and Thomas Zimmermann, Guest Editors

Answer Commonly Asked Project Questions

Michael W. Godfrey, *University of Waterloo*



Sensemaking is a term from cognitive psychology that concerns trying to understand a situation in the presence of ambiguous, contradictory, and complex data. The term is often used in the context of military operations, where three main interrelated activities occur: forming an awareness of the relevant key elements, forming an understanding of what it means to act on this information in a particular bounded context, and making (and later evaluating) appropriate decisions.

I believe the future of mining software repositories (MSR) lies in tying software development to the kind of sensemaking that managers and software developers perform daily, right now mostly on the basis of a “gut feeling.”

Progress will need to come on two fronts. First, through empirical study we need to better understand the kinds of questions that managers and developers commonly ask about their projects. Currently, it’s a challenge to answer even simple tracking questions, such as “Which projects are my team members working on?” and “How much activity occurred on the virtualization layer since last week?” In time, we hope to answer more complex questions about development, such as “Are we on schedule for next week’s release?” “How much did

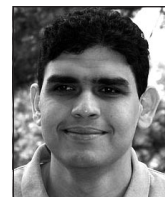
it cost to develop feature X?” and “What modules are proving unusually problematic?”

Second, we need to do some persuading. We must convince tool designers to insert hooks into their tools that can record useful information as painlessly as possible. We must also persuade developers to better instrument their own practices. To do so, we must be able to argue that the return on investment of their time and effort will be rewarded with improved knowledge about their system’s evolution. That’s the goal of today’s MSR research: to motivate the development of next-generation smarter tools.

Michael W. Godfrey is an associate professor in the University of Waterloo’s David R. Cheriton School of Computer Science. Contact him at migod@uwaterloo.ca.

Software Repositories: A Strategic Asset

Ahmed E. Hassan, *Queens University*



The MSR field will have a broad, powerful impact on practice. The possibilities are limitless thanks to MSR research innovations and the rich information in repositories. Repositories are readily available, so experimentation costs are negligible and the benefits can be empirically verified and quantified. Practitioners will recognize their repositories’ value as a strategic

asset. We must convince tool designers to insert hooks into their tools that can record useful information as painlessly as possible. We must also persuade developers to better instrument their own practices. To do so, we must be able to argue that the return on investment of their time and effort will be rewarded with improved knowledge about their system’s evolution. That’s the goal of today’s MSR research: to motivate the development of next-generation smarter tools.

asset instead of treating them as just static record-keeping stores. Moreover, practitioners will modify their processes to ease future mining efforts. More specifically, I expect advances in five areas.

First, modern development environments will enable the lightweight tagging and linking of data across repositories to ease future mining efforts. When committing a change, a developer will be able to link it in a structured way to email and other communications, bug reports, crashes, and effort information. This is an improvement on just typing a bug report ID or pasting an email conversation into it. The IBM Jazz development environment is a step in this direction.

Second, many tools for producing software repositories will provide APIs to ease future mining efforts. For example, configuration-management and bug-tracking infrastructures will provide APIs for querying and manipulating their repositories. Other development tools, such as build scripts, debuggers, and code editors, will start logging their operations in repositories so that MSR researchers can explore how developers are using them.

Third, repositories will become an important asset that organizations actively maintain and closely observe for quality throughout a project's lifetime. Practitioners will insist that when infrastructures such as configuration management systems (CMS) change, support for importing repositories is provided. Moreover, infrastructures will monitor the quality of data entered in their repositories. For example, they won't permit entering empty or meaningless CMS change messages.

Fourth, I see advanced CMS finally gaining traction in practice. For many years, we've had CMSs that could track changes at a structural level (for example, function changes), but no strong practical case existed for adopting these systems in practice. The demonstrated benefit of low-level tracking on MSR techniques will encourage practitioners to adopt such systems.

Finally, repositories will play a central role in understanding a software system's state. Instead of just measuring the complexity of a code base or its design, practitioners will closely examine a project's repositories. For example, when considering whether to invest in a company or to purchase a code base, organizations can closely

examine the project's repositories—similar to the way they examine the financial accounting books during such transactions.

Ahmed E. Hassan is an assistant professor with the School of Computing at Queen's University Canada. Contact him at ahmed@cs.queensu.ca.

Create Centralized Data Repositories

James Herbsleb,
Carnegie Mellon University



The last decade has witnessed an explosion of research that mines useful information and insights from the electronic traces of development activity. I

see this work evolving in at least three directions.

First, in addition to applying data-mining techniques to answer specific questions, I anticipate the creation of powerful exploratory visualization environments that will let nonstatisticians, such as developers and managers, understand the history, current state, and likely outcomes of large, complex projects.

Second, while the MSR community must now generally settle for analyzing data generated by software tools created for other purposes, future tool design should consider data generation as an explicit design criterion. We could create much richer data sets if our tools captured and created a centralized, accessible store of more detailed developer-activity traces. The traces could include every meaningful event, such as opening a file, displaying a specific document, and displaying one part of the code while editing another. These traces could, in turn, generate many kinds of event links, including temporal patterns (two pieces of code modified in close temporal proximity) or explicit references. Many new questions about the causes and consequences of development patterns and practices could be addressed with such data sets.

Finally, what we're learning about exploiting software repository data will likely apply to many other kinds of work. Currently, much work is done using individual applications on workstations and oc-

casional access to network resources such as the Web, shared drives, and document repositories. This tends to leave most work traces distributed across many machines and therefore inaccessible. However, we might be on the verge of a revolution in the movement toward delivering software as a service through a browser. Wherever this change in delivery mechanism happens, we'll have centralized data repositories as rich as we have for software, or perhaps richer. Instead of MSR, we'll have MXR, where X is most any work domain you care to study.

In my lab, we're actively pursuing all three of these directions.

James Herbsleb is a professor in Carnegie Mellon University's School of Computer Science and director of CMU's Software Industry Center. Contact him at jdh@cs.cmu.edu.

Embed Mining in Developer Tools

Gail C. Murphy,
University of British Columbia



Future software repositories will likely contain much broader information about a software development project. For instance, they might contain traces

of how a developer interacted with tools to navigate the software, chat sessions held about how part of the software works, or videos of requirements-gathering sessions.

Current MSR work has shown its ability to extract useful facts and trends from gathered information. In the coming years, I think we'll see these mining approaches and algorithms more embedded in tools that developers use as part of their regular work. To make this embedding a reality, we'll need to understand in which situations mined information can help a developer, how to deliver that information through innovative user-interface mechanisms, and how to engineer the mining computations to make the information accessible when developers need it in their work. The MSR future will provide fascinating work because it will bring together individuals with expertise from a wide range of areas, including algorithm development,

human-computer interfaces, data management, and, of course, software engineering.

Gail C. Murphy is a professor in the University of British Columbia's Department of Computer Science. She's also chief operating officer and cofounder of Tasktop Technologies. Contact her at murphy@cs.ubc.ca.

Help Developers Search for Information

Martin Robillard, *McGill University*



Software developers and other stakeholders spend a lot of time searching for information to solve problems, such as how to use a library efficiently. Often, the search process can be as useful as the results. That's because we build a better understanding of our task's context along the way: what distinguishes useful from useless information, how different pieces of information are linked, and so on.

If search processes can be useful in the future, we'd like to cost-effectively record and archive them in a repository. However, most search processes also involve a lot of wasted time: dead ends, irrelevant information, tangents, and so on. We have to find ways to avoid recording these.

I hope that future MSR research will increasingly focus on not only retrieving useful information and making accurate predictions but also inferring solutions and processes to complete complex recurring tasks. Software repositories give us hindsight—a tremendously powerful concept in knowledge creation. A person solving a problem today might not be able to tell whether something just discovered will be useful in the future. By mining repositories that archive users' past actions and past searches, we can better assess what should be documented, stored, and made available for retrieval.

In an ideal future, developers and other software development participants won't have to rediscover how to perform complex procedures because MSR-based techniques will have inferred the information from previous tasks and made it conspicuously available.

Martin Robillard is an assistant professor in McGill

University's School of Computer Science. Contact him at martin@cs.mcgill.ca.

Study the Social Side of Software Engineering

Prem Devanbu,
University of California, Davis



The bioinformatics field exploded 15 to 20 years ago when microarrays and gene-sequencing devices began producing torrents of data. Now, with the vast treasure trove of archived data from open source software (OSS), empirical software engineering is in a similar state of ferment. An exciting new topic is software engineering's human/social side. Unlike traditional projects, OSS projects make developer interactions—questions, answers, discussions, and arguments—visible in public mail archives.

Since the days of Fred Brooks (*The Mythical Man-Month*), we've known communication and coordination to be a key hurdle in large development teams. On the other hand, Eric Raymond ("The Cathedral and the Bazaar") and others have argued that OSS projects actually benefit from increased team size. Clearly, important questions remain about communication and coordination, and OSS provides a context for interesting and surprising answers to these questions.

In OSS, we can directly observe developer discussions on mailing lists. We can identify discussants and discussion content, such as patches and reviews. We can construct social networks, compare concomitant discussion and programming activity, measure the discussion-thread lengths and participant counts, and observe the entry of new participants and the exit of old ones. All this data can help answer various questions: Are OSS projects chaotic and bazaar-like, or do they have latent social structure? How do development and email activity relate (or not)? What types of discussions are the longest and most time consuming? How effective are OSS inspection discussions at finding defects? What kinds of defects are actually found?

Another critical phenomenon is the

quality of quality assurance information, such as bug fixes. Developers enter this data manually at their discretion, thus making it subject to error. Nevertheless, this data's quality determines how effective defect-prediction models and quality improvement programs (QIP) will be. What determines the quality of this data, and what are the consequences for prediction models and QIP efforts?

Prem Devanbu is on the Computer Science Department faculty at the University of California, Davis. He's also worked in industry for more than 20 years. Contact him at devanbu@cs.ucdavis.edu.

Deploy Mining to Industry

Audris Mockus, *Avaya Research*



Research based on data from project-support tools has improved our understanding of software development and enabled quantification of the effort, defects, and lead times associated with software technology. Data availability, uniformity over time, long histories, and detail for all parts of a system will continue to be important motivations for using such data. However, even though the use of this data in software research is routine, its use in industry remains elusive, except for the largest projects. To change that, I'd like to see more measurement support included with the underlying tools, such as Subversion and CVS (Concurrent Versions System). The most important improvement would be to make it easier and more natural for a developer using a version-control system to link each code submission with an associated problem report. Obviously, adding simple reporting capabilities, such as change trends and an expertise browser, would expose measurement's value to all users of version-control tools.

Clearly, version control, problem tracking, and forums aren't the only tools that can provide valuable software project information. As integrated-development-environment tools become more standard, we can capture more details of how the development proceeds, although privacy concerns and the complexity of underlying

data pose formidable challenges for anyone trying to use this information. Other tools, including test generation, test coverage, and build tools, present additional opportunities to understand key aspects of software development.

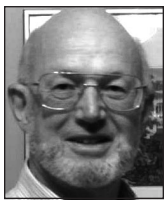
Simple models of effort, defects, and development lead time will become more sophisticated and will likely encompass important phenomena such as code reuse, offshoring, and distributed development. Better models of relationships among individuals involved in the multitude of software project tasks will likely improve our understanding of software development's social aspects.

Overall, the promise of research and tools that rely on data from software project support tools is tantalizing. For example, the ability to track the transfer of code ownership and to quantify how the product and organization coevolve will likely provide numerous lessons. It might significantly improve how software development organizations and products are created and structured.

Audris Mockus works in Avaya Labs' Software Technology Research Department. Contact him at audris@avaya.com.

Evaluate Analysis and Testing Approaches

Dewayne E. Perry,
University of Texas at Austin



Software repositories have provided a rich source for retrospective empirical studies on topics ranging from system faults to evolutionary phenomena. As

such, they provide extensive primary data about software systems and software development processes to be characterized, classified, and analyzed. It's frustrating that software repositories tend to be used mostly by researchers and seldom by developers or project managers, because they could provide significant support for continuous process and product improvement. For example, we could supply even richer data about faults when we deposit a version, and then use the richer data to detect fault-prone activities or fault-inducing types of changes.

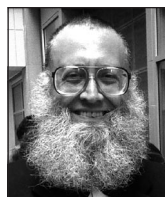
For both researchers and practitioners, many innovative uses of software repositories remain untapped. Although researchers have used them extensively for retrospective studies, there's been little use of them for prospective studies—that is, in experiments rather than just observational and relational studies. Change- and version-management repositories provide largely untapped resources for rigorously evaluating various software development techniques, methods, processes, and tools. For example, they provide rich data for evaluating analysis and testing techniques and tools. For each system component, we can build evolution and fault profiles, characterizing the kinds of changes as well as faults generated by different changes. Given this change and fault data, we can rigorously evaluate fault detection and prediction techniques and tools to determine the kinds of changes and faults for which they're the most effective. Given these rigorous evaluation approaches, we can then effectively and fruitfully compare different analysis and testing approaches, measuring how well they perform in the contexts most important for different project types.

This experimental approach sidesteps several validity problems of techniques currently used in such analysis and testing evaluations. For example, fault seeding is usually performed with little or no justification for the frequency and types of faults seeded. Repositories help us avoid such problems. We can then focus on the primary problem of building benchmark data sets covering a variety of domains as well as a variety of trade-offs in nonfunctional requirements.

Dewayne E. Perry is the Motorola Regents Chair in Software Engineering at the University of Texas at Austin. Contact him at perry@ece.utexas.edu.

Let Us Not Mine for Fool's Gold

David Notkin,
University of Washington



My interest in MSR dates back to about 1986, when Sharon Tuttle measured revision-control systems for her MS project to de-

termine their branching properties, longevity, the frequency of check-ins, and so on. Since then, the software industry boom, the Internet, the open source movement, and related phenomena have increased access to repositories. This access has provided and will continue to provide tremendous fodder for analyzing the evolution of authentic software artifacts.

MSR's potential is enormous. My chief concern is that we'll be distracted by the pleasure of and opportunities for finding relationships in and across these repositories that aren't causal. John Allen Paulos, in his lovely book *A Mathematician Reads the Newspaper*, describes how easy it is to create a conspiracy theory (such as those observed about Presidents Lincoln and Kennedy) for any two entities that offer a vast number of features to compare. Is a correlation we find during mining significant, or is it an artifact of something inconsequential?

Ultimately, we want to find causal relationships that we can use to define processes, tools, notations, educational approaches, and other means of increasing our ability to build better software systems. Mining repositories is, in itself, difficult; adding on an expectation such as this might be unreasonable. However, I believe it's fair to expect most work in this area to include some musings along the lines, "If we found extracted relationships to be causal, what might be some plausible avenues of exploiting that causality?" In other words, even if it's unreasonable to expect each result by itself to lead to improved approaches, MSR must have a trajectory leading in that direction. Otherwise, it will become less useful over time and will embody an opportunity cost in distracting software engineers and software engineering researchers from other fruitful paths. In other words, let us not mine for fool's gold. ☞

David Notkin is a professor and the Bradley Chair of Computer Science and Engineering at the University of Washington. Contact him at notkin@cs.washington.edu.

IEEE
Software
To subscribe, visit www.computer.org/software