Roundtable: What's Next in Software Analytics

Ahmed E. Hassan, Queen's University, Canada

Abram Hindle, University of Alberta

Per Runeson, Lund University

Martin Shepperd, Brunel University

Prem Devanbu, University of California, Davis

Sunghun Kim, Hong Kong University of Science and Technology

FOR THIS SPECIAL issue, we asked a panel of six established experts in software analytics to highlight what they thought were the most important, or overlooked, aspect of this field. They all pleaded for a much broader view of analytics than seen in current practice: software analytics should go beyond developers (Ahmed Hassan) and numbers (Per Runeson). Analytics should also prove its relevance to practitioners (Abram Hindle, Martin Shepperd). There are now opportunities for "natural" software analytics based on statistical natural language processing (Prem Devanbu). Lastly, software analytics needs information analysts and field agents like Chloe O'Brian and Jack Bauer in the TV show *24* (Sunghun Kim). Enjoy! —*Tim Menzies and Tom Zimmermann, guest editors*

Software Analytics: Going beyond Developers

Ahmed E. Hassan

Software analytics (SA) brings the notion of business intelligence to the software industry through fact-based decision support systems. Today, SA primarily focuses on helping individual developers with mundane coding and bug-fixing decisions by mining developer-oriented repositories such as version control systems and bug trackers. For instance, we can automatically determine the risk—the "bugginess"—of a code change by mining the actual risk caused by prior changes.[1]

Future SA research must look beyond these mundane tasks for SA to become a powerful, strategic, decision-making instrument. SA needs to service a project's various stakeholders—its marketing, sales, support, and legal teams—not just developers. It must go beyond mining developer-oriented repositories, to artifacts and knowledge across a project's various facets. Traditionally-mined code repositories should be mined and linked with other customer-facing repositories, such as operation logs, transcripts from customer support calls, blog posts, and video reviews.

Such a multifaceted view of SA would enable it to support more impactful business-level decisions instead of the usual, "Is this file buggy, and which developer can fix it?" Only then can we help developers and their managers reason more strategically about the importance of a piece of code and its impact on user satisfaction and revenue, assist support staff when answering customer calls through the linking of error logs to video tutorials, help marketers better target their ad campaigns based on field usage data, and guide sales staff in pricing features by understanding the inherent value that customers associate with each feature.

Reference

1. E. Shihab et al., "An Industrial Study on the Risk of Software Changes," *Proc. ACM SIGSOFT Symp. Foundations of Software Eng.* (FSE 12), ACM, 2012.

AHMED E. HASSAN is the NSERC BlackBerry Software Engineering Chair. NSERC is the Natural Sciences and Engineering Research Council of Canada, Canada's federal funding agency for

0740-7459/13/$31.00 © 2013 IEEE

university-based research. Hassan leads the Software Analysis and Intelligence Lab (SAIL) at the School of Computing, Queen's University, Canada, and has been a driving force behind the mining software repositories community and its associated conference (http://msrconf.org). Contact him at ahmed.hassan@gmail.com.

Proving Relevance to Practitioners

Abram Hindle

The low-hanging fruit of software analytics—measurement and reporting—is a resounding success. Modern software services such as GitHub, BitBucket, Ohlol, Jira, FogBugz, and the like employ wide use of visualization and even bug effort estimation. We can pat ourselves on the backs even if those developers never read a single one of our papers.

> The sweeter fruit of software analytics is spoiling: data-mining techniques are oblivious to the software domain.

Yet the sweeter fruit of software analytics is spoiling: data-mining techniques are oblivious to the software domain. I see a future in software analytics where layers of context are taken into consideration: the domain of software development (nonfunctional requirements, environments, tools, idioms, and so on), the domain of the software itself (databases, applications, and so on), and the context of the overall software project (requirements, glossary, architecture, community, and so on). For example, the *n*-gram models used by natural language processing ignore the source code's nested structure, but we need to incorporate this knowledge into our data-mining techniques. Thus, we need to stop blindly utilizing the latest data-mining tools and instead try to modify and configure them for software analytics.

I get a rush from an increase in precision or recall for bug duplication or commit classification, but what I don't get is confidence that this improvement is relevant to practitioners. Software analytics has to prove its relevance by showing its cost effectiveness versus the alternative, which is doing nothing. Doing nothing can be amazingly efficient. We need to evaluate these techniques with practitioners in mind. We don't need to evaluate if 10 percent more precision is meaningful for a bug triage or a manager—we need to evaluate if it's cost-effective.

The future of software analytics is in proving relevance to practitioners, proving the cost effectiveness of our techniques, and addressing the need for tools and techniques that leverage our knowledge of software engineering to provide more meaningful and less superficial software analytics.

ABRAM HINDLE is an assistant professor of computing science at the University of Alberta. He works on problems relating to mining software repositories, improving software engineering-oriented information retrieval with contextual information, and the impact of software maintenance on software power consumption. Contact him at abram.hindle@softwareprocess.es.

Mere Numbers Aren't Enough

Per Runeson

In recent years, software analytics has gained increasing interest from the software engineering community. This is a very good development toward a "systematic, disciplined, quantifiable approach to the development … of software," as stated in the ISO610.12 definition of software engineering. However, numbers aren't enough. The questions raised in software engineering are rarely sufficiently answered by a "4" or a function "$y = 3.1x + 2$". Numbers and equations are important to capture relations in the data, but for practical use, they must be accompanied with interpretation and visualization.

Interpretation of software analytics takes the findings of the number crunching into the real software engineering world of flesh and blood, organizations and company cultures, business and market. Mostly, it's a transfer from the quantitative domain to the qualitative domain. But let's say that software analytics find a negative correlation between project manager experience and project failure. Does that imply we should only have inexperienced managers? No, there's a third factor—project complexity. Experienced managers run more complex projects, which are more likely to fail! This interpretation process is exactly what's needed to make software analytics useful.

Visualization is another means of taking software analytics numbers into managers' domains. Although most software managers are technically and analytically skilled, they don't have time to dig into the details, so they need visualization approaches to fully grasp the findings. Graphs and charts generated by statistics and spreadsheet tools are a good start, but more research is needed on how to bring the message out of the software analytics to those who make decision based on them. The visualization is what will make the software analytics powerful. Basically, we should keep doing

research in software analytics, but let's not forget about the interpretation and the visualization.

PER RUNESON is a professor in software engineering at Lund University and leader of its Industrial Excellence Centre for Embedded Applications Software Engineering. Contact him at per.runeson@cs.lth.se.

Three Questions for Analytics

Martin Sheppard

The use of advanced machine learning and statistical methods to data mine various software engineering artifacts such as source code and change data has become a growth industry. Many interesting and useful models and discoveries are being made, but these kinds of approaches aren't without dangers. As researchers, we should ask ourselves the following three questions.

First, how much better is my model performing than a naive strategy, such as guessing or using the modal class? This might seem like an unnecessary question, but if analysis is restricted to outperforming another model or result, then there's a danger that we might end up with a model that's merely less bad than a dreadful model! If this seems far-fetched, I can vouch for it: Stephen MacDonell and I recently demonstrated that some previously published predictive models based on a combination of regression to the mean and case-based reasoning actually performed worse than simple guessing using permutation.[1] Naive benchmarks also have the advantage of being simple to use.

Second, how practically significant are the results? Typically, this is judged by using effect sizes.[2] But one of the dangers of focusing on null hypothesis significance testing and reporting p values, as is customary, is that when n is very large, which for data analytics is often the case, then even very small effects can become highly significant in the statistical sense (though not in the practical sense). Because our goal is to discover practically useful results, it's important not to become distracted by p values.

Third, how sensitive are the results to small changes in one or more of the inputs? We need to remind ourselves that we're dealing with noisy and uncertain data, so it's important to be aware of the impact on the model when, say, you have small measurement errors in one of the inputs. Sensitivity analysis is a means of systematically analyzing this kind of vulnerability.[3] It equips end users with the knowledge that a model critically depends on a particular input, so that they at least have the option of investing extra effort in ensuring its accuracy.

References

1. M. Shepperd and S. MacDonell, "Evaluating Prediction Systems in Software Project Estimation," *Information & Software Technology*, vol. 54, no. 8, 2012, pp. 820–827.
2. P. Ellis, *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*, Cambridge Univ. Press, 2010.
3. A.Saltelli, S. Tarantola, and F. Campolongo, "Sensitivity Analysis as an Ingredient of Modeling," *Statistical Science*, vol. 15, no. 4, 2000, pp. 377–395.

MARTIN SHEPPERD is professor of software technology at Brunel University, London, and associate editor of the journal *Empirical Software Engineering*. He is a Fellow of the British Computer Society. Contact him at martin.shepperd@brunel.ac.uk.

Toward "Natural" Software Analytics

Prem Devanbu

Analytics can be viewed as the discipline of designing and estimating statistical models of software phenomena, and then using those models to help programmers. From this elevated platform, we envision an entirely new and different frontier of research: using models from statistical natural language processing for a new kind of analytics.

> Everyday code is simple and highly predictable.

Natural languages such as English are complex, intricate, and expressive; any random Shakespearean sonnet will convince you of that. And yet, what most people write and say, most of the time, is highly repeatable and predictable. This mundanity of everyday language use, coupled with large online corpora, have made the statistical natural language processing (SNLP) revolution[1] possible, yielding incredible devices like Google Translate and Siri.

Over the past year or so, we've found that, surprisingly, code is no different.[2] While algorithms books bulge with amazing feats of programming prowess, most everyday code is simple and highly predictable. Indeed, we've been able to adapt standard n-gram models from statistical NLP to code, and train them on hundreds of millions of LOC. Using the entropy-based measures common in SNLP, we discovered that code is actually between 8 and 16 times more predictable than English.

We've already built two demonstration tools to exploit this fact. First, we showed that, even with a simple n-gram

model, we could significantly improve the performance of the powerful built-in Eclipse suggestion engine. Second, we found that statistical modeling can be used within the Dasher framework[3] to allow programmers with limited mobility to input code using just gestures.

But wait, there's more:

- It might be possible to implement code summarizations or retrieval (and even porting) as a *translation task*.
- It's likely that search-based software engineering methods can be improved by leveraging the highly skewed nature of programs' probability distributions (most possible programs, in fact, never occur in the wild).
- We also believe that with a large enough training corpora of annotated code, we can cast some static analysis problems as translation tasks.

There's a lot to do, and we urge you to join us!

References

1. K. Sparck Jones, "Natural Language Processing: A Historical Review," *Current Issues in Computational Linguistics: In Honour of Don Walker*, Kluwer, 1994.
2. A. Hindle et al., "On the Naturalness of Software," *Proc. Int'l Conf. Software Eng.* (ICSE), IEEE, 2012.
3. S.A. Wills and D.J.C. MacKay, "DASHER: An Efficient Writing System for Brain-Computer Interfaces?," *IEEE Trans. Neural Systems and Rehabilitation Eng.*, vol. 14, no. 2, 2006, pp. 244–246.

PREM DEVANBU is a professor of computer science at the University of California, Davis. He received a PhD from Rutgers. Contact him at devanbu@cs.ucdavis.edu or via www.cs.ucdavis.edu/~devanbu.

Wanted: Assistance from Information Analysts

Sunghum Kim

I'm a huge fan of spy thrillers such as *Mission: Impossible* and the TV series *24*. The field agents are amazing: they fearlessly stay focused on missions and make the right decisions at the right time. They're the heroes, but we shouldn't neglect the information analysts (such as Chloe on *24*) who usually sit behind a computer terminal at HQ. These people provide critical information, such as the backgrounds, strengths, and weaknesses of the people, places, and eventualities faced by the field agents. Without the information analysts, it's hard to imagine a successful mission. In fact, it's the information analysts who are the real heroes.

Unfortunately, when our software field agents (developers) fight against the bad guys, they don't have the assistance of information analysts available to them. Developers have to figure out all the necessary information about what and where and how to change the software by themselves. Consequently, it's not surprising to see many unsuccessful missions.

We need to provide the services of information analysts to developers and assist them in making the right decisions. Although excellent information analysts such as Chloe are in short supply, the good news is that software analytics can serve more or less the same function. Like human information analysts, software analytics can continually provide contextual information based on developers' current tasks. Decent information visualization and computer-human interaction technologies can help present this information efficiently.

SUNGHUM KIM is an assistant professor at the Hong Kong University of Science and Technology. His research area is software engineering, focusing on software evolution, software analytics, repository data mining, development social network mining, program analysis, and empirical studies. Contact him at hunkim@cse.ust.hk.