Metrics and Measurement

17-313: Foundations of Software Engineering

https://cmu-313.github.io

Josh Sunshine and **Michael Hilton**

Fall 2026

Gentle Reminder

- Please close laptops
- Tablets with stylus ok

Administriva

- **Problem set posted yesterday**
- **Weekly teamwork surveys WILL BE live**
- **Quiz on TypeScript Sept 15th**
- **Team Repos created. Let us know if you need help**
- **Project 2 posted on website**

  A) [Team Process & Planning] – 70 points – due Monday, September 14th, 11:59pm
  B) [First Sprint] – 95 points – due Monday, September 28th, 11:59pm
  C) [Second Sprint] – 110 points – due Friday, October 9th, 11:59pm

Learning Goals

- Explain the importance of measurement and metrics in Software Engineering
- Provide examples of metrics for software qualities and process
- Apply goal-based frameworks for decision making using metrics
- Identify the limitations and dangers of decisions and incentives based on measurements

Measurement in everyday life

- Economics
  - price, inflation rate, stock price, volume
- Medicine
  - heart rate, blood pressure, body temperature, ECG
- Engineering
  - force, torque, heat transfer coefficient, thermal efficiency
- Natural sciences
  - AQI, carbon footprint, soil pH

NS New Scientist

Ants use pedometers to find home

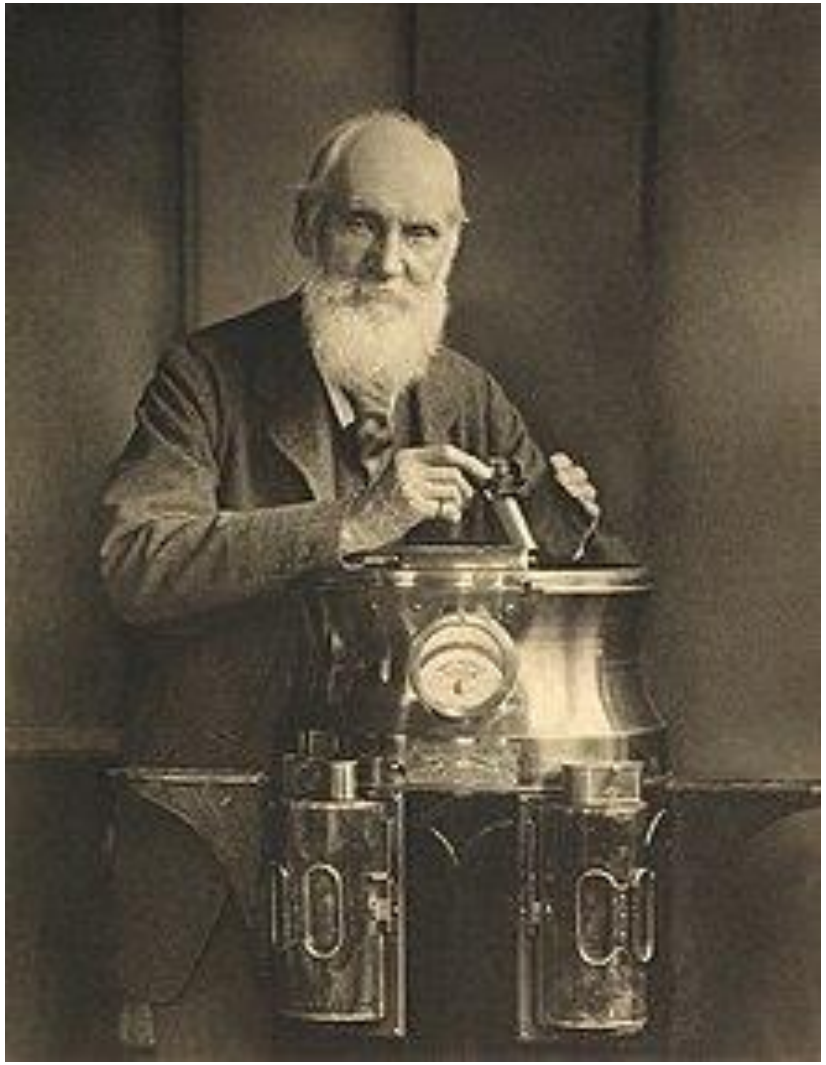
An experiment that involves attaching stilts to ants' legs reveals that the insects somehow keep a record of how many steps they take.

Jun 29, 2006

“To measure is to know; if you can not measure it, you can not improve it”

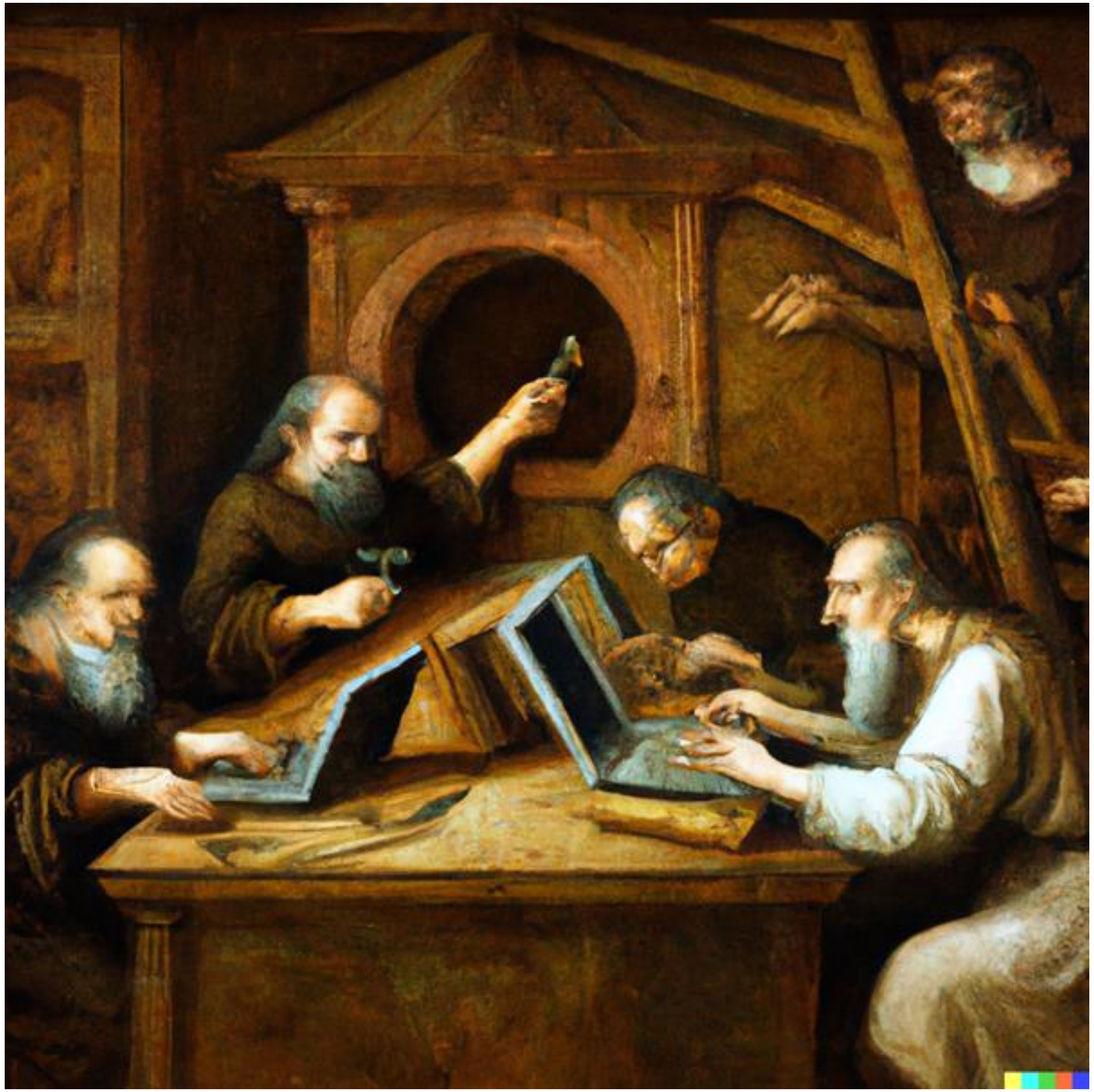
William Thomson, Lord Kelvin

$$K = \left(\frac{5}{9}(F - 32)\right) + 273.15$$



Software Development…

before Software Engineering



by DALL-E

Software Engineering

Principles and practices (technical and non-technical) for confidently building high-quality software

Software Engineering

Principles and practices (technical and non-technical) for confidently building **high-quality software**

What does this mean?
How do we know?

We need metrics and measurement!

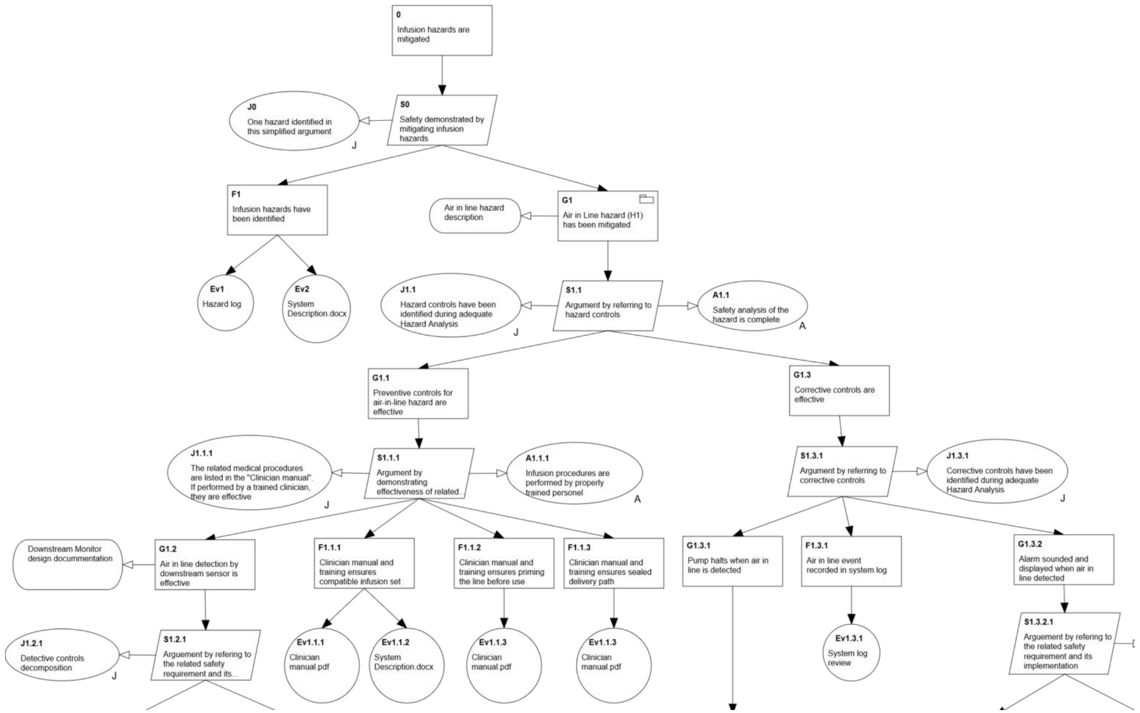
Safety Assurance Cases and AVs

Executing our safety case

At Aurora, we are using a safety case-based approach to inform, guide, and determine that our technology is acceptably safe to operate on public roads.

A safety case is a logical argument, supported by evidence, intended to justify that a system is acceptably safe for a specific application in a specific operating environment. A structured argument includes a specific claim—such as that our self-driving vehicles are acceptably safe to operate on public roads—that is broken down into sub-claims, which are then ultimately supported by evidence. We believe a safety case is the most effective and efficient path to safe driverless operations, and is imperative for any company looking to safely deliver commercial-ready self-driving vehicles at scale.

and ready to take over as necessary to ensure operational safety. Therefore our tailored safety case for this use case includes claims focused on vehicle controllability and vehicle operator hiring, training, and operational procedures, among others. However, as we approach the point of removing vehicle operators from the vehicles, these vehicle operator-centric claims will no longer be relevant. At that point, we will have completed a tailored safety case to include other claims from the Safety Case Framework related to demonstrating acceptably safe driverless operations within our ODD.



https://aurora.tech/vssa

AV Software is ______________________

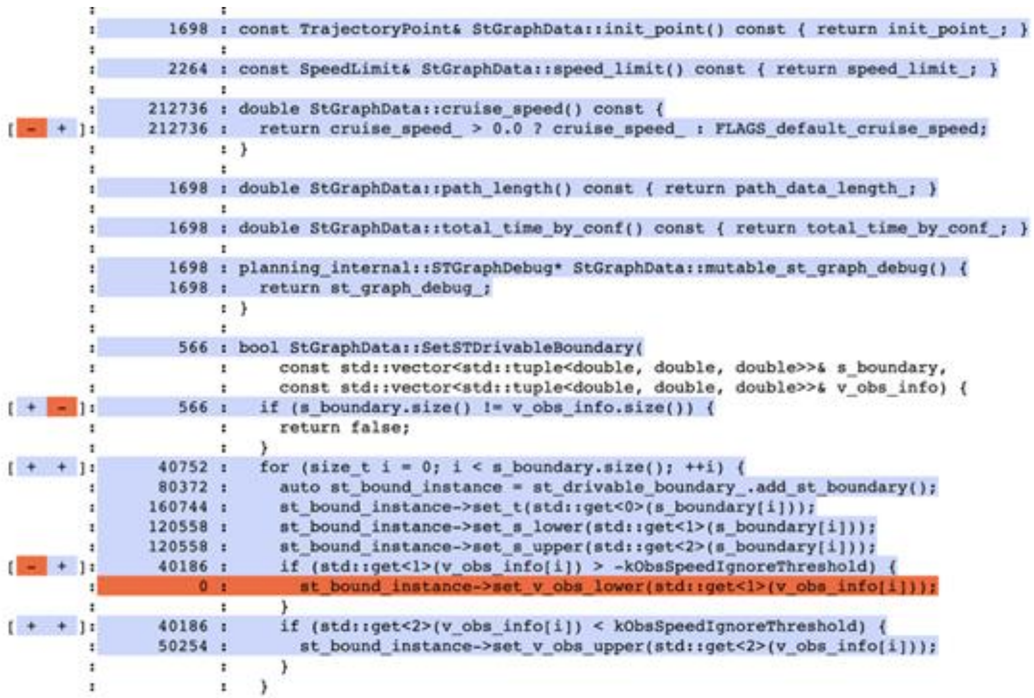
By what Metrics can we judge AV software (e.g., safety)?

Participation Activity

- What are some ways we could measure if the code is safe?
- Come up with two metrics, and then discuss with your neighbor.
- For each metric come up with a pro and a con for that metric.

(1) Code Coverage

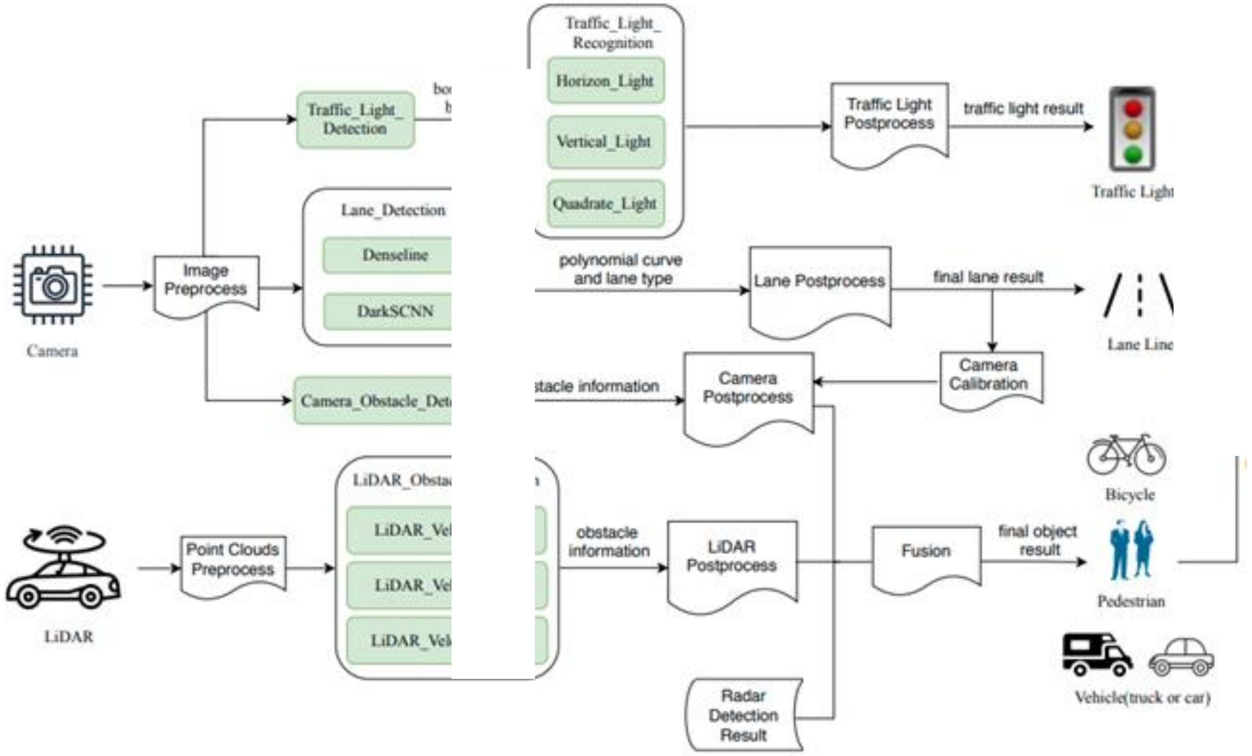
- Amount of code executed during testing.
- Statement coverage, line coverage, branch coverage, etc.
- E.g., 75% branch coverage 3/4 if-else outcomes have been executed



```
        :          :
        :     1698 : const TrajectoryPoint& StGraphData::init_point() const { return init_point_; }
        :          :
        :     2264 : const SpeedLimit& StGraphData::speed_limit() const { return speed_limit_; }
        :          :
        :   212736 : double StGraphData::cruise_speed() const {
[ - + ]:   212736 :   return cruise_speed_ > 0.0 ? cruise_speed_ : FLAGS_default_cruise_speed;
        :          : }
        :          :
        :     1698 : double StGraphData::path_length() const { return path_data_length_; }
        :          :
        :     1698 : double StGraphData::total_time_by_conf() const { return total_time_by_conf_; }
        :          :
        :     1698 : planning_internal::STGraphDebug* StGraphData::mutable_st_graph_debug() {
        :     1698 :   return st_graph_debug_;
        :          : }
        :          :
        :      566 : bool StGraphData::SetSTDrivableBoundary(
        :          :     const std::vector<std::tuple<double, double, double>>& s_boundary,
        :          :     const std::vector<std::tuple<double, double, double>>& v_obs_info) {
[ + - ]:      566 :   if (s_boundary.size() != v_obs_info.size()) {
        :          :     return false;
        :          :   }
[ + + ]:    40752 :   for (size_t i = 0; i < s_boundary.size(); ++i) {
        :    80372 :     auto st_bound_instance = st_drivable_boundary_.add_st_boundary();
        :   160744 :     st_bound_instance->set_t(std::get<0>(s_boundary[i]));
        :   120558 :     st_bound_instance->set_s_lower(std::get<1>(s_boundary[i]));
        :   120558 :     st_bound_instance->set_s_upper(std::get<2>(s_boundary[i]));
[ - + ]:    40186 :     if (std::get<1>(v_obs_info[i]) > -kObsSpeedIgnoreThreshold) {
        :        0 :       st_bound_instance->set_v_obs_lower(std::get<1>(v_obs_info[i]));
        :          :     }
[ + + ]:    40186 :     if (std::get<2>(v_obs_info[i]) < kObsSpeedIgnoreThreshold) {
        :    50254 :       st_bound_instance->set_v_obs_upper(std::get<2>(v_obs_info[i]));
        :          :     }
        :          :   }
```

(2) Model Accuracy

- Train machine-learning models on labelled data (sensor data + ground truth)
- Compute accuracy on a separate labelled test set
- E.g., 90% accuracy implies that object recognition is right for 90% of the test inputs.



Source: Peng et al. ESEC/FSE'20

(3) Failure Rate

- Frequency of crashes / fatalities
- Per 1,000 rides, per million miles, per month (in the news)

TRANSPO / WAYMO / TECH

Waymo's driverless cars were involved in two crashes and 18 'minor contact events' over 1 million miles

Image: Allen J. Schaben / Lo

/ The Alphabet-owned company pulls back the curtain on more stats from its public road testing. Of the 20 incidents, only two met the federal government's reporting criteria, and no one was injured.

By Andrew J. Hawkins, transportation editor with 10+ years of experience who covers EVs, public transportation, and aviation. His work has appeared in The New York Daily News and City & State.
Feb 28, 2023, 8:00 PM GMT+3 | 1 Comment / 1 New

'Complete meltdown': Driverless cars in San Francisco stall causing a traffic jam

By Jordan Valinsky, CNN Business
Updated 3:45 PM EDT, Mon August 14, 2023

A Cruise autonomous taxi in San Francisco.

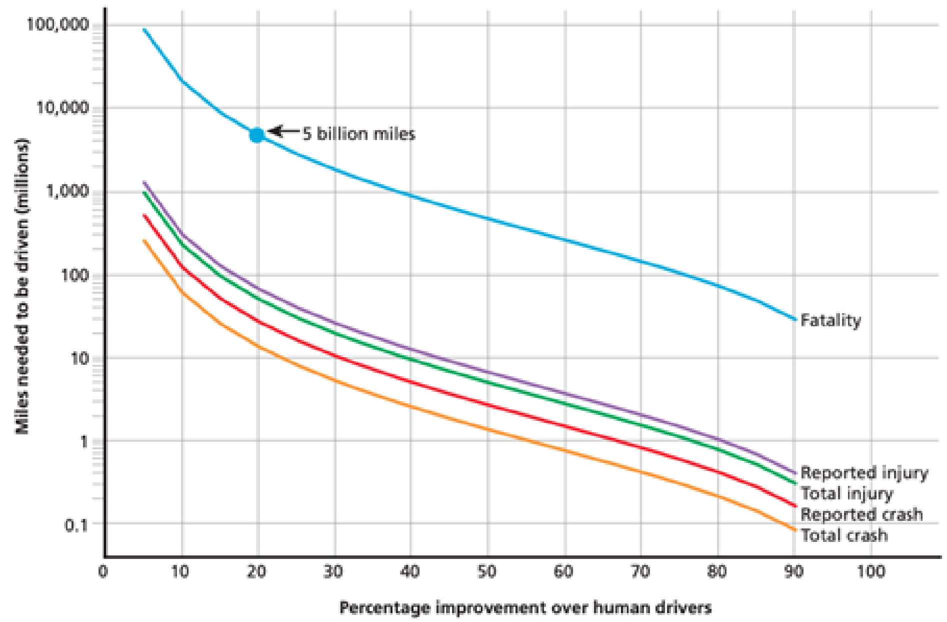
(4) Mileage

Driving to Safety

How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?

Nidhi Kalra, Susan M. Paddock

Figure 3. Miles Needed to Demonstrate with 95% Confidence that the Autonomous Vehicle Failure Rate Is Lower than the Human Driver Failure Rate



Source: waymo.com/safety (September 2021)

Updated numbers

Rider-only (RO) miles driven

Through March 2026, Waymo has driven 220.6M rider-only miles without a human driver

The Waymo Driver has tens of millions miles of real-world driving experience. This dashboard shows rider-only miles – miles that Waymo has driven without a human driver — in cities where we operate our ride-hailing service, Waymo.

Learn about our methodology

Locations	RO Miles through March 2026
Los Angeles	51.816M
San Francisco Bay Area	67.078M
Phoenix	80.551M
Austin	15.789M
Atlanta	5.379M

Waymo Driver compared to human benchmarks

This table shows how many fewer RO crashes Waymo had (regardless of who was at fault) compared to human drivers with the average benchmark crash rate if they were to drive the same distance in the areas we operate. Results have been rounded to the nearest whole number.

Learn about our methodology

Compared to an average human driver over the same distance in our operating cities, the Waymo Driver had

Overall crash reduction

94%
Fewer serious injury or worse crashes
↓ 47 FEWER

82%
Fewer airbag deployment in any vehicle crashes
↓ 305 FEWER

82%
Fewer injury-causing crashes
↓ 707 FEWER

Crash reductions involving injuries to Vulnerable Road Users

93%
Fewer pedestrian crashes with injuries
↓ 76 FEWER

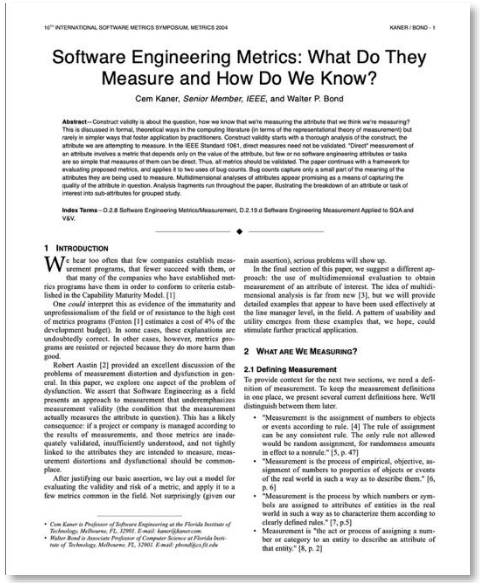
84%
Fewer cyclist crashes with Injuries
↓ 48 FEWER

84%
Fewer motorcycle crashes with injuries
↓ 32 FEWER

What is Measurement?

What is Measurement?

- “Measurement is the empirical, **objective assignment of numbers, according to a rule** derived from a model or theory, to **attributes of objects or events** with the intent of describing them”
  Kaner, Bond, *Software Engineering Metrics: What Do They Measure and How Do We Know?*



Software Engineering Metrics: What Do They Measure and How Do We Know?

Cem Kaner, *Senior Member, IEEE*, and Walter P. Bond

Abstract—Construct validity is about the question, how we know that we're measuring the attribute that we think we're measuring? This is discussed in formal, theoretical ways in the computing literature (in terms of the representational theory of measurement) but rarely in simpler ways that foster application by practitioners. Construct validity starts with a thorough analysis of the construct, the attribute we are attempting to measure. In the IEEE Standard 1061, direct measures need not be validated. "Direct" measurement of an attribute involves a metric that depends only on the value of the attribute, but few or no software engineering attributes or tasks are so simple that measures of them can be direct. Thus, all metrics should be validated. The paper continues with a framework for evaluating proposed metrics, and applies it to two uses of bug counts. Bug counts capture only a small part of the meaning of the attributes they are being used to measure. Multidimensional analyses of attributes appear promising as a means of capturing the quality of the attribute in question. Analysis fragments run throughout the paper, illustrating the breakdown of an attribute or task of interest into sub-attributes for grouped study.

Index Terms—D.2.8 Software Engineering Metrics/Measurement, D.2.19.d Software Engineering Measurement Applied to SQA and V&V.

1 Introduction

We hear too often that few companies establish measurement programs, that fewer succeed with them, or that many of the companies who have established metrics programs have them in order to conform to criteria established in the Capability Maturity Model. [1]

One *could* interpret this as evidence of the immaturity and unprofessionalism of the field or of resistance to the high cost of metrics programs (Fenton [1] estimates a cost of 4% of the development budget). In some cases, these explanations are undoubtedly correct. In other cases, however, metrics programs are resisted or rejected because they do more harm than good.

Robert Austin [2] provided an excellent discussion of the problems of measurement distortion and dysfunction in general. In this paper, we explore one aspect of the problem of dysfunction. We assert that Software Engineering as a field presents an approach to measurement that underemphasizes measurement validity (the condition that the measurement actually measures the attribute in question). This has a likely consequence: if a project or company is managed according to the results of measurements, and those metrics are inadequately validated, insufficiently understood, and not tightly linked to the attributes they are intended to measure, measurement distortions and dysfunctional should be commonplace.

After justifying our basic assertion, we lay out a model for evaluating the validity and risk of a metric, and apply it to a few metrics common in the field. Not surprisingly (given our main assertion), serious problems will show up.

In the final section of this paper, we suggest a different approach: the use of multidimensional evaluation to obtain measurement of an attribute of interest. The idea of multidimensional analysis is far from new [3], but we will provide detailed examples that appear to have been used effectively at the line manager level, in the field. A pattern of usability and utility emerges from these examples that, we hope, could stimulate further practical application.

2 What are We Measuring?

2.1 Defining Measurement

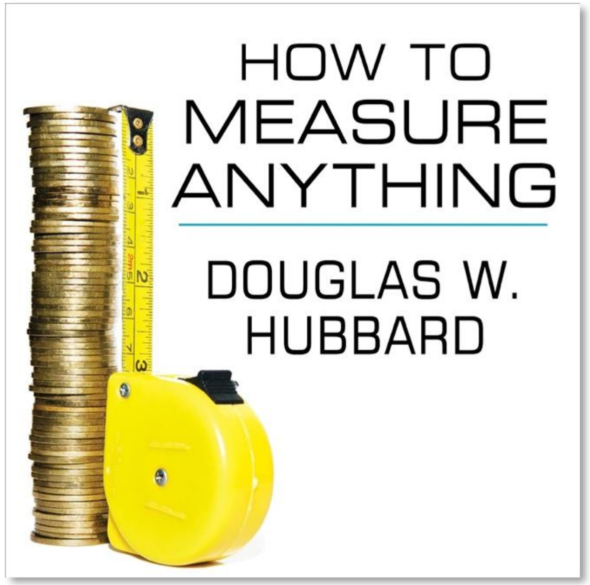
To provide context for the next two sections, we need a definition of measurement. To keep the measurement definitions in one place, we present several current definitions here. We'll distinguish between them later.

- "Measurement is the assignment of numbers to objects or events according to rule. [4] The rule of assignment can be any consistent rule. The only rule not allowed would be random assignment, for randomness amounts in effect to a nonrule." [5, p. 47]
- "Measurement is the process of empirical, objective, assignment of numbers to properties of objects or events of the real world in such a way as to describe them." [6, p. 6]
- "Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to characterize them according to clearly defined rules." [7, p.5]
- Measurement is "the act or process of assigning a number or category to an entity to describe an attribute of that entity." [8, p. 2]

• *Cem Kaner is Professor of Software Engineering at the Florida Institute of Technology, Melbourne, FL, 32901. E-mail: kaner@kaner.com.*
• *Walter Bond is Associate Professor of Computer Science at Florida Institute of Technology, Melbourne, FL, 32901. E-mail: pbond@cs.fit.edu*

What is Measurement?

- **"A quantitatively expressed reduction of uncertainty based on one or more observations."**
  Hubbard, *How to Measure Anything …*



What is Measurement?

- "A software quality metric is a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given attribute that affects its quality."

IEEE 1061

Metric

MEASURES

Attribute

DESCRIBES

Entity

What is Measurement?

Method to obtain a number or symbol → **Metric**

MEASURES

Quality of interest → **Attribute**

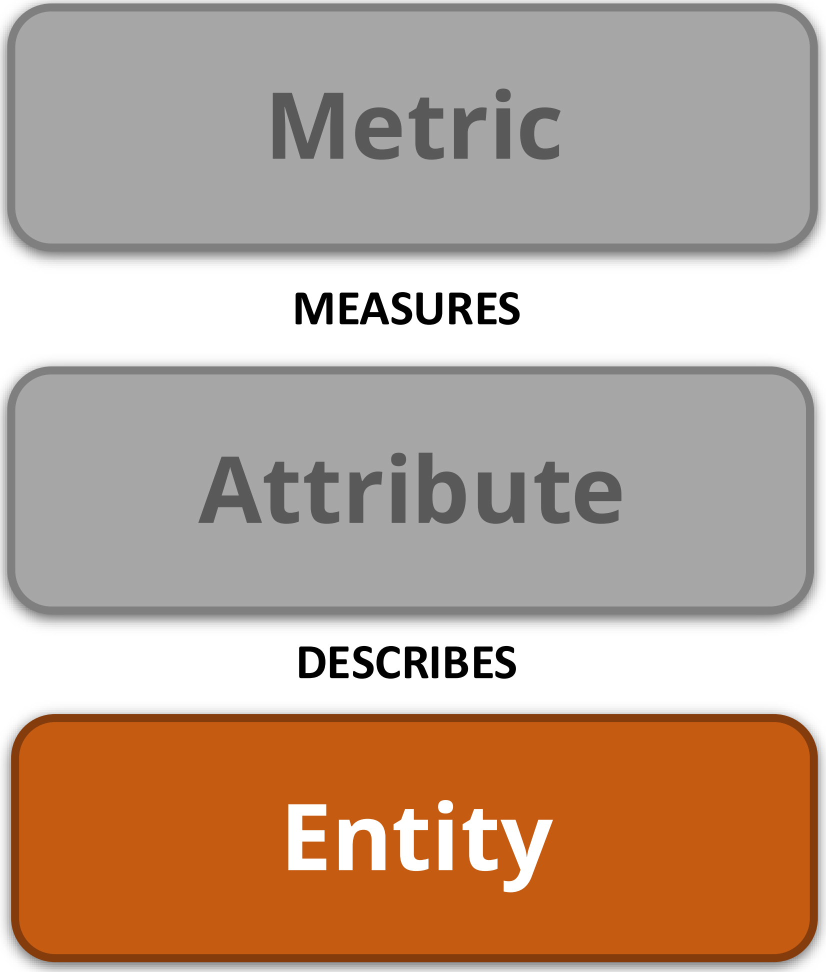
DESCRIBES

Object or process → **Entity**

Entities represent an Object or Process

Examples

- Software Product
- Modules
- Software Development Process
- People



Attributes represent Qualities of Interest

Examples for Software Quality

- Reliability
- Security
- Scalability
- Extensibility
- Portability
- Availability
- Safety
- Observability
- Accuracy
- Robustness
- Resilience
- Timeliness
- Responsiveness
- Intuitiveness

Metric

MEASURES

Attribute

DESCRIBES

Entity

Attributes represent Qualities of Interest

Examples for Software Dev. Process

- Development efficiency
- Meeting efficiency
- Conformance to process
- Accuracy of predictions
- Fairness in decision making
- Regulatory compliance
- On-time release
- …

Metric

MEASURES

Attribute

DESCRIBES

Entity

Attributes represent Qualities of Interest

Examples for People

Developers

- Productivity
- Agility
- Well-Being + Job Satisfaction
- Communication and Collaboration
- Creativity
- Morale
- Regulatory Compliance

End Users

- Product Satisfaction
- Ease of Use
- Feature Usage
- Regulatory Compliance

Metric

MEASURES

Attribute

DESCRIBES

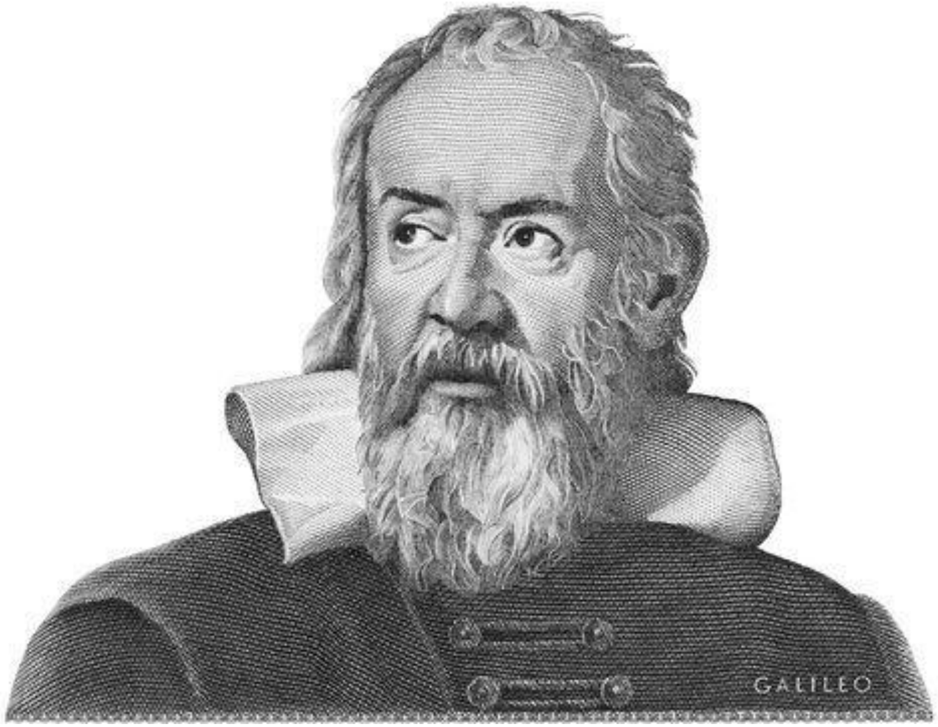
Entity

What about Non-Trivial Qualities?

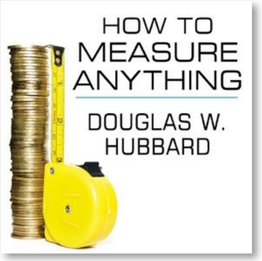
- **Software**
  - code elegance
  - code maintainability
- **Process**
  - development efficiency
  - fairness in decision making
- **Team**
  - productivity
  - collaboration
  - creativity

"Measure what is measurable, and make measurable what is not so."

Galileo Galilei

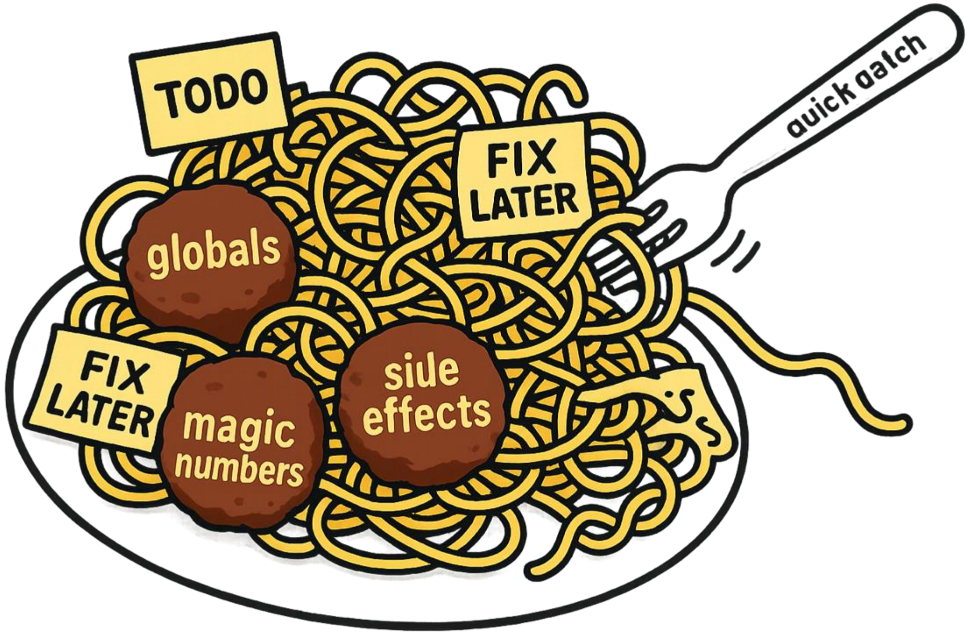


Everything is Measurable



- If we care about X, then, by definition, X **must be detectable**
  - how could we care about things like *quality*, *risk*, *security*, or *public image* if these things were totally undetectable, directly or indirectly?
  - if we have reason to care about some unknown quantity, it is because we think it corresponds to desirable or undesirable results in some way
- If X is detectable, then it **must be detectable in some amount**
  - if you can observe a thing at all, you can observe more of it or less of it
- If we can observe it in some amount, then it **must be measurable**

Douglas Hubbard, How to Measure Anything, 2010



Example:
Code Complexity

thanks to ChatGPT for the terrible image

Number of Lines

- Easy to Measure!   wc –l file1 file2…

LOC (aprox)	projects
450	Expression Evaluator
2,000	Sudoku
300,000	SQLite
1,000,000	Git
3,000,000	Kubernetes
15,000,000	gcc
29,000,000	Linux kernel
~50,000.000	Windows 11
2,000,000,000	Google (MonoRepo, reported)

Normalizing Lines of Code

- Ignore comments and empty lines
- Ignore lines < 2 characters
- Pretty print source code first
- Count statements (logical lines of code)
- See also: cloc

```
for (i = 0; i < 100; i += 1) printf("hello"); /* How many lines of code is this? */
```

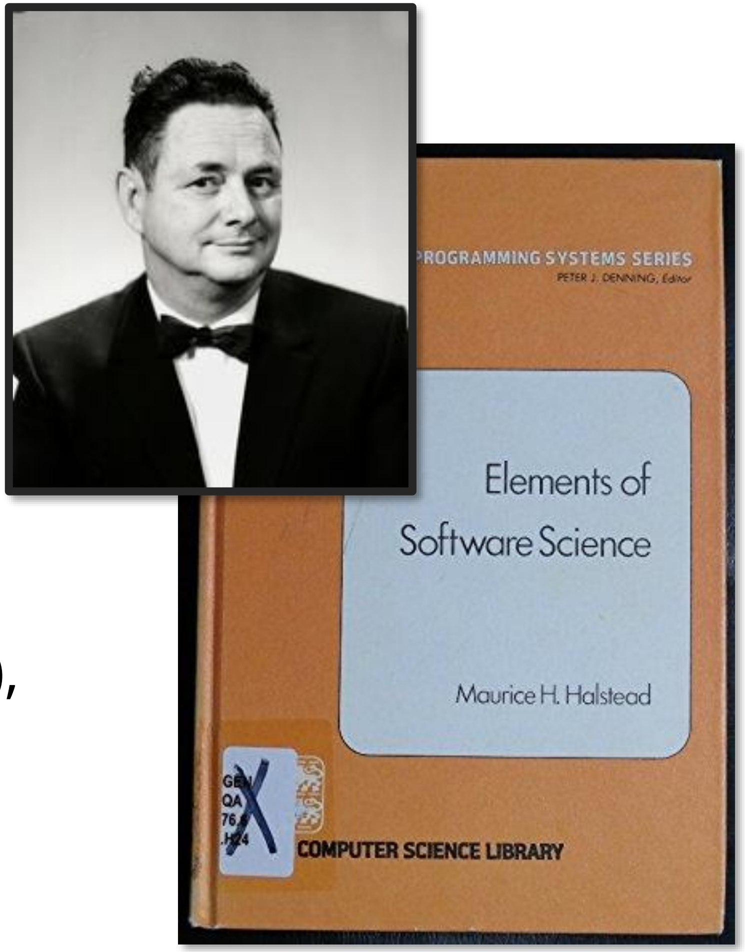
```
/* How many lines of code is this? */

for (
        i = 0;
        i < 100;
        i += 1
   ) {
        printf("hello");
}
```

Normalization by Language

Language	Statement factor (productivity)	Line factor
C	1	1
C++	2.5	1
Fortran	2	0.8
Java	2.5	1.5
Perl	6	6
Smalltalk	6	6.25
Python	6	6.5

Source: "Code Complete: A Practical Handbook of Software Construction", S. McConnell, Microsoft Press (2004)
and http://www.codinghorror.com/blog/2005/08/are-all-programming-languages-the-same.html u.a.

Halstead’s Metrics (1977)



- Based on number of operators (e.g., print, +, -) and operands (name, “maurice”)
  - n_1 and n_1: # of distinct operators and operands
  - N_1 and N_2: # of instances of operators and operands

- Derived metrics include vocabulary (*N*), length (*n*), volume (*V*), difficulty (*D*), effort (*E*), time to write (*T*), number of bugs (*B*)

$$D = \frac{n_1}{2} + \frac{N_2}{n_2}$$

$$V = N \times \log_2 n$$

$$E = D \times V$$

$$N = N_1 + N_2$$

$$T = \frac{E}{18}$$

$$B = \frac{V}{3000}$$

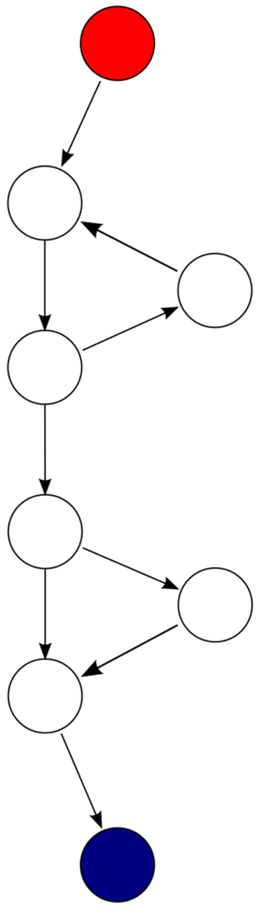
$$n = n_1 + n_2$$

McCabe Cyclomatic Complexity (1976)

- Computed according to control flow graph
  - tells you how many tests you need for full branch coverage

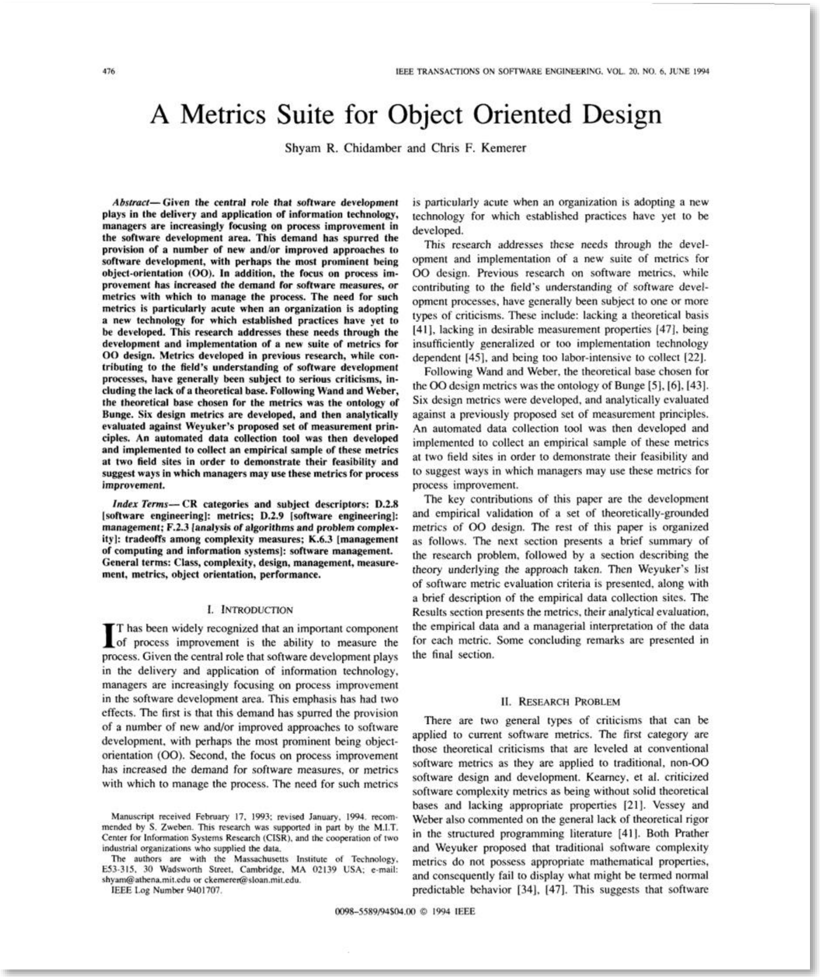
- Equal to **number of decision points + 1**
  - if, while, do-while, ?:, catch, switch, case
  - && and || in if condition

"For each module, either limit cyclomatic complexity to [X] or provide a written explanation of why the limit was exceeded."
NIST Structured Testing methodology



Object-Oriented Metrics (1994)

- Number of Methods per Class
- Depth of Inheritance Tree
- Number of Child Classes
- Coupling between Object Classes
- Calls to Methods in Unrelated Classes
- …



A Metrics Suite for Object Oriented Design

Shyam R. Chidamber and Chris F. Kemerer

***Abstract*— Given the central role that software development plays in the delivery and application of information technology, managers are increasingly focusing on process improvement in the software development area. This demand has spurred the provision of a number of new and/or improved approaches to software development, with perhaps the most prominent being object-orientation (OO). In addition, the focus on process improvement has increased the demand for software measures, or metrics with which to manage the process. The need for such metrics is particularly acute when an organization is adopting a new technology for which established practices have yet to be developed. This research addresses these needs through the development and implementation of a new suite of metrics for OO design. Metrics developed in previous research, while contributing to the field's understanding of software development processes, have generally been subject to serious criticisms, including the lack of a theoretical base. Following Wand and Weber, the theoretical base chosen for the metrics was the ontology of Bunge. Six design metrics are developed, and then analytically evaluated against Weyuker's proposed set of measurement principles. An automated data collection tool was then developed and implemented to collect an empirical sample of these metrics at two field sites in order to demonstrate their feasibility and suggest ways in which managers may use these metrics for process improvement.**

***Index Terms*— CR categories and subject descriptors: D.2.8 [software engineering]: metrics; D.2.9 [software engineering]: management; F.2.3 [*analysis of algorithms and problem complexity*]: tradeoffs among complexity measures; K.6.3 [management of computing and information systems]: software management. General terms: Class, complexity, design, management, measurement, metrics, object orientation, performance.**

I. Introduction

It has been widely recognized that an important component of process improvement is the ability to measure the process. Given the central role that software development plays in the delivery and application of information technology, managers are increasingly focusing on process improvement in the software development area. This emphasis has had two effects. The first is that this demand has spurred the provision of a number of new and/or improved approaches to software development, with perhaps the most prominent being object-orientation (OO). Second, the focus on process improvement has increased the demand for software measures, or metrics with which to manage the process. The need for such metrics is particularly acute when an organization is adopting a new technology for which established practices have yet to be developed.

This research addresses these needs through the development and implementation of a new suite of metrics for OO design. Previous research on software metrics, while contributing to the field's understanding of software development processes, have generally been subject to one or more types of criticisms. These include: lacking a theoretical basis [41], lacking in desirable measurement properties [47], being insufficiently generalized or too implementation technology dependent [45], and being too labor-intensive to collect [22].

Following Wand and Weber, the theoretical base chosen for the OO design metrics was the ontology of Bunge [5], [6], [43]. Six design metrics were developed, and analytically evaluated against a previously proposed set of measurement principles. An automated data collection tool was then developed and implemented to collect an empirical sample of these metrics at two field sites in order to demonstrate their feasibility and to suggest ways in which managers may use these metrics for process improvement.

The key contributions of this paper are the development and empirical validation of a set of theoretically-grounded metrics of OO design. The rest of this paper is organized as follows. The next section presents a brief summary of the research problem, followed by a section describing the theory underlying the approach taken. Then Weyuker's list of software metric evaluation criteria is presented, along with a brief description of the empirical data collection sites. The Results section presents the metrics, their analytical evaluation, the empirical data and a managerial interpretation of the data for each metric. Some concluding remarks are presented in the final section.

II. Research Problem

There are two general types of criticisms that can be applied to current software metrics. The first category are those theoretical criticisms that are leveled at conventional software metrics as they are applied to traditional, non-OO software design and development. Kearney, et al. criticized software complexity metrics as being without solid theoretical bases and lacking appropriate properties [21]. Vessey and Weber also commented on the general lack of theoretical rigor in the structured programming literature [41]. Both Prather and Weyuker proposed that traditional software complexity metrics do not possess appropriate mathematical properties, and consequently fail to display what might be termed normal predictable behavior [34], [47]. This suggests that software

Manuscript received February 17, 1993; revised January, 1994, recommended by S. Zweben. This research was supported in part by the M.I.T. Center for Information Systems Research (CISR), and the cooperation of two industrial organizations who supplied the data.

The authors are with the Massachusetts Institute of Technology, E53-315, 30 Wadsworth Street, Cambridge, MA 02139 USA; e-mail: shyam@athena.mit.edu or ckemerer@sloan.mit.edu.

IEEE Log Number 9401707.

Takeaways: Code Complexity

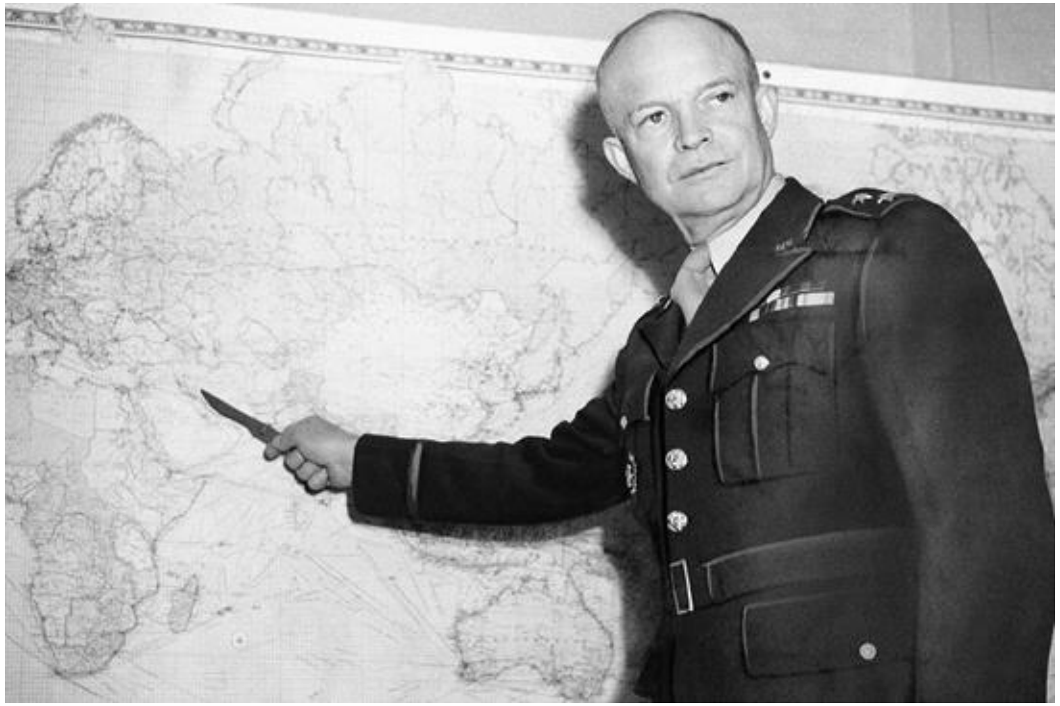
- There are **lots of complexity metrics** that measure different things, which may or may not be what we care about
  - there is no single value that fully captures “code complexity”
  - most of these metrics are confounded by code size!

- We need to be **intentional** about what we are measuring
  - how hard is this code to read?
  - how difficult is this code to maintain?
  - does this function need to be refactored?

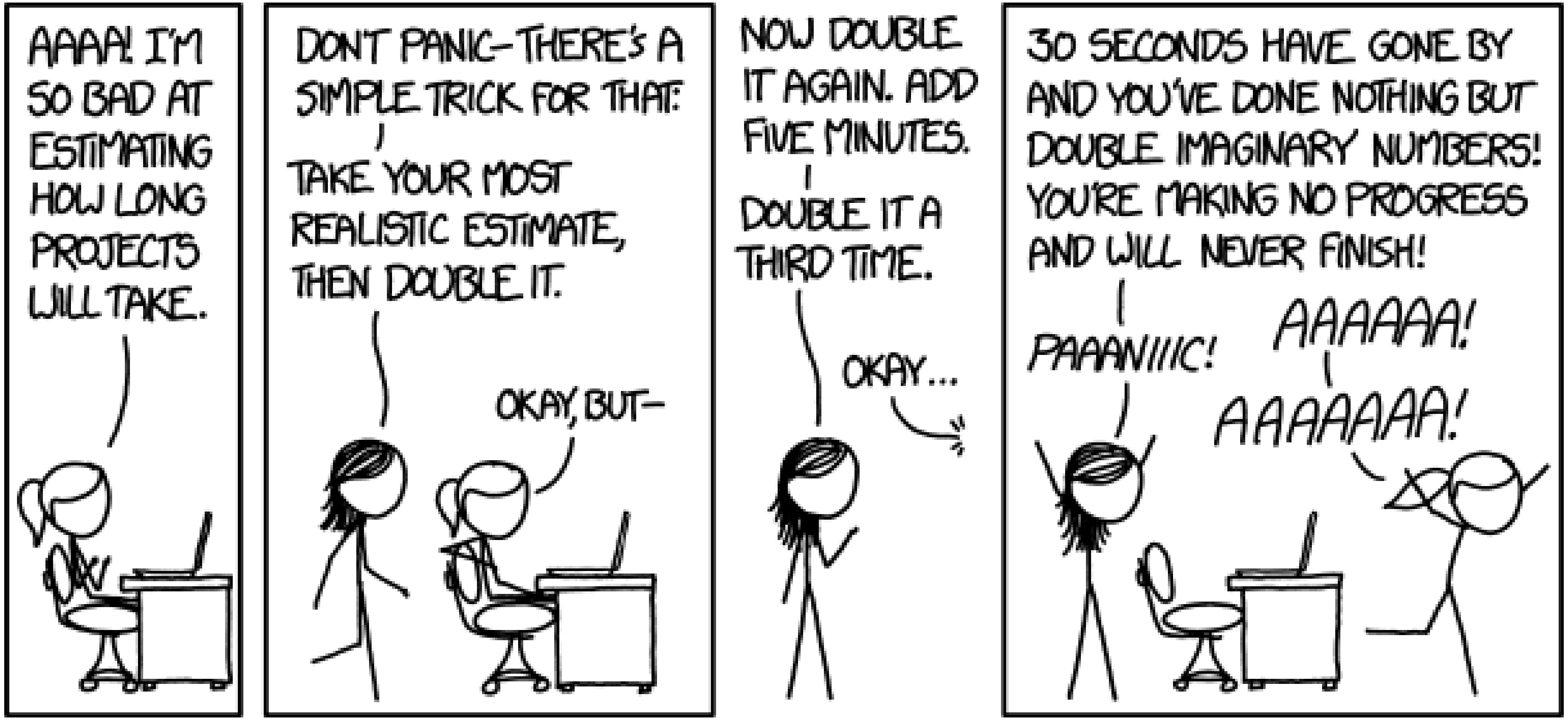
Example 2:
Time Estimation

"Plans are nothing, planning is everything"

-Dwight D. Eisenhower

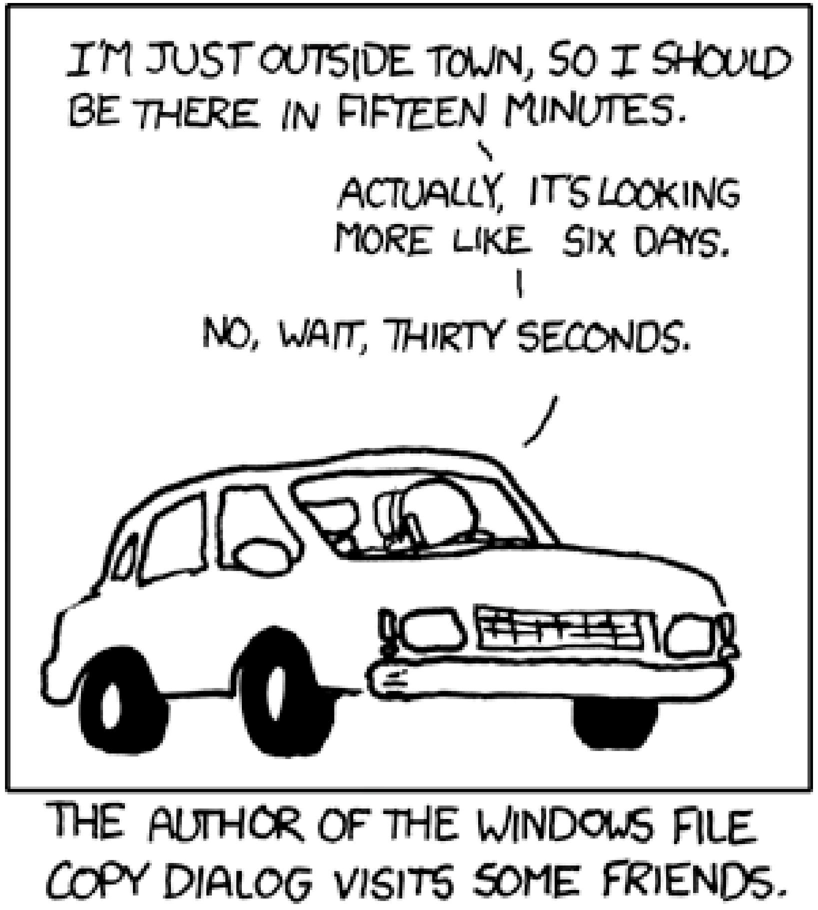


Time estimation



https://xkcd.com/1658/

Time estimation

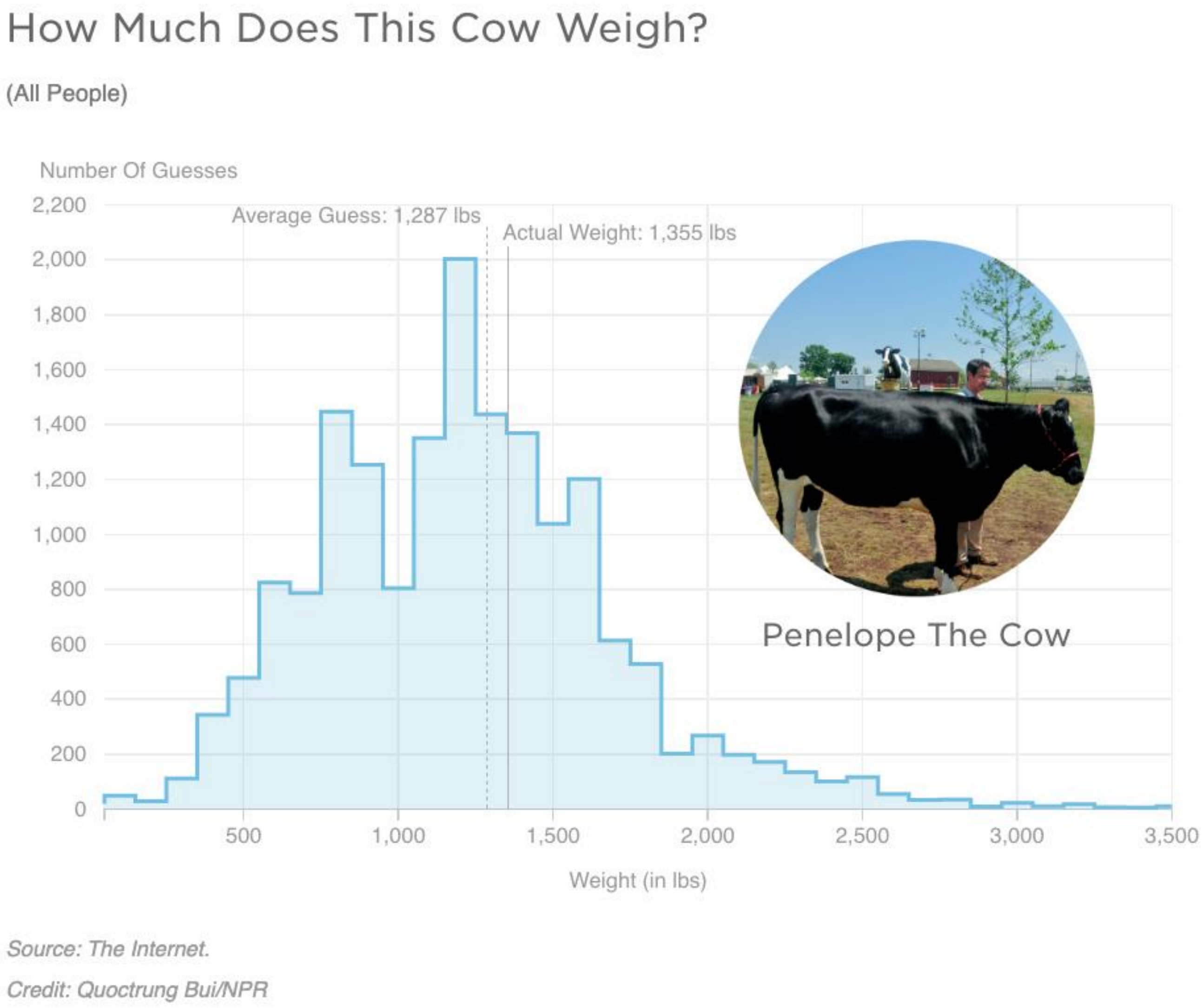


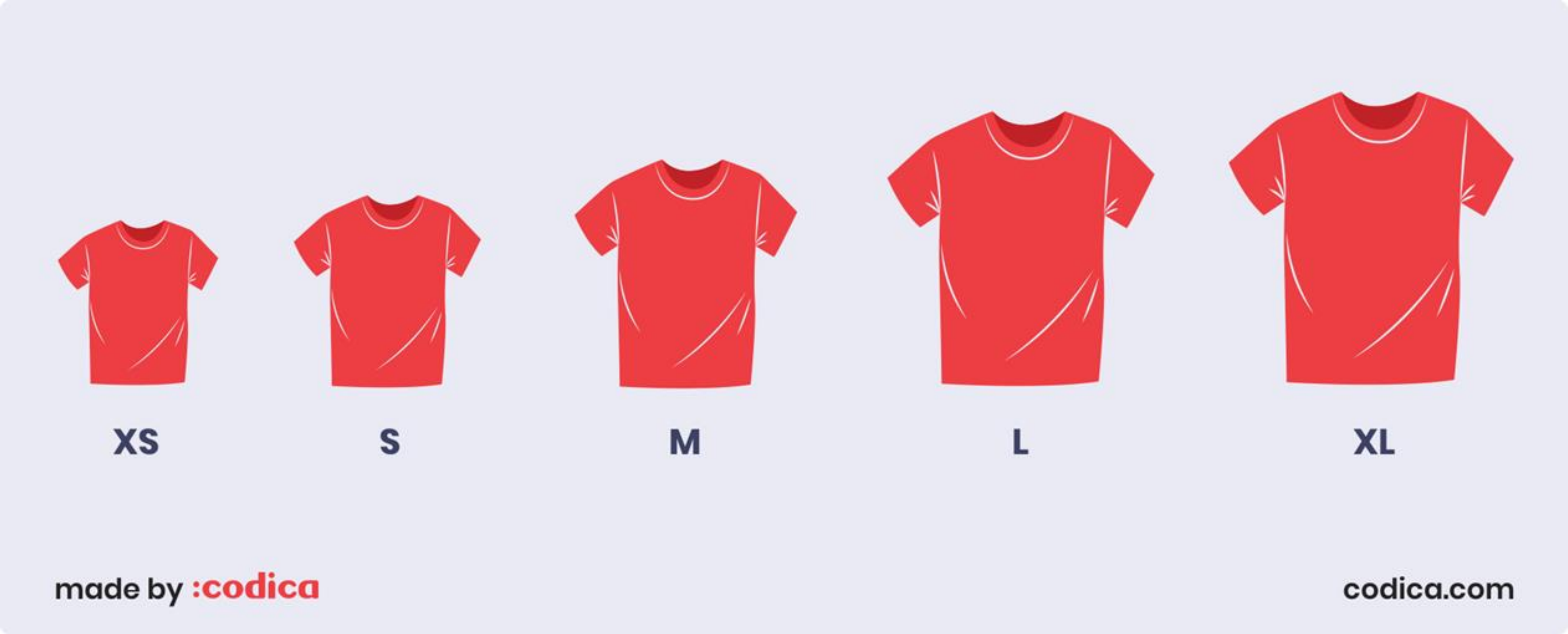
https://xkcd.com/612

Improving Time Estimates

- Prevent conformity bias
- Do you have a comparable experience to base an estimate on?
- How much design do you need for each task?
- Break down the task into smaller tasks and estimate them.

Wisdom of the Crowd



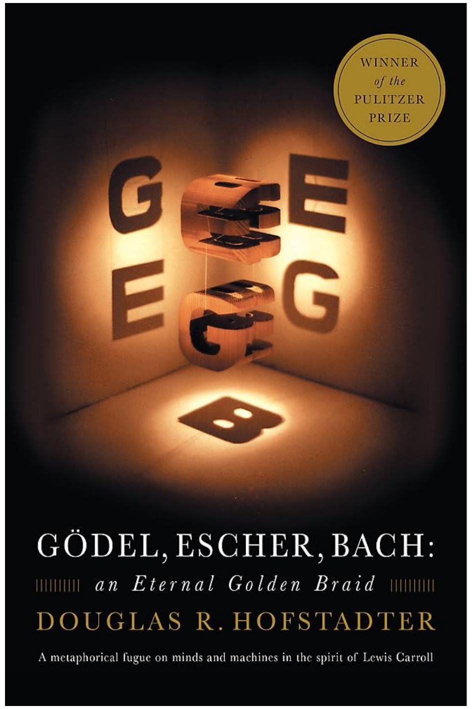

XS
S
M
L
XL
made by :codica
codica.com


1
2
3
5
8
13
21

$\times \pi$

Hofstadter’s Law

“It always takes longer than you expect, even when you take into account Hofstadter’s Law”



Is Estimation Evil?


About Search Site Categories

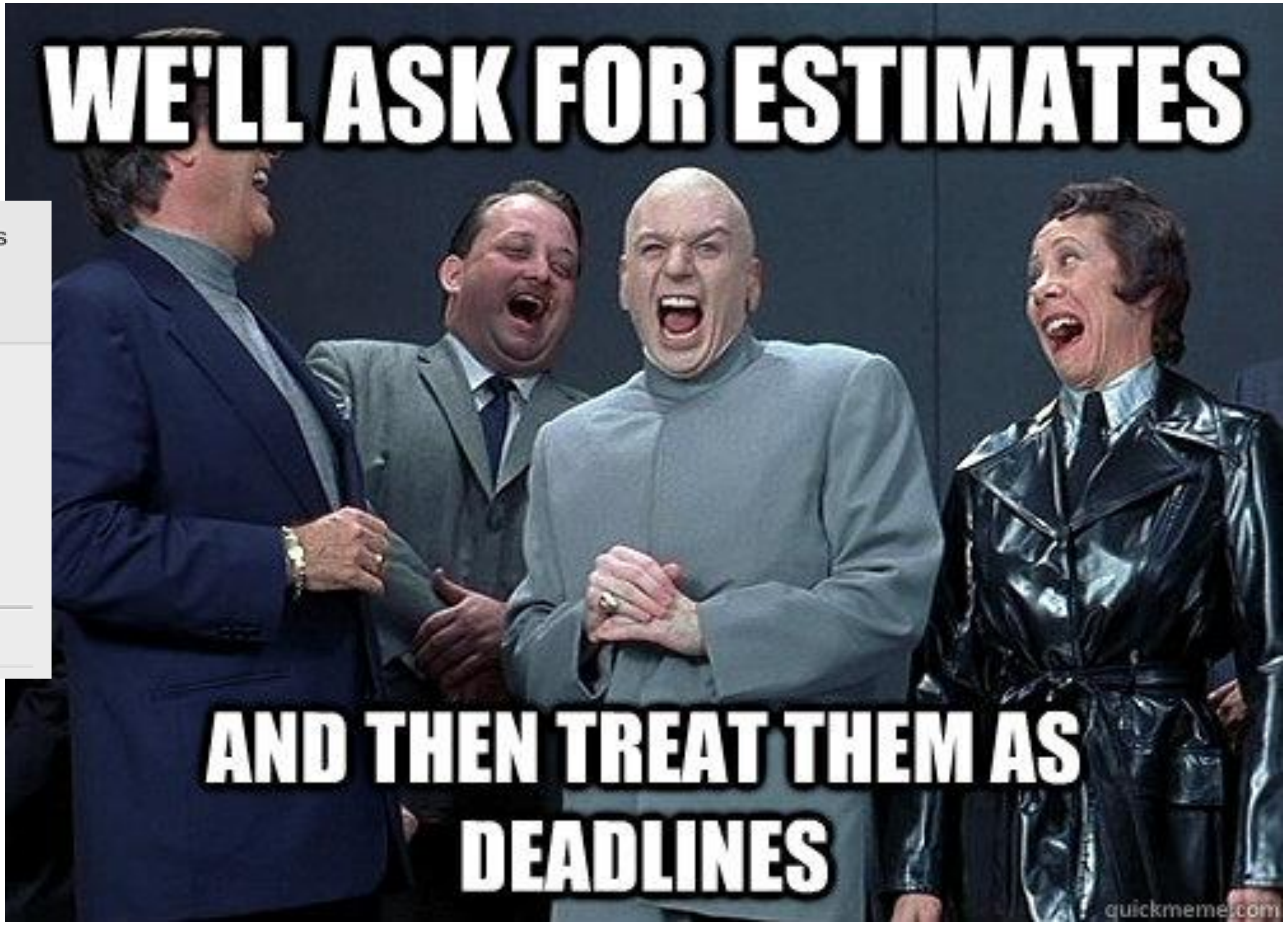
Estimation is Evil

© Feb 1, 2013 • [Agile-Related, estimation]

The following article is recovered from the February 2013 issue of the Pragmatic Programmers magazine.

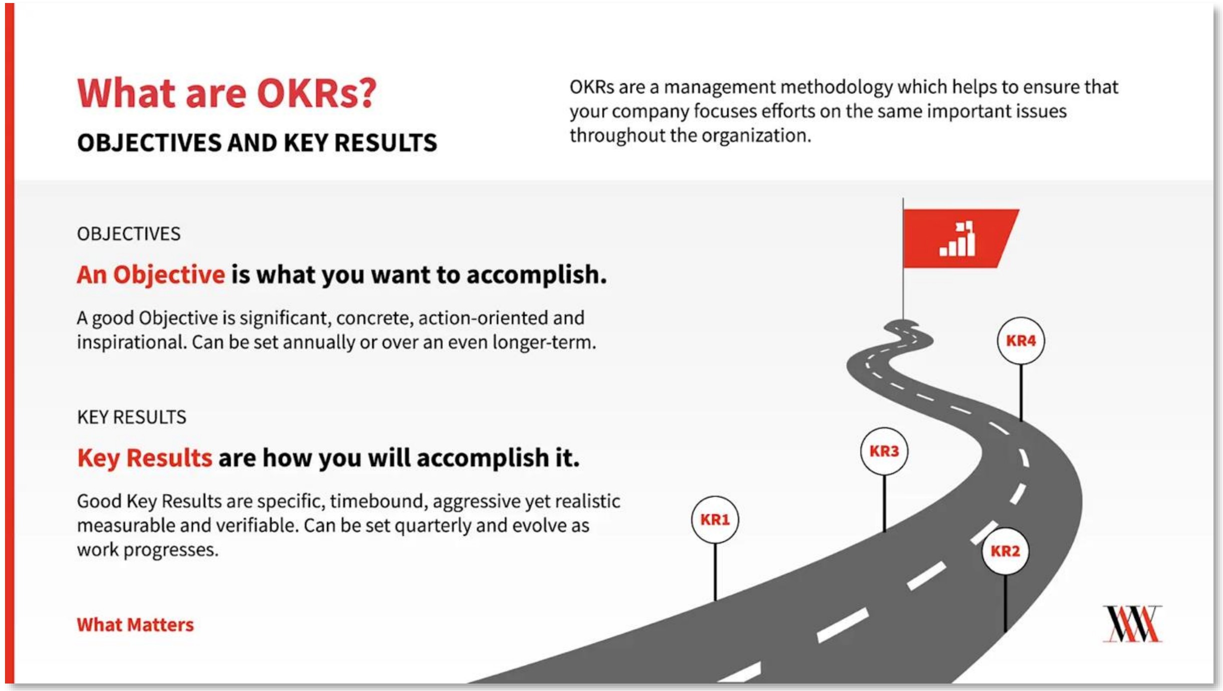
Overcoming the Estimation Obsession

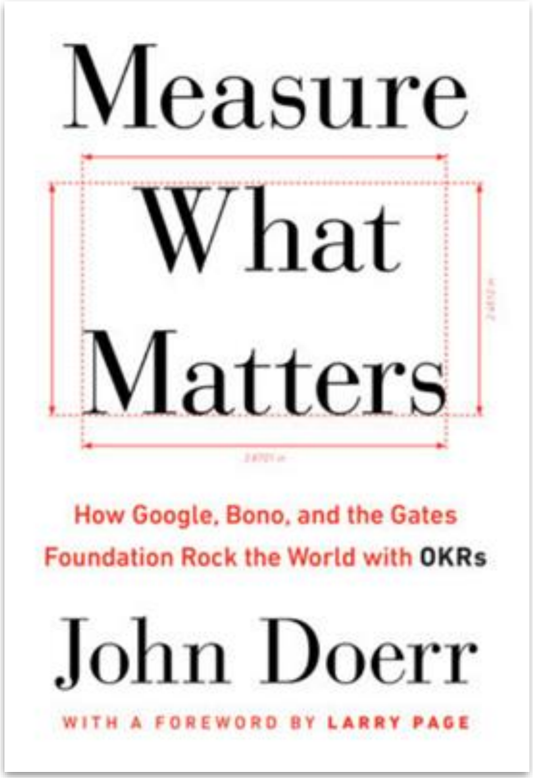
Ron Jeffries's essay [Estimation is Evil]()



"The key result has to be measurable. But at the end you can look, and without any arguments: **Did I do that or did I not do it?** Yes? No? Simple. No judgments in it."

Andy Grove, Inventor of OKRs



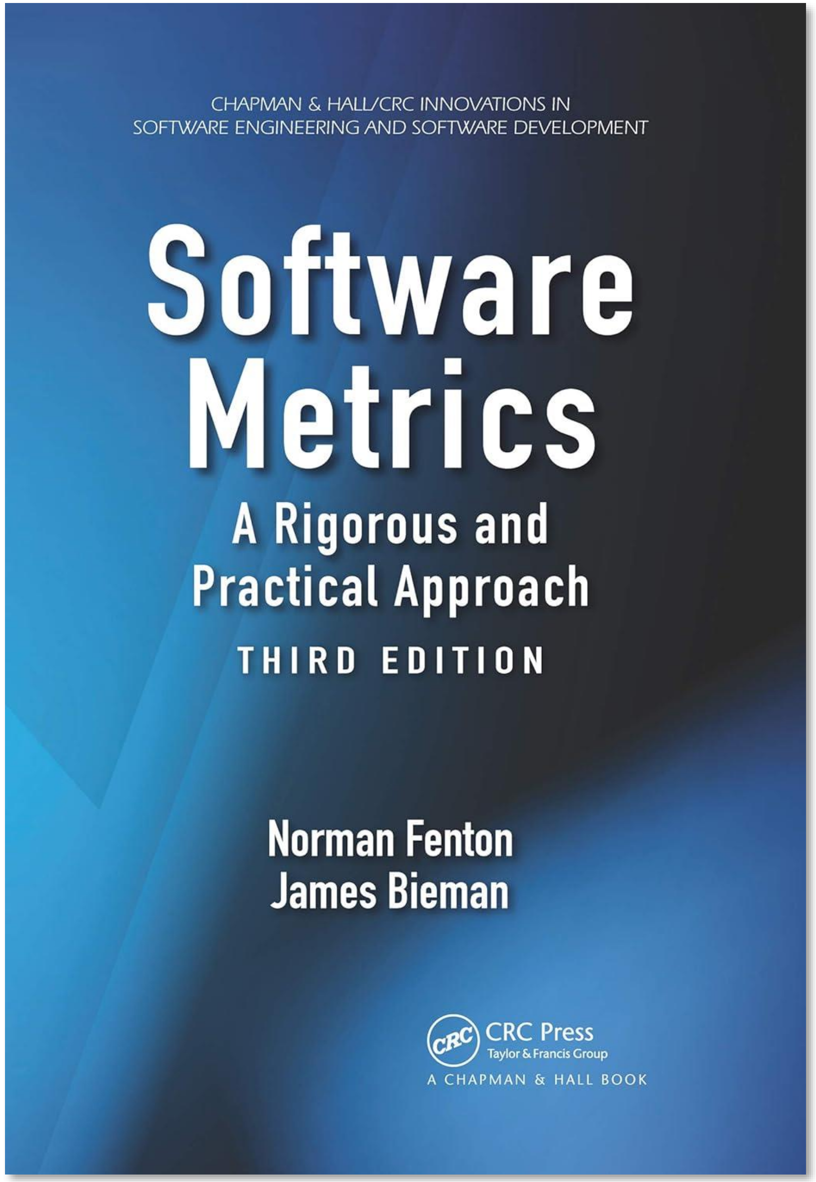


"OKRs have helped lead us to 10x growth, many times over."

Larry Page, CEO of Google

“Every measurement action must be motivated by a particular goal or need that is clearly defined and easily understandable.”

Software Metrics: A Rigorous and Practical Approach
N.Fenton, J.Bieman



Goal-Based Frameworks

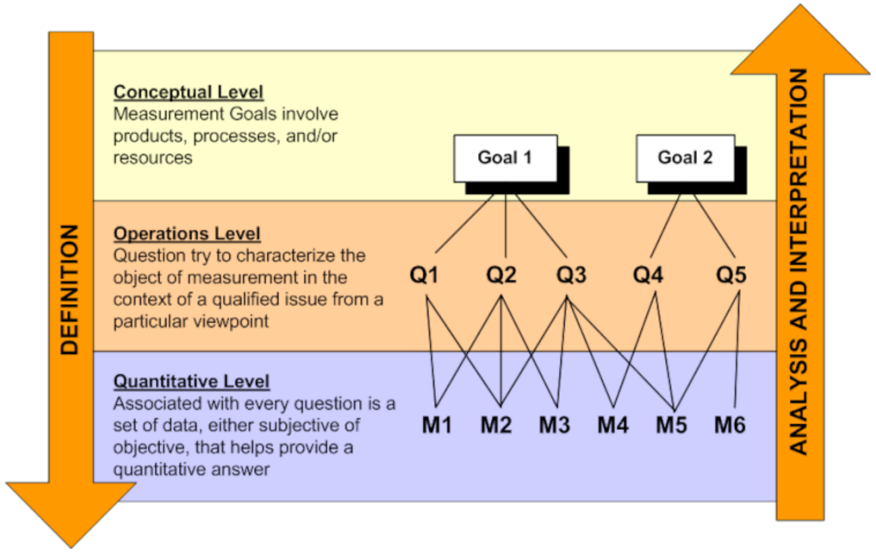
- Objectives and Key Results (OKRs)
- **Goal-Question-Metric (GQM)**
- Assurance Cases
- …

The GQM Framework

Goal: What do you want to achieve?

Questions: What do you need to answer to know whether your goal is met?

Metrics: What measurements do you need to answer each question?



THE GOAL QUESTION METRIC APPROACH

Victor R. Basili[1] Gianluigi Caldiera[1] H. Dieter Rombach[2]

(1) Institute for Advanced Computer Studies
Department of Computer Science
University Of Maryland
College Park, Maryland

(2) FB Informatik
Universität Kaiserslautern
Kaiserslautern, Germany

GQM: Defining Goals

P: **Purpose** (improve, evaluate, monitor, ...)

I: **Issue** (reliability, usability, effectiveness, ...)

O: **Object** (final product, component, process, activity)

V: **Viewpoint** (any stakeholder)

Goal:

Evaluate the **effectiveness** of the **organization's coding standard** from the **team**'s perspective

Questions:

- How comprehensible are the coding standards?
- What is the impact of coding standards on the efficiency and productivity of the team?

Metrics:

- Survey results measuring team members' understanding
- Number of revisions required to achieve standard compliance
- Code size:
  - # of lines of code,
  - # of classes,
  - # of functions

Participation Activity

- Apply the **Goal-Question-Metric** framework to explore various aspects of AV software
- Define one goal, two questions, and at least one metric per question
- Share with the class!

Example

Goal: **Ensure energy efficiency and sustainability** from the point of view of the **organization and environmental analysts**

Q1: What is the energy consumption under different driving conditions?

Metrics: Miles per kWh
{city, highway, mixed}
{winter, summer, fall}

Q2: How efficient is the battery management system?

Metrics: Battery life
number of charge cycles
peak battery temperature

Measurement for Decision Making

- Fund project?
- More testing?
- Fast enough? Secure enough? Safe enough?
- Code quality sufficient?
- Which feature to focus on?
- Should we refactor the code? Rewrite?
- Developer bonus?
- Time and cost estimation? Predictions reliable?

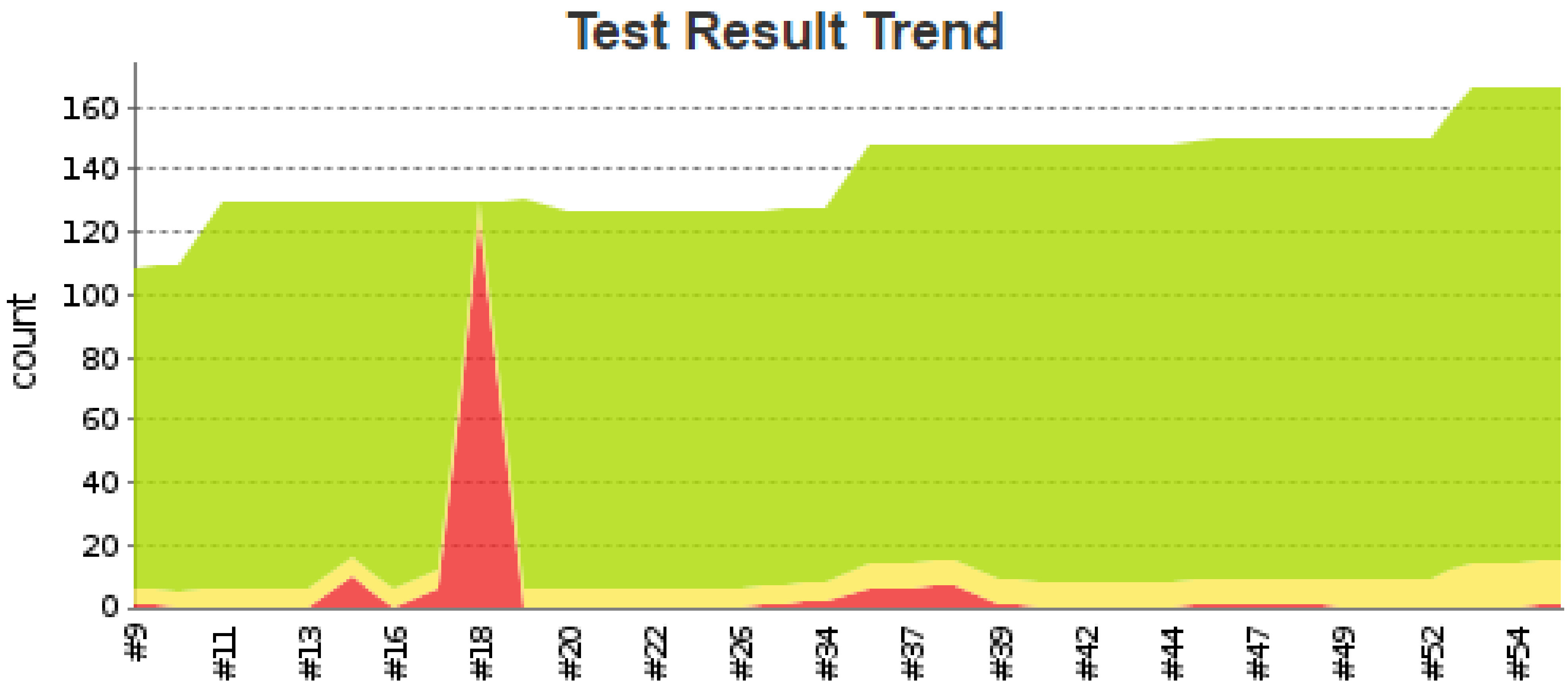
Analysis and Interpretation

Challenge: How do we actually interpret the measurements from our metrics?

Calibrate thresholds based on feedback

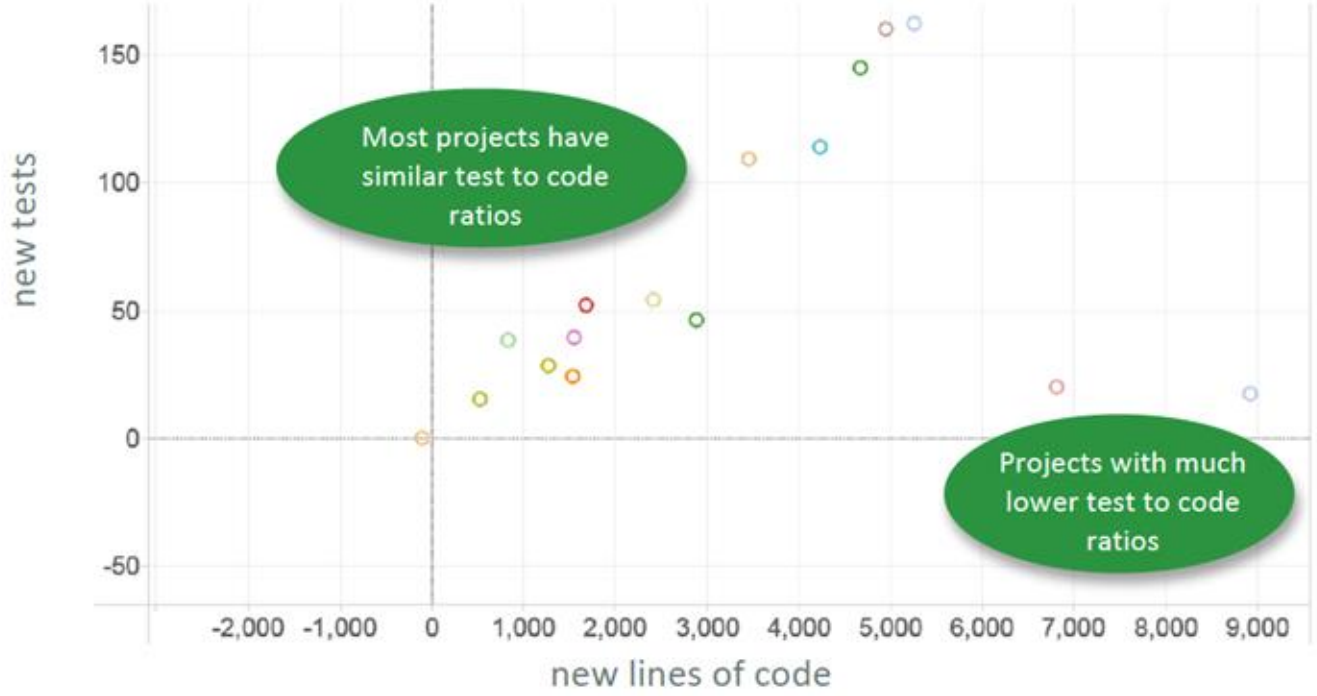
- **Problem:** ”I’ll know it when I see it”
  - ”That function is way too complex!”
- **Solution:** Use existing code to obtain a set of thresholds
  - pick exemplars (clear pass / fail / borderline)
  - present to team, track agreement, and derive thresholds
    - E.g., cyclomatic complexity shouldn’t be higher than 10
  - periodically recalibrate over time (e.g., technology changes)

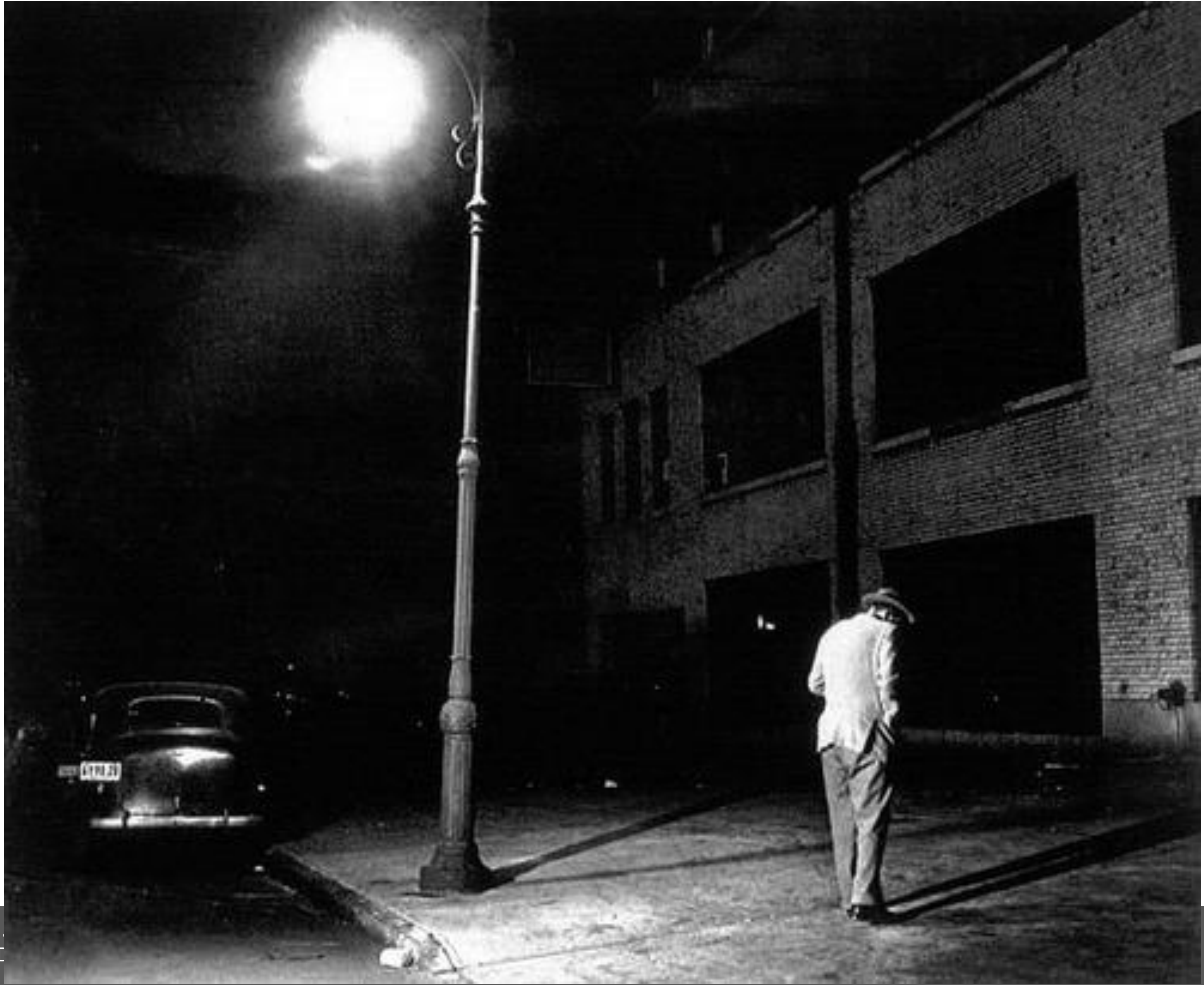
Track metrics over time to assess trends



Benchmark against existing standards

- Monitor **similar** products, projects, modules, teams
- Refer to **external standards** if possible
  - e.g., ISO 26262
- Record typical values for metrics of interest
- Investigate deviations

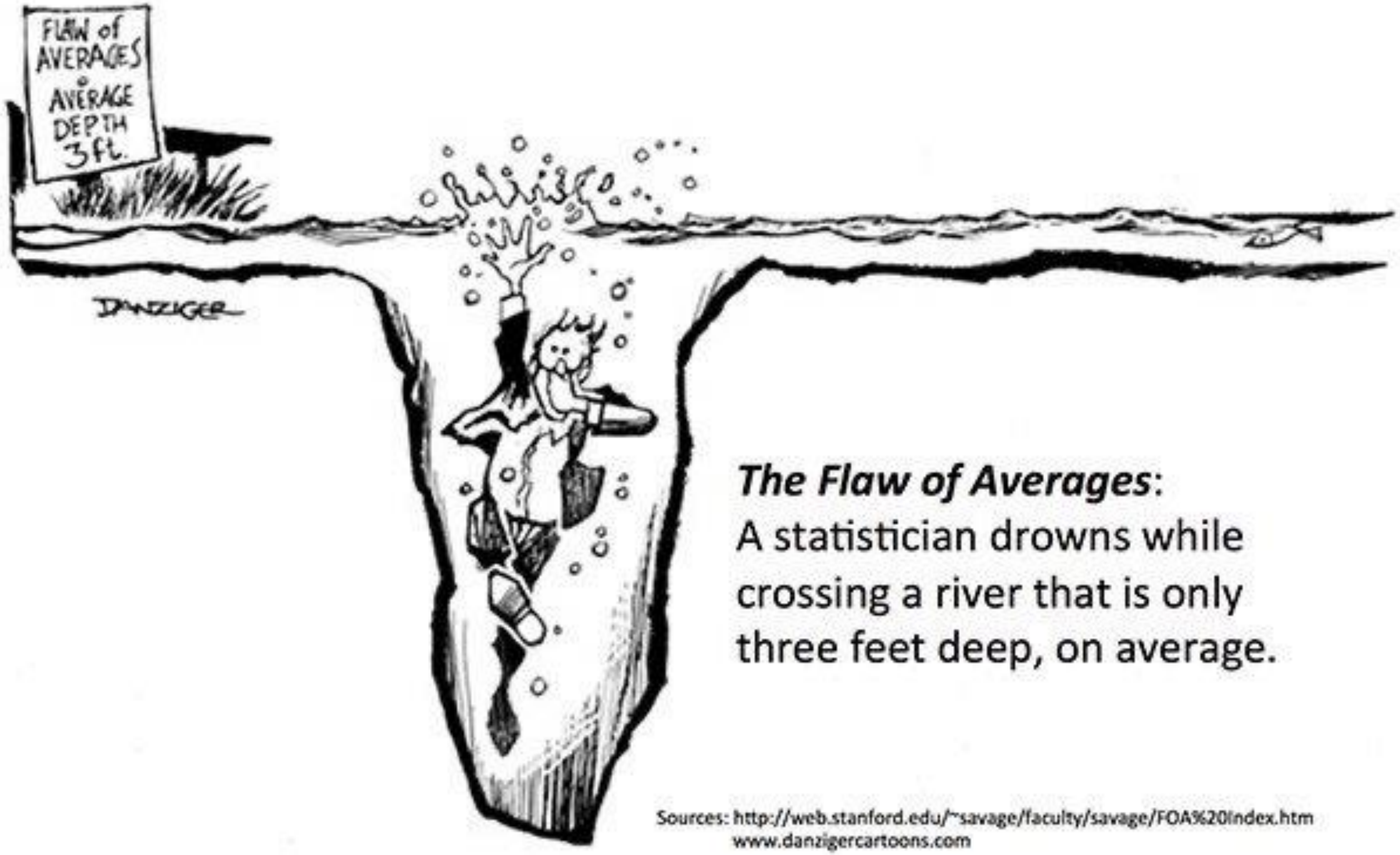




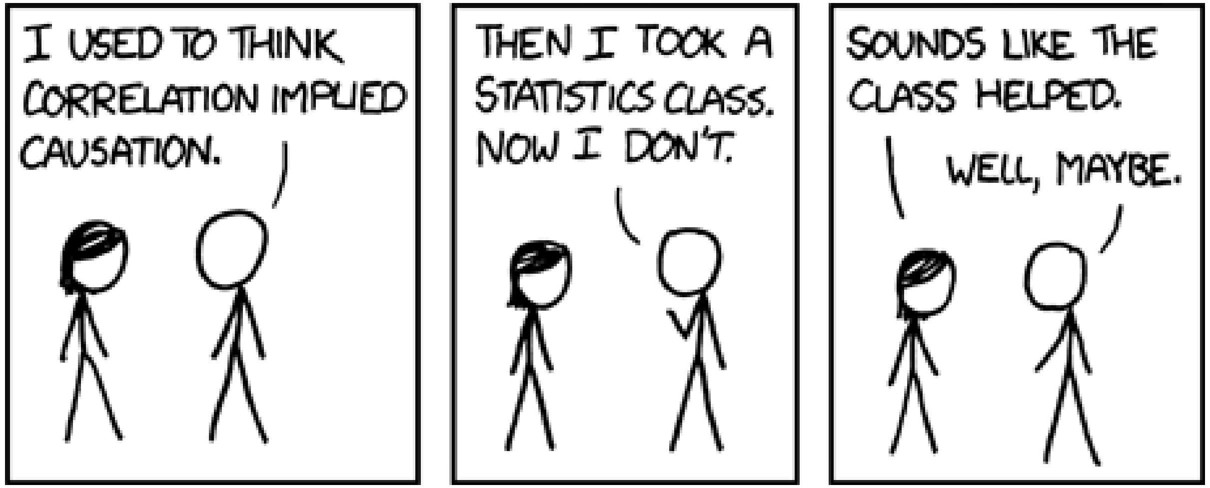
The Streetlight Effect

- A known **observational bias**
- People tend to look for something only where it's easiest to do so
  - If you drop your keys at night, you'll tend to look for it under streetlights

Bad Statistics: What could possibly go wrong?

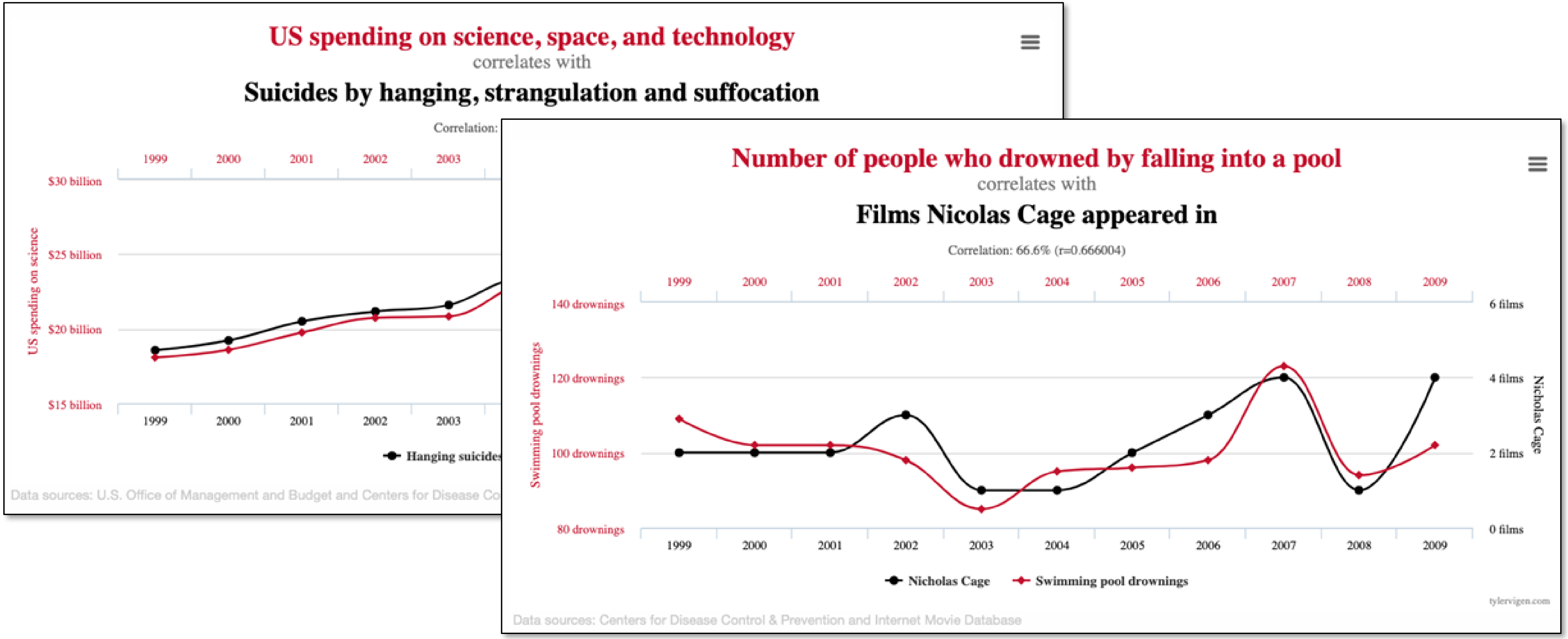


Making Inferences

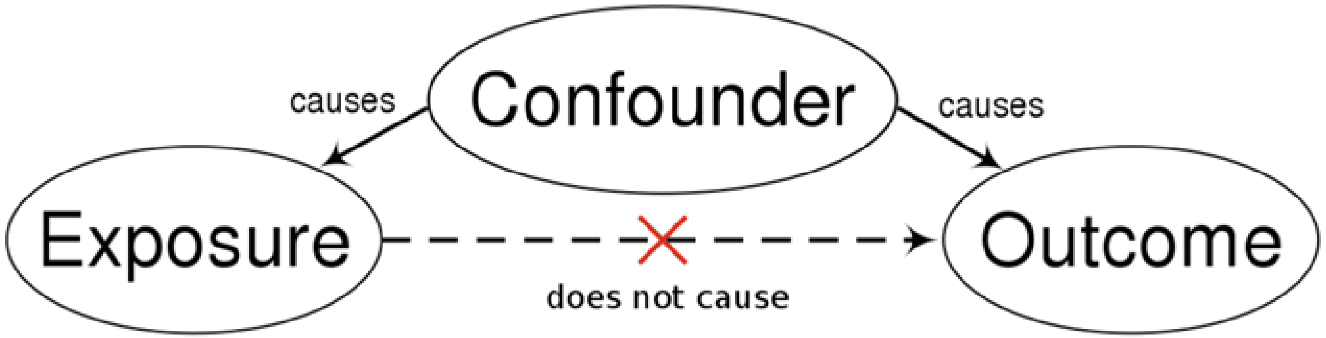


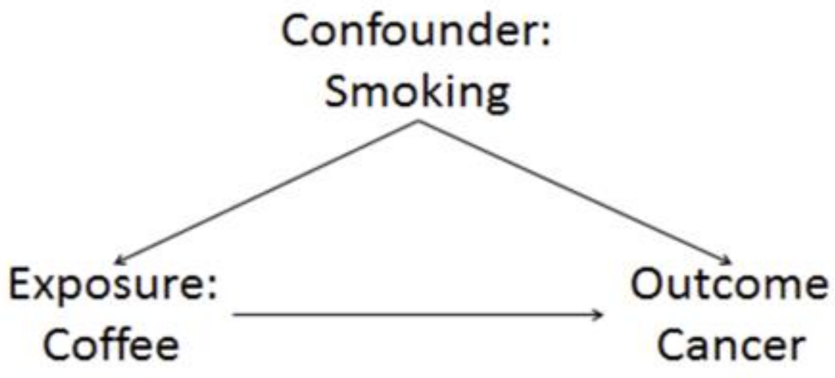
- To infer causation:
  - Provide a theory (from domain knowledge, independent of data)
  - Show correlation
  - Demonstrate ability to predict new cases (replicate/validate)

Spurious Correlations



Confounding Variables





- If you look only at the coffee consumption → cancer relationship, you can get very misleading results
- Smoking is a confounder

RESEARCH-ARTICLE

Coverage is not strongly correlated with test suite effectiveness

Authors: Laura Inozemtseva, Reid Holmes Authors Info & Affiliations

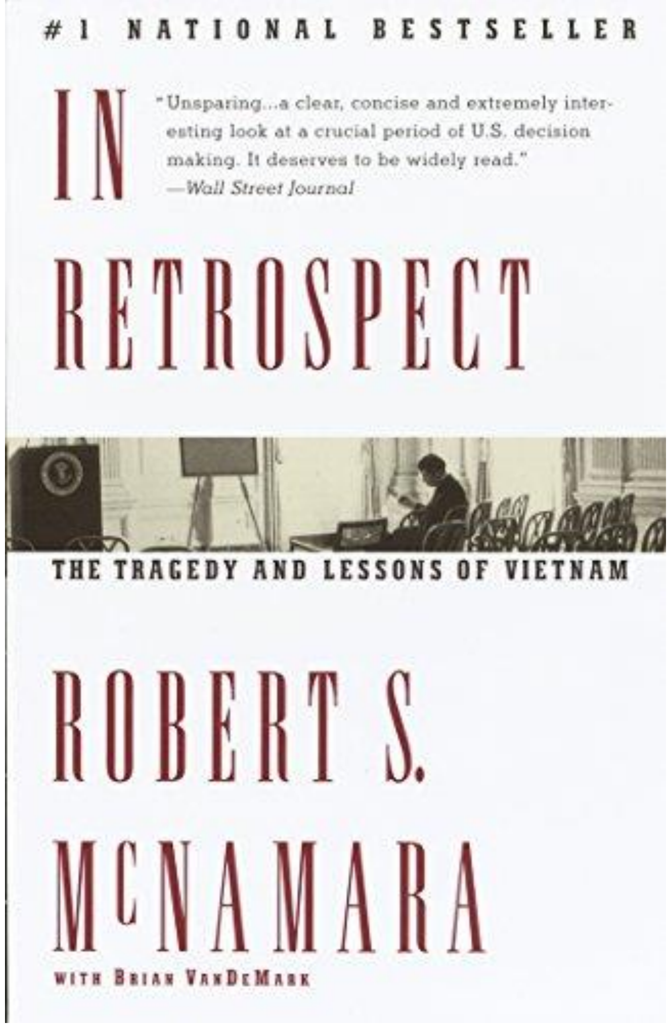
ICSE 2014: Proceedings of the 36th International Conference on Software Engineering • May 2014 • Pages 435–445 • https://doi.org/10.1145/2568225.2568271

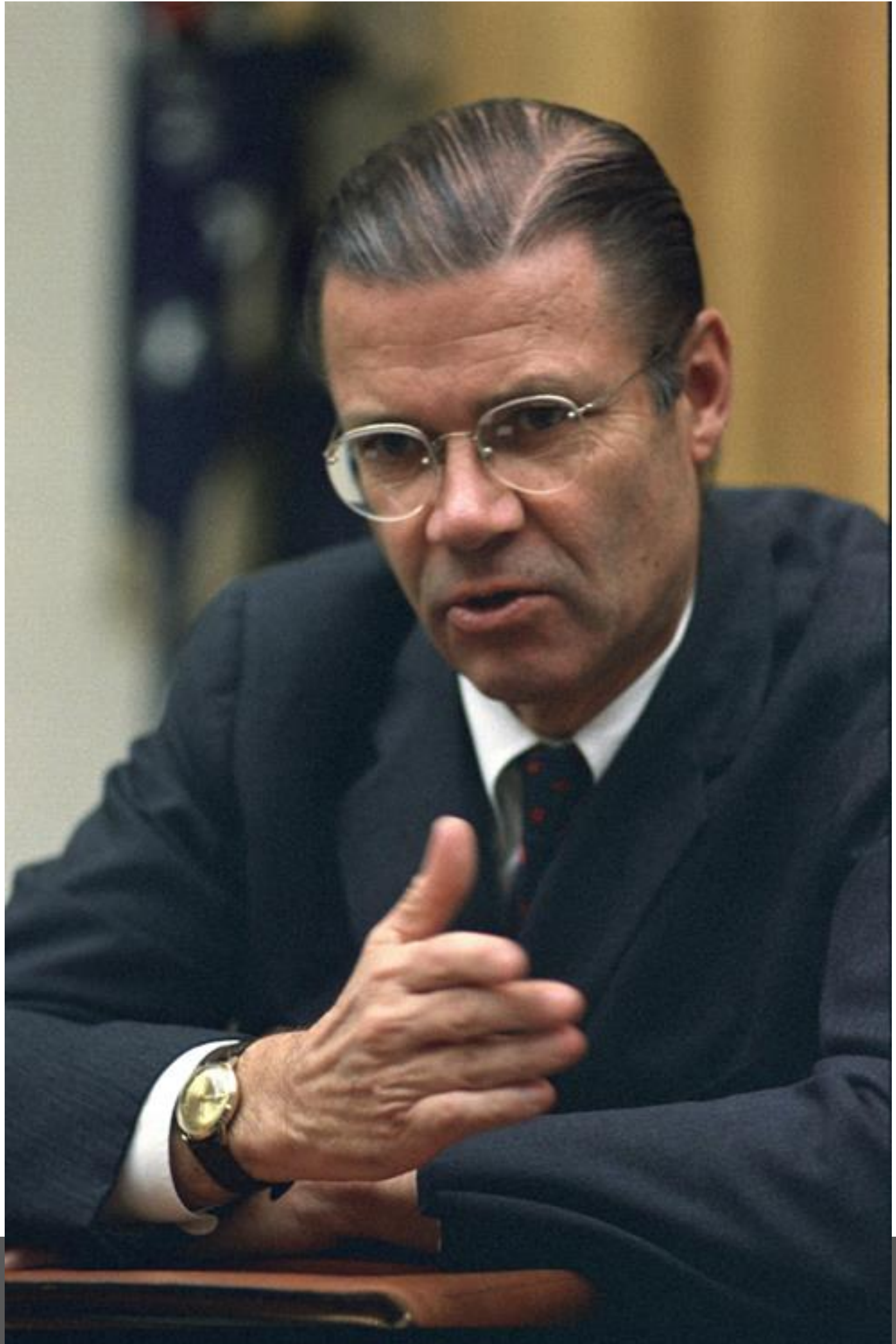
"We found that there is a low to moderate correlation between coverage and effectiveness when the number of test cases in the suite is controlled for."

Most prior studies didn't account for the confounding influence of test suite size

Measurement Reliability

- Extent to which a measurement yields similar results when applied multiple times
- Goal is to reduce uncertainty, increase consistency
- Example: Performance
  - Time, memory usage
  - Cache misses, I/O operations, instruction execution count, etc.
- Law of large numbers
  - Taking multiple measurements to reduce error
  - Trade-off with cost


#1 NATIONAL BESTSELLER
IN RETROSPECT
"Unsparing...a clear, concise and extremely interesting look at a crucial period of U.S. decision making. It deserves to be widely read."
—Wall Street Journal
THE TRAGEDY AND LESSONS OF VIETNAM
ROBERT S. McNAMARA
WITH BRIAN VANDEMARK



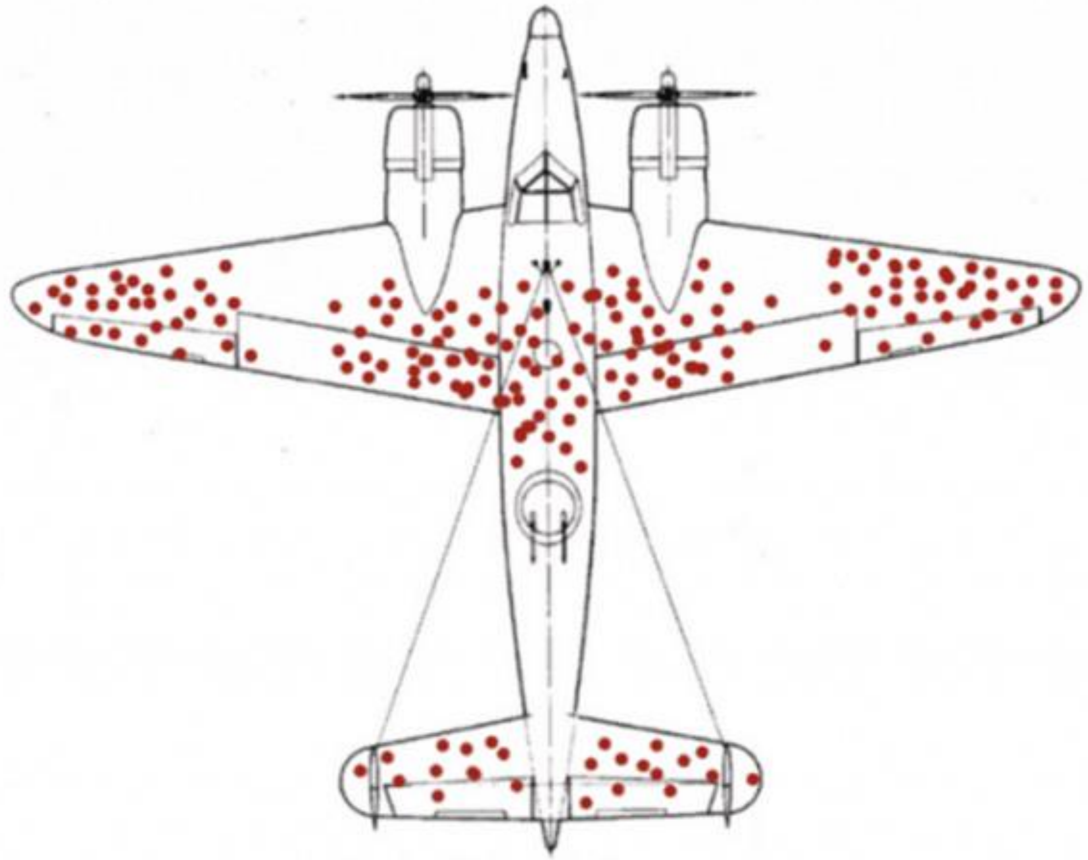

FROM THE DIRECTOR OF "THE THIN BLUE LINE" AND "GATES OF HEAVEN"
"IF THERE IS ONE MOVIE THAT OUGHT TO BE STUDIED BY MILITARY AND CIVILIAN LEADERS AROUND THE WORLD AT THIS TREACHEROUS HISTORICAL MOMENT, IT IS 'THE FOG OF WAR,' ERROL MORRIS'S PORTRAIT OF FORMER UNITED STATES DEFENSE SECRETARY, ROBERT S. MCNAMARA."
"SMART AND COMPLETELY FASCINATING!"
"STUNNING! SUPERBLY MADE!"
"SPLENDID! A MASTERPIECE!"
THE FOG OF WAR

McNamara Fallacy

- Measure whatever can be easily measured
- Disregard that which cannot be measured easily
- Presume that which cannot be measured easily is not important
- Presume that which cannot be measured easily does not exist

https://chronotopeblog.com/2015/04/04/the-mcnamara-fallacy-and-the-problem-with-numbers-in-education

Survivorship Bias



Goodhart's law: "When a measure becomes a target, it ceases to be a good measure."

http://dilbert.com/strips/comic/1995-11-13

The New York Times

The Price of Wells Fargo's Fake Account Scandal Grows by $3 Billion

The bank reached a settlement with federal prosecutors and the Securities and Exchange Commission after abusing customers.

Share full article 199

Wells Fargo used fraud to open up fake accounts and force customers into services

Incentivizing Productivity

- What happens when developer bonuses are based on:
  - Lines of code per day?
  - Amount of documentation written?
  - Low number of reported bugs in their code?
  - Low number of open bugs in their code?
  - High number of fixed bugs?
  - Accuracy of time estimates?
  - Number of tokens used?

What You Need to Know

Metrics are important in Software Engineering

Apply goal-oriented approaches to software metrics

Provide examples of metrics for software qualities and process

Understand limitations and dangers of decisions and incentives based on measurements

Questions to Consider (Projects)

- What properties do we care about and how do we measure them?
- What is being measured? Does it (to what degree) capture the thing you care about? What are its limitations?
- How should it be incorporated into process?
- What are potentially negative side effects or incentives?