



What Happened to Software Metrics?

Jeffrey Voas and Rick Kuhn, NIST

A panel of seven experts discuss the past 40 years of software metrics, with a focus on evidence-based methods.

The software community of the 1980s was abuzz with seemingly endless approaches to producing higher-quality software. At the forefront was software metrics and its corresponding techniques, tools, and process-improvement schemes. Cyclo-matic complexity, Halstead metrics, source lines of code (SLOC), Fagan inspection, defect counting, number of defects prediction, reliability estimation and modeling, and other metric-oriented ideas were floated as solutions to the software quality “quagmire.”

To elucidate what went wrong—and right—in software metrics over the past 40 years and to explore future opportunities for new or hybrid metrics, we interviewed a panel of seven experts: Alain Abran (University of Quebec), Vic Basili (University of Maryland), Jim Bieman (Colorado State University), Ram Chillarege (Chillarege Inc.), Taghi Khoshgoftaar (Florida Atlantic University), Edward F. Miller (Software Research Inc.), and Adam Porter (Fraunhofer Institute). See the “Roundtable Panelists” sidebar for information about the panel. The panelists’ individual insights are presented below.

STATIC AND DYNAMIC SOFTWARE METRICS

Computer: If you could only recommend one static software metric

and one dynamic software metric, what would they be, and why?

ALAIN ABRAN: In most knowledge fields based on quantitative information, such as accounting, finance, engineering, and medicine, many quantitative ratios (or other formulae) are recommended for various contexts and purposes; nobody would expect a single measure or quantitative formula to be sufficient for analysis and decision-making. All of these industries have invested considerably in defining strict standards for basic measures and their various combinations, as well as in data collection and analysis to establish multidimensional industry benchmarks against which to compare.

The software industry, by contrast, has unrealistic expectations that poorly defined “metrics” can solve complex problems at almost zero cost. I recommend not one specific metric but a full set of measurement standards, as documented and recommended by the nonprofit International Software Benchmarking Standards Group (www.isbsg.org).

JIM BIEMAN: Any recommendation for a measure depends on context.

For static metrics, if you need to know how much software you have, a software-size metric is appropriate. If you seek information about design structure, you can measure code properties such as coupling, cohesion, complexity, and so on in numerous ways. If you want to know how testable your system is, you can statically measure how many specified “test requirements” it contains. For example, the number of statements or branches can indicate how difficult it will be to achieve a particular level of statement or branch coverage during testing.

Regarding dynamic metrics, runtime performance—such as time and space requirements—is clearly important for many applications. Another important and useful dynamic metric is the test coverage that’s achieved for specified test criteria. Finally, the most important dynamic measure is the number and frequency of defects discovered (or failures reported) in a system after release.

VIC BASILI: If you asked me to recommend one physics metric—for example, mass or energy?—I would immediately tell you that it’s a ridiculous question. You should select measures based on what you want to know and what you’re going to do with that information. The Goal-Question-Metric approach (proposed in 1984) identified relevant metrics by defining specific measurement goals. Defining goals involves specifying the object you’re measuring (for example, a product, process, or model), the focus of interest (for example, cost, defect removal, change, reliability, or user friendliness), the purpose (for example, to characterize, analyze, evaluate, or predict), the perspective of the person wanting the information (for example, the manager, developer,

or organization), and the context (for example, the organization’s characteristics and context variables). All of these help define what measures you need and how you’re going to interpret them.

RAM CHILLAREGE: It’s hard to find commercial software organizations with good metrics that are regularly measured and reviewed. The two most commonly recognized and understood metrics are SLOC and complexity. SLOC is better understood despite the high variance displayed among programming languages. Complexity is understood to a lesser degree. So, these are two that I would recommend.

TAGHI KHOSHGOFTAAR: Recommending one static or one dynamic software metric is akin to suggesting a one-size-fits-all solution, which is impossible in software engineering. Software-systems development and software-engineering measurements have evolved dramatically in the past two decades, emphasizing multifaceted focal points of critical importance. Instead of focusing on a single metric, we should explore intelligent data-wrangling and feature-engineering methods to best exploit the scores of auto- and expert-defined software metrics recorded by data-collection tools.

EDWARD MILLER: On the static side, the general understanding is that the more complex software is, the harder

it is to get right. So first off, I’d choose code size; using SLOC is probably the simplest. But simple as it is, its value is limited. I’ve seen very complex chunks of code that are solid and reliable. And I’ve seen collections of little components that you’d think would work out of the box but fail miserably when put through a test suite.

On the dynamic side, if you’re concerned about quality for the end user, then test-coverage metrics are the way to go. In the 1990s, we recommended branch coverage, but there were many fans of statement coverage, which was a lot easier to measure.

ADAM PORTER: It really depends on what you want to use the metrics for. If you asked construction workers to name their two most useful tools, they might consider tape measures and carpenter’s levels indispensable for some jobs but swear by plumb lines and speed squares for other jobs. The job defines which tools are right, not the other way around.

Similarly, organizations can create their own metrics. In many cases, that’s a better way to go, because they know best what they’re trying to understand and do with the metric. Collecting data just because it’s available doesn’t yield insights—you must have a goal in mind to improve or understand your software development in some way, and then define the data you want to collect based on that.

That said, I find that counting SLOC is a valuable, easy-to-compute

DISCLAIMER

Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

static metric for volume of work to be done. Similarly, I often look at line coverage percentage as a simple, easy-to-understand dynamic metric of testing effort.

SOURCE LINES OF CODE

Computer: There was once a common belief that all static code metrics essentially boiled down to SLOC. Was that true? If so, is it still true? If not, why?

ABRAN: Research findings from the late 1970s and early 1980s indeed pointed to the overall conclusion that the various static code metrics depended strongly on SLOC. Not much research has been conducted since to negate that conclusion. Personally, I'm not a big fan of SLOC-based metrics because they depend too much on technologies (such as programming languages, programming styles, and local coding standards) and their different implementations by tool vendors or researchers, thereby inhibiting the reproducibility and interpretation of values from analysis models across technologies, tools, and contexts.

BIEMAN: Many questions can only be answered if you know how much code is in a software system, subsystem, or version. The short answer is “yes”—the most useful static software metric is still the number of SLOC. SLOC's key advantage is that developers generally understand how it's measured, and it intuitively indicates how much source code a method, class, function, and so on contains. SLOC is regularly used as the denominator in derived measures such as defects discovered per KLOC. Of course, SLOC has many limitations. Different programming and layout styles, as well as different counting protocols (count comments, declarations, and so on) can affect SLOC.

BASILI: Certainly SLOC is a reasonable static metric if you want to know how big something is; but, of course, it depends on context and purpose.

Are you using [the metric] to characterize your products? If so, make sure the context is the same (same programming language, possibly the same application domain, and so forth). If the context is the same, are you using [the metric] to, for example, evaluate which process gives the smallest product, or to predict, say, the likely amount of resources needed to build the new product?

CHILLAREGE: For a long time, the function-point community maintained a steady following among practitioners. However, the function-point definition works best for classical business applications. Function points were captured manually and had limited applicability, so gradually faded away. In addition, the back-firing tables that convert function points to SLOC always made me wonder why using one would be much different from the other. So, SLOC, for all its perceived faults, remains the core size metric. Most current static code analyzers spit out the number. Thus, it's more visible in today's agile teams, although it might or might not be successfully used in projects.

KHOSHGOFTAAR: Studies have shown the defect-prediction capability of static code metrics, including SLOC. In software metrics' early days, limited availability of data and/or good data-collection tools influenced the general direction of research. To say that all static code metrics were being essentially equated to SLOC isn't true. However, one could argue that SLOC relates more to similarly simple metrics such as basic Halstead metrics; case studies have shown the different predictive powers of SLOC and complexity-based metrics, such as cyclomatic complexity. The answer lies in feature engineering with software metrics, as well as in the correlation among software metrics.

MILLER: SLOC highly correlates with every other metric, particularly

Halstead metrics. So if they're all correlated, why not use the simplest one? Source-code obfuscation creates many problems. For example, in JavaScript, so much is lost by removal of the context contained in the comments, and source expansion tricks didn't work well. Eventually, SLOC came to dominate people's thinking.

PORTER: In the late 1980s and early 1990s, much software-metrics research focused on defining metrics that assessed existing software's quality or predicted quantities such as expected development effort or the number of latent faults in a code base. Comparative studies of these metrics, however, generally failed to show that they were significantly and repeatedly better than using SLOC. However, metrics can be defined over many different software development artifacts, available at different times, and used for many different purposes. So saying all static metrics boil down to SLOC is too simplistic.

CAPABILITY MATURITY MODEL

Computer: In the 1980s and 1990s, many organizations were sold on the idea of process metrics such as the Capability Maturity Model (CMM). The US Department of Defense (DOD) invested heavily in that idea, and some have argued that this added significant financial burdens to military IT and software systems. Did it work? And where is CMM today?

ABRAN: Organizations without well-managed processes are unpredictable in terms of cost, duration, quality, functionality delivered, and so on. These uncertainties lead to poor quality, high costs when projects must be extensively reworked, and considerable waste when projects fail. Process-improvement models have been designed and adopted primarily to manage the risks and uncertainties associated with out-of-control

ROUNDTABLE PANELISTS



Alain Abran is a professor of software and IT engineering at the University of Quebec. His research interests include software productivity and estimation models, software engineering foundations, software quality, software

functional size measurement, software risk management, and software maintenance management. Abran received a PhD in electrical and computer engineering from École Polytechnique de Montréal. He was 2004 co-executive editor of the Guide to the Software Engineering Body of Knowledge and chairs the Common Software Measurement International Consortium. Contact him at alain.abran@etsmtl.ca.



Victor Basili is a professor emeritus of computer science at the University of Maryland. His research interests include empirical software engineering: evaluating and improving the software process and product. Basili received a PhD in computer science from the University of Texas, Austin.



James Bieman is the graduate director and a professor of Computer Science at Colorado State University. His research interests include software design quality evaluation and improvement.

Bieman received a PhD in computer science from the University of Louisiana at Lafayette. Contact him at bieman@colostate.edu.



Ram Chillarege is president of Chillarege Inc. His research focuses on semantics and quantitative methods to reason about software engineering processes, products, and people, and invented Orthogonal Defect Classification (ODC), which

increased the speed of root cause analysis by two orders of magnitude. He received a PhD in computer engineering from the University of Illinois, Urbana-Champaign. He received the IEEE Technical Achievement and Meritorious Service awards,

chairs the IEEE Software Reliability Engineering steering committee, and is an IEEE Fellow. Contact him at ram@chillarege.com.



Taghi Khoshgoftaar is the Motorola Endowed Chair Professor in the Department of Computer and Electrical Engineering and Computer Science at Florida Atlantic University. His research interests include data mining and machine

learning, big data analytics, software engineering, biomedical and health informatics, and computer security and intrusion detection systems. Khoshgoftaar received a PhD in statistics from Virginia Polytechnic and State University. Contact him at taghi@cse.fau.edu.



Edward F. Miller is founder and CEO of Software Research Inc., which develops automated testing tools such as TestWorks and eValid and allied technologies. He currently consults on automated web quality testing, cybersecurity, and intellectual property matters. Miller received a PhD in electrical

engineering from the University of Maryland. Contact him at edward.f.miller@gmail.com.



Adam Porter is executive director of the Fraunhofer USA Center for Experimental Software Engineering and a professor at the University of Maryland. His research focuses on tools and techniques for identifying and

eliminating bottlenecks in industrial software and systems development processes, experimental evaluation of fundamental software engineering hypotheses, and development of tools that demonstrably improve fundamental software and systems development processes. Porter received a PhD in computer science from the University of California, Irvine. Contact him at aporter@cs.umd.edu.

development processes. The organizations I've observed that have implemented these management concepts are successful and well managed, whether or not they adopted CMM.

BIEMAN: Many government agencies and companies require specified CMMI [CMM Integration] levels before

organizations can bid on a software development project. A CMMI evaluation makes the development process visible by measuring numerous process attributes. According to the CMMI Institute (cmminstitute.com), organizations in 98 countries use CMMI. Approximately 14,000 CMMI appraisals were conducted during the

past 10 years. Most (76 percent) of the appraised groups had fewer than 100 employees, and more than 70 percent of the appraised organizations use an agile development process. The number of appraisals has been increasing by nearly 20 percent per year, with the greatest increases in China, the US, India, and Mexico.

CHILLAREGE: Watts Humphrey [whose group at the Software Engineering Institute (SEI) developed CMM] explained to me that software-engineering metrics couldn't yet measure the quality of software acquired by the government. Thus, contractually there was no realistic way to enforce an acceptable criterion for software. Therefore, he strongly felt that

That process creates too many culture changes and can get quite expensive.

KHOSHGOFTAAR: SEI's original CMM and the CMM successors have been put to pasture, largely as a result of the lack of CMM's deep integration into the organizations' processes. DOD investment did lead large military defense contractors to accept CMM. However,

organization's goals is to define and implement process metrics that capture quality drivers specific to that organization.

CMM is based on well-studied notions of statistical process control and continuous process improvement. The general idea was that if you can measure a process, then you might be able to repeat it. If you can repeat it, then you might be able to improve it. If you can improve it, then you might be able to fine-tune it, and so on. In essence, first get consistency and control, then go for improvement.

The CMM framework assumed that to go through these steps, organizations needed certain capabilities, such as configuration management. I've seen companies vastly improve by working through this framework. However, a company with a given level of capabilities won't automatically produce a better product faster and at lower cost than companies with lower CMM levels. It really depends on what the company is doing with these capabilities and whether and how fast the underlying development requirements changed. Once specific CMM levels became prerequisites for getting DOD contracts, some companies were only interested in getting the credential and not in using their capabilities to improve. Additionally, the process itself became more heavyweight, hard to adjust, document-focused, and expensive, ultimately becoming less cost-effective for many practitioners.

You should select measures based on what you want to know and what you're going to do with that information.

the only way forward was to ensure that the suppliers' processes were acceptable, which would result in good software being delivered.

CMM is predicated on the premise that the process is far more measurable and controllable than the work product, namely, the software code. This set in motion the management of software for the next couple of decades. India in the late 1990s aspired to enter the software business, and CMM provided an excellent vehicle to systematically gain process skills and establish credibility in the market. Interest in the US was muted for various reasons.

Software, being an intellectual activity, defied many of the classical techniques of process control used in manufacturing. Although CMM's premise was mostly unproven, it gave management a clear framework to direct work and a ready assessment of achievement.

BASIL: The concept of capability maturity was based on [management theorist] Philip Crosby's original idea and was used to assess an organization's maturity. He never meant it as a prescription for building a mature organization but rather as a mechanism for finding the organization's weaknesses. The goal wasn't to keep adding processes until you get to level three and then start dropping or refining them.

in many cases, significant maturity success was achieved only after incorporating the CMMI approach. One could deduce that CMMI was a result of lessons learned from the stand-alone practice of CMM by organizations.

MILLER: Yes, but only indirectly. Many papers at QW/QWE [Quality Week/Quality Week Europe] pivoted off the basic CMM idea, bending it to fit the needs of "quality assurance and testing" organizations. The key notion was maturity of the internal process used and technical maturity of the team of programmers, developers, and testers involved.

With Harlan Mills' "chief programmer teams" [at IBM] in the early 1970s, everyone understood that democracy in programming work was no virtue. But not everyone could fill the shoes of a Mills-like "chief programmer." The compromise seems to have been to develop metrics for the entire team, which led to CMM and subsequent models. Why it worked is pretty simple: it forced people to think in process terms and pay attention to outcomes.

PORTER: CMM isn't a set of process metrics but rather a set of key process areas that, when implemented effectively, should help companies improve their software quality while controlling cost. One of a CMM

NEW STATIC METRICS

Computer: The software metrics of the early 1990s were mainly static; however, software's behavior is dynamic. Do we have newer static metrics that better reveal software behavior and semantics than only software syntax?

ABRAN: I'm puzzled by this view that coding is the only software development artifact to monitor and control. For instance, requirement quality and size are the foremost artifacts

underlying the whole development process; ambiguous and incomplete requirement specifications lead to major problems, including continuous reworking throughout all subsequent development stages, improper planning and monitoring, and, of course, incomplete or inaccurate definitions of testing artifacts. I haven't seen significant advances in SLOC-related metrics since the early 1990s; however, there have been significant advances in requirement specification and architectural measurement that can be extended throughout the full lifecycle to ensure traceability in later project phases and normalization of various technology-dependent ratios.

BIEMAN: I'm interested in the design structure at an intermediate abstraction level. For example, you can analyze a software design (and implementation) in terms of the existence and number of realizations of various design patterns and the connection between design-pattern realizations. Another measure that can be very useful in analyzing the testability of a system is to count the number of test requirements that test cases must cover to achieve particular criteria. We can also understand more about a design by categorizing and counting design-pattern realizations.

BASIL: There were lots of dynamic metrics in the 1990s, for example, reliability and performance. It's not clear whether a static metric can provide insight into the dynamic behavior of software unless you look at that metric's variation over time. Reliability and performance metrics are commonly used in many organizations; for example, look at Elaine Weyuker and Tom Ostrand's more recent work applying reliability models at AT&T.

KHOSHGOFTAAR: Software development is a complex process, with many variable attributes including development methodology and project objectives. Therefore, it's difficult

to determine a consistently good metric for predicting software behavior. Case studies have shown SLOC and other simple metrics are better defect predictors for some projects than for others. Likewise, newer metrics, both static and dynamic, show varying effectiveness at predicting software behavior for different projects. Software metrics' current focus has been on combining syntax descriptors and dynamic software attributes. In general, the novelty of newer static metrics will vary from expert to expert; however, the discussion should also include feature selection for optimal metric selection based on modeling goals.

MILLER: The software testing community has put a lot of research effort into extrapolating beyond the structural metrics, but I've not seen much that reveals anything particularly valuable for predicting trouble spots.

In a different arena, you'll find many patents and patent applications dealing with manipulation of a webpage DOM [Document Object Model] and extraction of user-oriented metrics from delivered webpages. This is very neat and sophisticated, even if the importance and application aren't fully clear. Some very big companies collect such things as "DOM settling time" from remote machines globally, despite the fact that it's not really a significant performance bottleneck for all but the hairiest webpages.

PORTER: One interesting trend in modern software development is model-driven software engineering. Models are increasingly being used, especially for embedded cyber-physical systems, to specify requirements, analyze prototype implementations, and even generate system code. Metrics defined on models rather than source code are currently being developed. These metrics have many desirable properties. For instance, they're defined at the requirement or behavioral level, which is often more understandable to the end customer

than source code or implementation-level metrics are.

STRUCTURAL METRICS

Computer: Structural metrics measuring dynamic behavior have been around for decades. The most commonly cited are statement coverage, branch coverage, and modified condition decision coverage [MCDC], plus a few module-level coverage metrics for object-oriented code. What percentage of developers in your industry or profession use one or more of these metrics? Are there other dynamic metrics being used?

BIEMAN: I don't have concrete, quantitative information concerning the use of coverage tools in industry. Anecdotal evidence from discussions with industry practitioners and the wide availability of coverage tools suggest that the coverage achieved during testing is commonly measured.

BASIL: Coverage metrics have been refined for newer development paradigms and languages such as object-oriented design. Their primary use has been to identify test quality. Of more importance is requirement coverage—ensuring that all requirements are appropriately covered and checking the requirements coverage vis-à-vis the various code-coverage metrics. Coverage metrics don't measure the dynamic behavior of the software product but the quality of the test suite. They're still commonly used to cover unit tests as well as system tests.

KHOSHGOFTAAR: A look at the PROMISE software project repository (openscience.us/repo) indicates the large extent to which organizations use structural code metrics to model software project behaviors. Researchers have used execution-based metrics such as computational time to model dynamic behavior. More recent studies have categorized dynamic software metrics such as cohesion-based dynamic metrics,

coupling-based dynamic metrics, and execution-traces-based metrics. Some predictive studies showed such metrics' superior power over traditional structural metrics.

MILLER: This brings to mind the software coverage metrics war from the 1970s through the early 1990s. Statement coverage was easy to measure but gave you a false sense of security. MCDC was harder to achieve and therefore got far less traction. Path or verification condition coverage was very hard to measure and got almost no traction.

The discussions were fascinating and building the measurement tools was exciting, but only a few developers really had the resources to use these tools in their intended way. Besides, when a budget crunch hit, coverage testing was one of the first steps to toss out. So, sad to say, overall usage of test-coverage metrics was probably less than 1 percent.

But it might be worth mentioning the modern practice of delivering a “new version” of a product to a subset of your user community and then waiting for the complaints—a kind of “involuntary crowd-testing.” It's sneaky but effective at inexpensively

isn't sufficient—you must also evaluate the test cases' quality and quantity. A single test case that executes all the lines of a system isn't useful.

SOFTWARE RELIABILITY MODELING

Computer: Software reliability modeling and theory have played a role in the past and continue to do so now. What percentage of developers in your industry or profession use reliability modeling? And is there one or two you recommend over others?

BIEMAN: I don't know that software reliability models are commonly used in most organizations.

BASIL: The most effective use of reliability measurement is when the system is operational, to predict how the system will perform in practice.

CHILLAREGE: “Software reliability” in the broadest possible definition would include terms and measures such as token, defect rates, backlog, closer time, customer satisfaction, first-time fix, re-create, criticality, pervasiveness, and trigger. Some of these are commonplace in industry but unheard of in academic articles. However, numerous

MILLER: John Musa's work [on software reliability engineering (SRE)] was seminal in this area, but there's no “wear out” phenomena to drive the model. That always struck me as a fundamental stumbling block. Without an underlying theory, statistical analysis is meaningful only for one methodology and one team. Change anything and the numbers could go anywhere. It wasn't something we put any stock in because we always fixed (or at least documented) every error as fast as we could. Zero outstanding critical errors was the continuous goal.

PORTER: In the 1980s and early 1990s, software reliability growth models were heavily investigated. More recently, there's been relatively little new research in the main academic software engineering conferences. However, as cluster and grid computing models became more popular in the late 1990s and early 2000s, practical measurements and applications of reliability (and availability) metrics were and continue to be used and improved. Descendants of these concepts are used in today's cloud computing infrastructures.

As far as recommending reliability models, you should fit the model to the data, not the other way around. Unfortunately, there's a lack of simple, out-of-the-box reliability modeling software packages for software developers to experiment with. Reliability modeling also generally requires that your testing environment accurately reflects your operational environment, which is difficult or impossible to do in many cases (think of cloud-based services or mobile computing).

There's a lack of simple, out-of-the-box reliability modeling software packages for software developers to experiment with.

ironing out goof-ups!

PORTER: While I don't have a well-validated percentage to report, the use of simple test-coverage metrics has increased substantially in recent years. One reason is that use of automated testing tools and environments has exploded in the last decade. It's increasingly easy to build and execute large test suites and capture test-coverage information as a nearly free by-product. However, coverage usually

academic articles discuss software reliability nuances that are foreign to even experienced software engineers in industry. This chasm has been bridged, just barely, in the past 20 years. Consequently, the industry hobbles along without leveraging a fairly large community of academic researchers. Despite this, software reliability metrics are probably the most widely used software engineering metrics—far more than the metrics that have to do with size, complexity, or productivity.

RELATIONSHIP BETWEEN METRICS AND TESTING

Computer: Software testing techniques and tools are often based on metrics. What do you see as the relationship today between metrics and testing?

ABRAN: Metrics per se are only inputs into quantitative models looking

for relationships across a number of variables. The challenge is that such relationships have been inadequately investigated to figure out which threshold values are meaningful in various contexts, including the very specific context of the software programs being tested.

Most of these code complexity and logic complexity metrics correspond to algorithms that capture only some of the targeted aspects—none of which directly represent what needs to be tested. By contrast, any functionality measured by a function-points method represents what functions must be tested under various sets of conditions; therefore, identifying these for measurement purposes can be reused directly as functional scenarios for testing purposes from both the developer and user perspectives and their quantitative information can be used for various analyses.

BIEMAN: A developer can use one of the readily available coverage tools to determine whether coverage goals are met. However, testers know that their goal isn't 100 percent coverage but rather to find 100 percent of the faults. Unfortunately, no tool can tell you that.

BASIL: I believe the most common metrics for testing are coverage metrics.

CHILLAREGE: Software testing could be a beneficiary of good metrics, especially given the numerous research ideas on methods to better test software. However, this is hardly the case in industry.

To put it in perspective, software testing continues to be one of the least advanced methods in the software development process. The product groups are most often better than their IT cousins. Most testing is manual. Test automation tends to be the high watermark for many organizations. Although automated testing's value is broadly recognized, its penetration in the practice is relatively low. It's also

the case that building a completely automated test environment is nontrivial. DevOps and agile methods have encouraged a focus on automation. The good news is that it's picked up in the past couple of years.

The software testing services that are sold are often time and materials contracts. And most testing vendors are reluctant to automate because they perceive it as a net loss of revenue. Leading-edge vendors take a longer term perspective and see automated testing as a win-win.

KHOSHGOFTAAR: Software testing techniques and tools aren't limited to guidance from different software metrics, including static code metrics and dynamic metrics. The software-testing phase often suffers due to compressed deployment time frames, prompting the output of metrics-based predictive models to guide software testing. However, much of testing is also guided by test cases' planning and code coverage. The project's criticality influences its software-testing emphasis. In today's agile-development environment, software development and testing are iterated in a compressed time frame. So, the emphasis on guidance by metrics on software testing and testing tools tends to increase.

MILLER: I've noticed some more modern metrics oriented to webpages. One is a "heat map" based on users' recorded GUI activity on a webpage front. At least a couple of vendors offer heat maps based on data consolidated across many users, sometimes without the users' permission. What's attractive is that you get a cleaner picture of what users think is important, so you know where to focus testing: right where users were really looking.

But thinking historically, I'm skeptical as to whether testing wasn't guided by any metric other than "what's important right now." Test teams, rightly enough, focused on the latest additions to an application but I

don't recall any teams that systematically measured and then tested in response to a metric.

PORTER: Popular testing metrics do a reasonable and generally cost-effective job of helping developers understand how thoroughly they're testing their software. Rather than viewing 100 percent coverage as an overriding goal, developers often use coverage information to point out where their test suites are inadequate. For this reason and because complex code-coverage metrics can be prohibitively expensive to collect (especially for very large systems), lighter-weight dynamic test-coverage metrics will be an interesting research topic in the near future.

PROCESS IMPROVEMENT

Computer: Process improvement suggested that a better process and better organization would produce better software. Did that ultimately occur, and can you suggest examples?

ABRAN: In organizations with sound, continuous process improvement, I've observed considerable improvement in the developer's credibility from all perspectives: quantity of functional requirements delivered (quantified objectively using ISO-recognized measurement methods), quality delivered, and predictability, as well as significantly fewer failed projects; that is, projects are abandoned in a timely manner where appropriate. I've also noted higher maturity levels (leading to a better understanding of process capability) and more realistic expectations (instead of inflated claims of delivery within an impossible schedule and unrealistic budgets).

BIEMAN: Paying careful attention to the development process and organization will lead to better software. Many, if not most, software development organizations are using some form of an agile process (for example, Scrum).

BASILI: The best example is NASA Software Engineering Laboratory work during the 1980s and 1990s, when we showed how various methods reduced costs and improved quality (as measured in resources expended and defects delivered). The improvement came from evolving the processes to meet the particular context based on measurement and feedback. More recently, look at the work of Lionel Briand and his Software Verification and Validation Laboratory (www.wfr.uni.lu/snt/research/software_verification_and_validation_lab).

CHILLAREGE: When process improvements are successfully implemented, the gains are phenomenal. But instances of continuous improvement are rare. In our work, we've seen improvements so explosive that the numbers are embarrassing. At IBM, a process improvement program based on orthogonal defect classification (ODC) yielded savings of over \$100 million. The same technology when applied at Nortel yielded similar results. In each instance, senior management understood the methods used and was the primary sponsor. The work was executed by a small technical team that had access and influence in the organization. In both instances, the work spanned between 1 and 3 years.

contract doesn't encourage process improvement. It places the responsibility on business and vendor management, which might be unable to find and leverage the necessary software engineering knowledge to successfully build process improvement into the contract.

KHOSHGOFTAAR: To a certain extent, yes, an improved focus on better processes and organization has resulted in less faulty software. The CMMI Institute and the SEI maintain reports of software development organizations that have measurably benefited from improving their development and organizational process. However, those examples typically come from high-assurance and/or mission-critical software projects, which have much to lose from poor software.

MILLER: I hate to be pessimistic, but I don't think the process-improvement movement made many inroads. Which is sad, because having a better process almost certainly improves the product's quality. But in the real world, programmers and developers chase bug reports more than anything systematic.

PORTER: As long as you don't read the word "better" to mean more detailed, formal, rigid, and so on, then I would

they became a truly excellent organization widely recognized for their innovative products.

But process improvement goes beyond CMMI-type approaches. Many organizations have adopted and institutionalized agile methods and now produce better, more cost-effective software than they did before the switch. Other organizations have invested strategically in building specialized domain knowledge within their development team and, again, now produce better software and are more effective in their specific customer markets.

COMMERCIAL OFF-THE-SHELF PRODUCTS

Computer: Once COTS products became the standard for software distribution, and source code was no longer available to customers, where did metrics fit in?

ABRAN: SLOC-based metrics are almost irrelevant in a COTS context. Industry hasn't used new software metrics, although they could have used function points to manage numerous COTS implementation and maintenance issues, including normalization of data collection to facilitate internal and external benchmarking for portfolio management and to objectively verify claimed productivity improvements with COTS.

BIEMAN: Customers can (and do) measure the size of COTS products by number of bytes of storage (both RAM and disk). Dynamic measurements can still be used.

BASILI: This changed the game for source-code metrics and forced whole new processes to be developed to take COTS into the equation. It's a good example of why metrics must be defined for the context.

CHILLAREGE: The COTS and metrics connection is at best remote. No engineering method that has matured

Testers know that their goal isn't 100% coverage but rather to find 100% of the faults. Unfortunately, not tool can tell you that.

When organizations attempt process improvements without the guidance of experienced people, they often fail due to poor implementation and lack of skill. What stands out after 20 years of implementing process improvements across the globe is how few organizations support and implement them successfully. Software outsourcing as a time and materials

say yes. Better organizations using better processes (as defined by them) will produce better software. For a concrete example, we worked for many years with a local company called Keymind. They decided to follow the CMMI approach, investing heavily in measuring their performance and improving their skills and tooling. Their investments ultimately paid off, and

can be allowed to deliver a service and claim it's not accountable on key parameters that affect society: reliability, injury, productivity, safety, and so forth. Yet the software industry has. Only today has the threat of software's security threats finally caught attention. The initial run-up on security was accepted after embarrassing disclosures by large firms. With its impact on politics, it's finally garnered more attention. Yet, the focus is mainly on protection and damage control and not on the fundamentals of the technology.

For years, the technical communities that wielded influence criticized disciplines such as metrics and reliability that focused on programming's behavioral aspects. Consequently, funding and generations of students and researchers were guided by priorities that ignored these very industry-relevant areas. Today, those disciplines that wielded influence have been commoditized, and we have a dearth of technical effort in needed software engineering areas. I founded and headed the Center for Software Engineering at IBM Research. My prediction then was that software engineering would regret the disposition held by the software technical community. IBM wisely let my opinion be heard, albeit without further investment or action. Twenty years later, we're witnessing the consequences that we, the collective technical society, chose.

KHOSHGOFTAAR: Because source code isn't available for COTS software, evaluation metrics tend to fall under categories of cost, return on investment, reliability, availability, and general black-box testing. The development organization might rely on quality-certifying entities that independently test COTS products and maintain data on product quality and related features for acquisition teams to review.

PORTER: Metrics aren't restricted to source code. Organizations can and do define metrics for non-source

code development artifacts, including requirements, user-visible display screens, and system resource files. The example of COTS is good. In work we're doing for a large government organization, we've been using metrics to understand how much COTS

customization this organization will need to perform.

We looked at many measures and realized that COTS development processes and activities, not just development of the glue-ware and integration of COTS components, must be considered; for example, what it takes to learn the capabilities of COTS products, configure COTS components to satisfy requirements, resolve issues with interfacing development teams, and enhance individual COTS products. Each activity requires significant effort, can cause great difficulty for a project, and isn't usually fully planned and allocated the effort needed to develop a project.

FUTURE DIRECTIONS

Computer: Are current metrics cost-effective? What aspects of software development aren't being adequately addressed by metrics today, but could be? What are some fruitful areas for metrics research?

ABRAN: The key issue with software metrics isn't cost, but whether they support decision-making. Organizations and industries must invest in analysis models relevant to their contexts and collect historical data to bootstrap their own models and threshold values for decision-making. Metrics tool vendors' professional practices must also improve. At present, whatever metrics they propose lack traceability to well-documented benchmarks or international standards.

Much research on software metrics is wrongheaded. Researchers too often collect metrics solely because they're easily automated. Then, using whatever open source data they can acquire, without verifying its quality, they figure out which ones might lead

When process improvements are successfully implemented, the gains are phenomenal.

to more accurate outcomes for whatever purpose. This isn't a sound research methodology.

BIEMAN: Most of the metrics used are relatively cheap to apply. However, metrics misuse can add costs by misdirecting developers. I'd like to see more research in two areas: evaluation of the measurable benefits and costs of applying common design advice, and process advice in terms of time to market, delivered faults, and maintainability; and use of Bayesian networks to build causal models for decision-making under the inherent uncertainty involved in software development.

CHILLAREGE: I invented ODC more than 25 years ago. ODC extracts the semantics contained in defects into four principal groups and, within each, bins them into independent categories. This multidimensional categorical data behaves like eigenvalues in the software development process space, thus creating a new measurement system. A dozen different process measurements and evaluations can be performed with ODC data. It's changed how root-cause analysis is performed, reducing effort by two orders of magnitude.

KHOSHGOFTAAR: Current software metrics are generally cost-effective. However, their extent of usage and role are dictated by project and organizational goals. An area that could use further insight via measurements is

FURTHER READING

Below are the collective group's recommendations for those interested in learning more about software metrics fundamentals.


- » A. Abran, *Software Metrics and Software Metrology*, John Wiley & Sons, 2010.
- » T. Ball et al., "If Your Version Control System Could Talk ...," *Proc. Workshop Modeling and Empirical Studies of Software Engineering (ICSE 97)*, 1997.
- » V. Basili et al., "SEL Software Process-Improvement Program," *IEEE Software*, vol. 12, no. 6, 1995. pp. 83–87.
- » N. Fenton and J. Bieman, *Software Metrics: A Rigorous and Practical Approach*, CRC Press, 2014.
- » N. Fenton and M. Neil, *Risk Assessment and Decision Analysis with Bayesian Networks*, CRC Press, 2012.
- » R.B. Grady and D.L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, 1987.
- » M.H. Halstead, *Elements of Software Science*, Elsevier Science, 1977.
- » H. Hecht, *A Survey of Software Tools Usage*, tech. report NBS 500-82, NIST, 1 Nov. 1981; www.nist.gov/publications/final-report-survey-software-tools-usage.
- » D.W. Hubbard, *How to Measure Anything: Finding the Value of Intangibles in Business*, John Wiley & Sons, 2010.
- » W.S. Humphrey, *Introduction to the Personal Software Process*, Addison-Wesley, 1996.
- » Int'l Software Benchmarking Standards Group, www.isbsg.org.
- » *ISO/IEC 15939:2007: Systems and Software Engineering—Measurement Process*, Int'l Org. Standardization, 2007; www.iso.org/standard/44344.html.
- » J.D. Musa, *Software Reliability Engineering*, Osborne/McGraw-Hill, 1988.
- » R. van Solingen and E. Berghout, *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development*, McGraw Hill, 1999.

the human impact of software quality. This is generally measured via defects and process metrics. An interdisciplinary focus between software engineering and psychology might yield insights. Another area is the influence of big data analysis on existing software development practices.

PORTER: Cost-effectiveness is context dependent. In addition, metrics will be more cost-effective if the definition of the metrics and the process to gather these metrics are carefully

designed so as to be cost-effective. Many companies create ad hoc measurement plans and don't take advantage of data automatically captured by development or test tools, nor do they plan for automated preprocessing or compilation of related data for easier analyses.

As for fruitful research areas, there's room for defining and validating metrics over development models. Model-driven development approaches are increasingly finding their way into standard practice.

We thank our panelists for sharing their expertise and for their candor. To learn more about the fundamentals of software metrics, see the "Further Reading" sidebar for recommended resources. So what do you think: are software metrics still relevant? What's the best way to incorporate metrics into modern software development and delivery? 

JEFFREY VOAS is a computer scientist at NIST. His research interests include the Internet of Things and fundamental computer science shortcomings. Voas received a PhD in computer science from the College of William and Mary. He is a contributing editor for *Computer's* Cybertrust column and a Fellow of IEEE and the American Association for the Advancement of Science. Contact him at j.voas@ieee.org.

RICK KUHN is a computer scientist at NIST. His research interests include combinatorial methods in software testing and access control models. Kuhn received an MS in computer science from the University of Maryland, College Park. He is a Senior Member of IEEE. Contact him at kuhn@nist.gov.

myCS Read your subscriptions through the myCS publications portal at <http://mycs.computer.org>