

The Infamous Ratio Measure

Hakan Erdogmus

Software productivity and quality are commonly expressed as derived metrics—in particular, the ratio of two base measures. For productivity, which I addressed in my previous column (“Measurement Acquiescence,” March/April 2008), the base measures are output (or a proxy thereof) and effort. For quality, common base measures are defect count and software size. The March/April 2007 *IEEE Software* article “Misleading Metrics and Unsound Analyses” (by Barbara Kitchenham, David Ross Jeffery, and Colin Connaughton) drew attention to the pitfalls of using ratios; however, ratios and other derived metrics are inevitable in software measurement. An appreciation of what they’re good for, and not good for, can help to avoid these



pitfalls. I focus on ratios here.

Why we need ratios

We need ratios for whatever limited portability we can achieve in software measurement.

Ratios provide the relativity required for comparison, something we can’t do with only absolute measures. They increase portability by hiding scale. Ironically, the loss of scale information also counteracts portability in various decision-making situations in which economies and diseconomies of scale play an important role. But without abstracting away from scale at least temporarily, it’s difficult to talk about how projects, artifacts, and teams

behave over time, both relative to each other and in response to changes in context.

Economic concepts such as productivity are inherently defined as ratios. Dependencies among base variables should not modify such definitions. For example, the effort prediction model COCOMO (Constructive Cost Model) suggests a polynomial relationship between effort and software size, resulting in diseconomies of scale. However, would this dependency change the COCOMO definition of software productivity from something other than the simple ratio of size to effort? The information, according to COCOMO, that larger projects tend to be less productive is external to this ratio metric.

The tendency of projects to move at different rates as they progress is observable if we track their absolute output over a moving time window of fixed duration. However, if the observations’ durations vary significantly, we must resort to using ratios by normalizing absolute measurements with respect to individual observations’ scopes. Similarly, if resources vary over time, meaningful comparison is possible if the measurements are adjusted for differences in scale. Such comparisons need not, and often should not, be for the express purpose of deciding which projects, artifacts, or teams are “better” over all performers. They may simply reveal notable, and justified, trends in project and product outcomes.

Interpreting simple ratios

A ratio metric’s proper interpretation begins with acknowledging the loss of information originally captured in the base measures. Consider a sprinter’s and a marathoner’s performance over several races

Mission Statement:

To be the best source of reliable, useful, peer-reviewed information for leading software practitioners—the developers and managers who want to keep up with rapid technology change.

of various distances. The marathoner's average speed, a ratio metric, would be lower than the sprinter's. This information isn't enough to decide which runner is a more accomplished athlete nor estimate who's likely to cover an arbitrarily specified distance in the shortest possible time. However, it lets us set behavioral expectations for specific decisions. If we're considering hiring one of the runners as a messenger for short-distance deliveries subject to strict deadlines, knowing that the sprinter is faster (based on average speed) over repeated dashes (based on the context in which that level of performance is valid), we might choose the sprinter.

Think of a ratio metric as representing an instantaneous quantity, a rate of change, or an average as opposed to an invariant equation relating the metric's value to its components. Remember that the metric's value is not independently and directly observable: it can be computed only from direct observations of the base measures. We must therefore avoid assuming a linear model governing the relationship between the metric and the base measures. A derived metric's value usually cannot be controlled by arbitrarily manipulating the base measures.

A case in point is defect density, a familiar ratio metric used to gauge software quality. We can't achieve a smaller defect density simply by increasing a system's size. Suppose an ongoing project has a defect density of 1.5 defects per KLOC at its current size of 10 Kbytes. On the basis of this measurement, should we expect about 15 defects when the software ultimately grows to 100 KLOC? A "yes" answer would mistakenly imply a linear predictive model. Here defect density represents a short-term measurement without long-term predictive power.

Spurious correlations

When I started looking more closely at the implications of using ratios, a colleague referred me to the book *Ratio Correlation: A Manual for Students of Petrology and Geochemistry* (University of Chicago Press, 1971). In this book, Felix Chayes presents statistical techniques for detecting relationships between molecules found in earth samples. In petrology and geochemistry, the occurrence frequency of molecules of interest is expressed as proportions because absolute counts aren't directly comparable across

samples. Correlations between such proportions are the main focus of Chayes' book.

A proportion is the ratio of two quantities where each quantity is the sum of a set of base variables. When two proportions (say $X/(X + Y)$ and $Y/(Y + Z)$) share base variables (in this case, Y), they exhibit a spurious correlation, called a null correlation, due to the shared component. Therefore, any statistical test designed to reveal a true relationship between two proportions must take into account this null correlation, for otherwise the test might indicate a false association. This insight also applies to a software metric expressed as a ratio.

Predictive models

Suppose a large legacy system's modules are randomly sampled for testing and the sampled modules' defect counts are tracked to optimally allocate testing resources. Based on multiple observations, we wish to develop a predictive model to estimate the defect density of an arbitrary module from the module's size. Should we regress module size against defect density? Or is it better to regress module size directly against the number of defects found and then estimate defect density from the predicted defect count?

Figure 1 shows artificially generated scatter plots of defect density versus module size for uncorrelated and correlated base measure samples. Distinguishing between the two plots is difficult because of the negative null correlation's domination. Hence, any predictive model derived from these measurements must take care to eliminate the spurious effect. Figure 2, on the other hand, shows plots of the absolute number of defects found against module size for the same data, where the relationship between size and quality is explicit. A predictive model is thus better off using the base variables directly. In this case, the use of the ratio metric, defect density, as a dependent variable would be unwise because it artificially increases the strength of the underlying relationship.

Functional forms

The scatter plots in figure 1 don't give us much information about the functional relationship between size and quality. It's apparent that size and quality are inversely proportional; however, this insight isn't particularly surprising or useful. But the

Hakan Erdogmus

hakan.erdogmus@computer.org

EDITOR IN CHIEF EMERITUS:
Warren Harrison, Portland State University

ASSOCIATE EDITORS IN CHIEF

Design/Architecture: Uwe Zdun, Vienna Univ. of Technology; zdun@infosys.tuwien.ac.at

Development Infrastructures: Martin Robillard, McGill University; martin@cs.mcgill.ca

Distributed and Enterprise Software: John Grundy, University of Auckland; john-g@cs.auckland.ac.nz

Empirical Results: Forrest Shull, Fraunhofer Center for Experimental Software Engineering, Maryland; fshull@fc-md.umd.edu

Human and Social Aspects: Helen Sharp, The Open University, London; h.c.sharp@open.ac.uk

Management: John Favaro, Consulenza Informatica; john@favaro.net

Processes and Practices: Frank Maurer, University of Calgary; maurer@cpsc.ucalgary.ca

Programming Languages and Paradigms: Sophia Drossopoulou, Imperial College London; s.drossopoulou@imperial.ac.uk

Quality: Annie Combelles, DNV/Q-Labs; annie.combelles@dnv.com

Requirements: Ann Hickey, University of Colorado at Colorado Springs; ahickey@uccs.edu

DEPARTMENT EDITORS

On Architecture: Grady Booch, IBM

Bookshelf: Art Sedighi, SoftModule

Design: Rebecca J. Wirfs-Brock, Wirfs-Brock Associates

Loyal Opposition: Robert Glass, Computing Trends

Not Just Coding: J.B. Rainsberger, Diaspar Software Services

Requirements: Neil Maiden, City University, London

Software Technology: Christof Ebert, Vector

Tools of the Trade: Diomidis Spinellis, Athens Univ. of Economics and Business

User Centric: Jeff Patton, consultant

Voice of Evidence: Forrest Shull, Fraunhofer Center for Experimental Software Engineering

ADVISORY BOARD

Stephen Mellor, consultant (chair)
Jennitta Andrea, ClearStream Consulting
Elisa Baniassad, Chinese University of Hong Kong
J. David Blaine, consultant
David Dorenbos, consultant
Kaoru Hayashi, SRA
Simon Helsen, SAP
Juliana Herbert, Herbert Consulting
Gargi Keeni, Tata Consultancy Services
Karen Mackey, Cisco Systems
Steve McConnell, Construx Software
Erik Meijer, Microsoft
Grigori Melnik, Microsoft
Bret Michael, Naval Postgraduate School
Ann Miller, University of Missouri, Rolla
Dependra Moitra, Infosys Technologies, India
Frances Paulisch, Siemens
Linda Rising, consultant
Wolfgang Strigel, consultant
Dave Thomas, Bedarra Research Labs
Laurence Tratt, Bournemouth University
Jeffrey Voas, SAIC
Markus Völter, consultant

STAFF

Senior Lead Editor
Dale C. Strok
 dstrok@computer.org

Senior Editorial Services Manager
Crystal Shif

Magazine Editorial Manager
Steve Woods

Staff Editors
**Rebecca Deuel, Shani Murray,
 Dennis Taylor, Linda World**

Assoc. Peer Review Manager
Hilda Carman

Publications Coordinator
Kathleen Henry
 software@computer.org

Production Editor
Jennie Zhu

Technical Illustrators
Ann Monn, Alex Torres

Sales and Marketing Director
Dick Price

Digital Library Marketing Manager
Georgann Carter

Senior Business Development Manager
Sandra Brown

Senior Advertising Coordinator
Marian Anderson

CONTRIBUTING EDITORS

**Thomas Centrella, Molly Gamborg, Robert Glass,
 Keri Schreiner, Joan Taylor**

CS PUBLICATIONS BOARD

Sorel Reisman (chair), Angela Burgess,
 Chita R. Das, Van Eden, Frank E. Ferrante,
 David A. Grier, Pamela Jones, Phillip A. Laplante,
 Simon Liu, Paolo Montuschi, Jon Rokne,
 Linda I. Shafer, Steven L. Tanimoto

MAGAZINE OPERATIONS COMMITTEE

David A. Grier (chair), David H. Albonesi,
 Arnold (Jay) Bragg, Carl K. Chang,
 Kwang-Ting (Tim) Cheng, Norman Chonacky,
 Fred Douglass, Hakan Erdogmus, James A. Hendler,
 Carl E. Landwehr, Dejan Milojicic,
 Sethuraman (Panch) Panchanathan,
 Crystal R. Shif, Maureen Stone,
 Roy Want, Jeffrey Yost

Editorial: All submissions are subject to editing for clarity, style, and space. Unless otherwise stated, bylined articles and departments, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in IEEE Software does not necessarily constitute endorsement by the IEEE or the IEEE Computer Society.

To Submit: Access the IEEE Computer Society's Web-based system, Manuscript Central, at <http://cs-ieee.manuscriptcentral.com/index.html>. Be sure to select the right manuscript type when submitting. Articles must be original and not exceed 5,400 words including figures and tables, which count for 200 words each.

FROM THE EDITOR

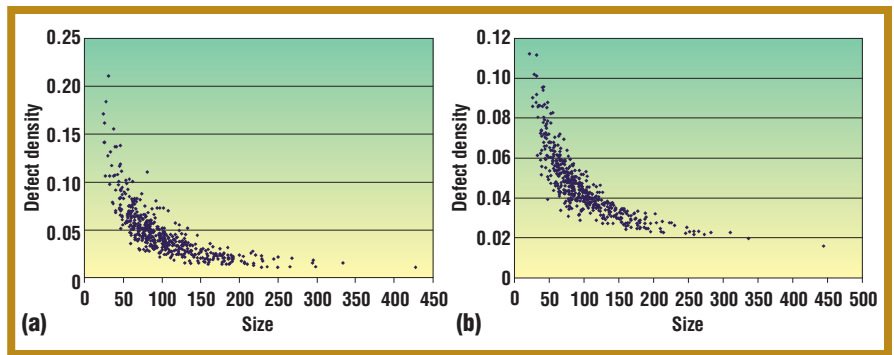


Figure 1. Size vs. defect density for (a) uncorrelated and (b) correlated samples.

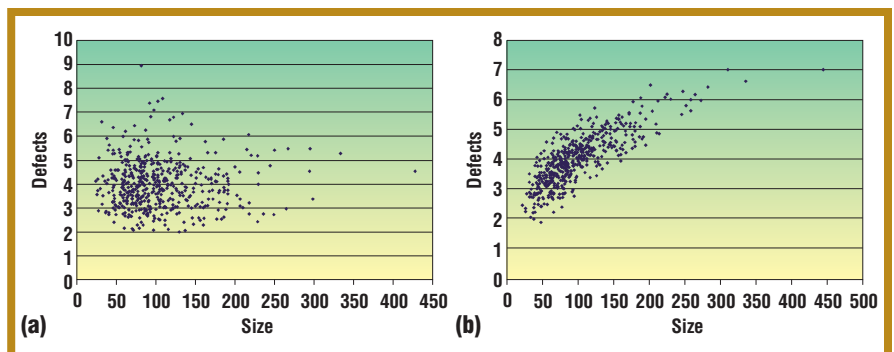


Figure 2. Size vs. defects found for (a) uncorrelated and (b) correlated samples.

Software Boards

I welcome two new members to *Software's* boards. Uwe Zdun is a faculty member in the Vienna University of Technology's Information Systems Institute. Uwe is an active member of the patterns community, a contributor to several open source projects, an expert in distributed-systems design, and author of the book *Remoting Patterns: Foundations of Enterprise, Internet, and Real-time Distributed Object Middleware*. He replaces Philippe Kruchten as the new associate editor in chief for the magazine's Design and Architecture coverage area. Grigori Melnik, a senior product planner in the Patterns and Practices group at Microsoft, joins the advisory board. Grigori has been active in both industry and academia since 1990, with extensive teaching and applied research experience spanning e-business systems, Web engineering, test-driven approaches, empirical software engineering, and software process adoption.

I thank the departing associate editor in chief Philippe Kruchten for his four years of invaluable service. Philippe has been a pillar, and gratefully, will continue to contribute in a new role: as editor of a department on software professionalism to debut in the second half of 2008. Look for his columns in the months to come.

diminishing-returns relationship visible in figure 2b is meaningful. It tells us that the measured system's larger modules tend to be proportionately less defect prone, with implications for distributing limited testing resources among differently sized modules.

Some experts justifiably stress that non-functional relationships are more impor-

tant for making decisions than correlations between base measures. Chayes explains how we can statistically correct for spurious correlations when using simple ratios. However, if the goal is to understand behavioral properties, simple ratios might not be too helpful because they obscure the actual functional forms.

One temporary solution is to adjust a simple ratio metric for any observed scale economies or diseconomies so that values represent something closer to a true ratio scale. For figure 2b, such a metric would be defects squared per unit size (the observations were generated using a quadratic function with a random error). Unfortunately, this solution risks being transient, and therefore impractical. Although functional forms matter, we don't know yet how fluid or stable they are. Are they universal enough to be treated as laws of nature? Until empirical evidence demonstrates that new technologies and processes do not modify these forms, it would be safer to acknowledge the possibility that functional associations between size and defects as well as effort and output vary both over time and across contexts.

Until then, simple ratios, with their interpretation as first derivatives or averages, will exhibit more universality. Some attempts to discover general, persistent laws may prove fruitful. Notably, A. Güneş Koru and his colleagues Dongsong Zhang and Hongfang Liu have been gathering support for a logarithmic relationship between module size and defect proneness (see their PROMISE 2007 paper "Modeling the Effect of Size on Defect Proneness for Open Source Software" at <http://promisedata.org/?cat=130>). If such findings are ultimately shown to be insensitive to contextual factors and future innovations, a revision of traditional definitions of software productivity and quality will be warranted.

So, should software measurement avoid simple ratios and other derived metrics? Not as long as we interpret them at face value, use them intelligently, consider scale effects in their interpretation and usage, and don't discount the value of raw data and contextual information. Derived metrics help us quickly see the big picture, but they are often inadequate, and they can at times be misleading.

If your perspective differs, write to me at hakan.erdogmus@computer.org.

Acknowledgments

Many thanks to A. Güneş Koru and Khaled El-Emam who over the years have illuminated me through engaging discussions.

CALL FOR ARTICLES

Mining Software Archives

PUBLICATION: Jan./Feb. 2009

SUBMISSION DEADLINE: 1 July 2008

As we organize and integrate development activities using software tools, we leave behind automatically accessible data. We can glean a tremendous amount of information from these archives by applying data-mining and machine-learning techniques. The resulting patterns, rules, and insights can increase program quality and programmer productivity.

THIS SPECIAL ISSUE SEEKS ARTICLES ON

- Approaches, applications, tools, search techniques, visualization techniques
- Case studies on extracting data from archives of large, long-lived projects
- Analyzing change patterns, collecting and predicting effort data
- Mining social networks from large software projects
- Techniques for predicting reliability and defects
- Mining requirements documents and predicting requirements volatility
- Prioritizing test suites by leveraging historical test data
- How developers, teams, and managers can leverage mined data in day-to-day work

Articles should have a practical orientation and be written in a style accessible to practitioners. Overly complex, purely research-oriented, or theoretical treatments are not appropriate.

IEEE
Software

GUEST EDITORS:

- Nachiappan Nagappan, Microsoft Research, nachin@microsoft.com
- Andreas Zeller, Saarland University, zeller@acm.org
- Tom Zimmermann, University of Calgary, tz@acm.org

For general author guidelines:
www.computer.org/software/author.htm
For submission details:
software@computer.org