

Measurement

One-Slide Summary

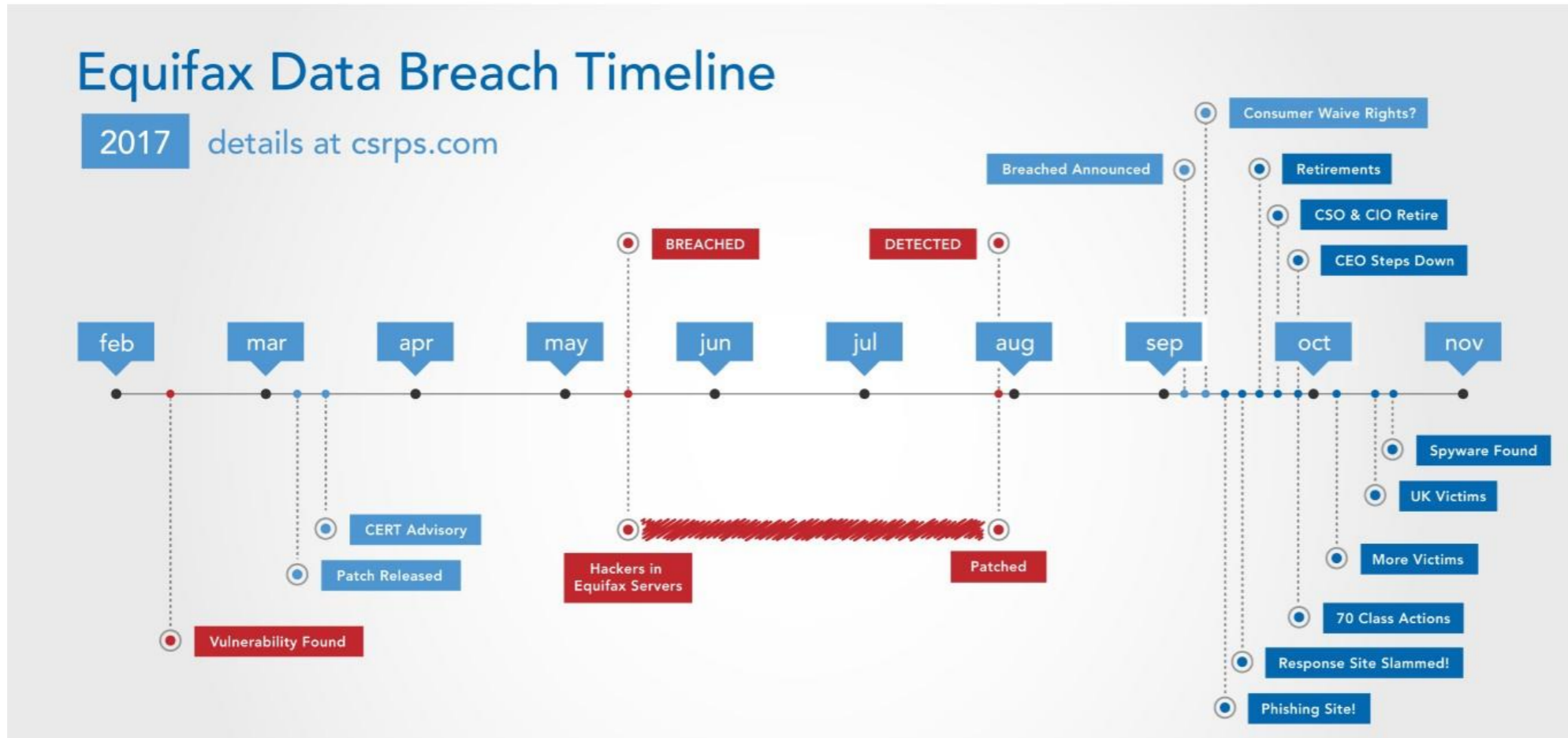
- **Software metrics** are widely used in industry to support decision-making. Metrics are often inadequately supported and thus lack **validity**. They should be used **carefully**.
- **Measurement** is a fundamental activity but is influenced by human **biases**. It is easy to **misinterpret** data or focus on what is easy to **measure**. Metrics can **incentivize** perverse behavior.
- Managers are more concerned with real-world s/w use metrics than individual productivity.

Story So Far

- Using a **software process** correctly could **improve efficiency**. We need **information** to do so (e.g., to identify **risk**) but may lack it because of **uncertainty**.
- If only we could *measure* things to gain information about them ...



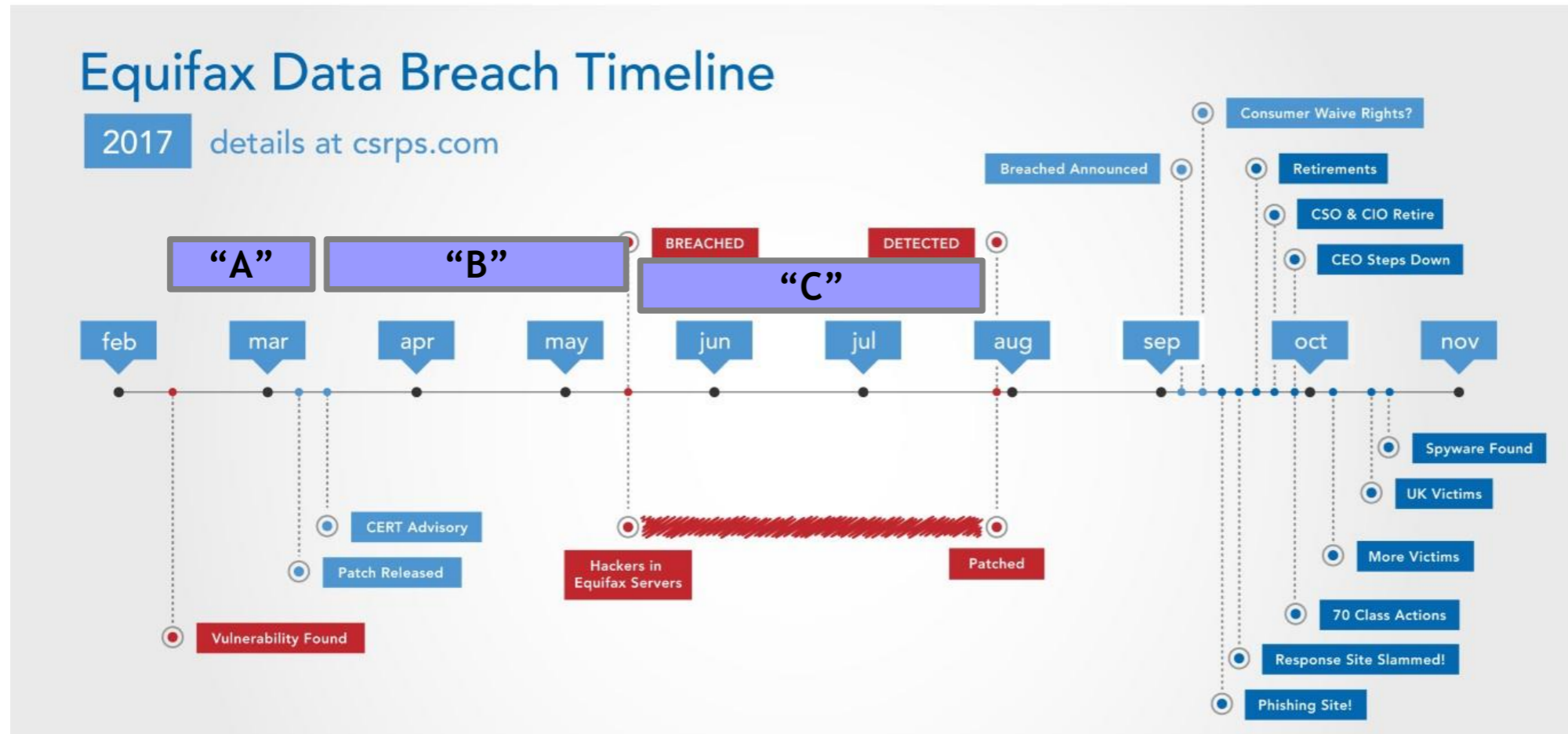
Who Cares About Process Again?



Reminder: “cybercriminals accessed approximately 145.5 million U.S. Equifax consumers' personal data, including their full names, Social Security numbers, birth dates, addresses, and, in some cases, driver license numbers.”

US-CERT: US Computer Emergency Readiness Team, responsible for analyzing and reducing cyberthreats, etc.

Consider Time Ranges: A vs. B+C



Reminder: “cybercriminals accessed approximately 145.5 million U.S. Equifax consumers' personal data, including their full names, Social Security numbers, birth dates, addresses, and, in some cases, driver license numbers.”

Who Cares About Process Again?

Equifax Data Breach Timeline
2017 details at [csrps.com](https://www.csrps.com)

The diagram shows a horizontal timeline with several stages. Three purple boxes labeled "A", "B", and "C" are positioned above the main timeline. "A" and "B" are above the "BREACHED" and "DETECTED" markers respectively. "C" is above the "Breached Announced" marker. To the right of the main timeline, there are four blue boxes: "Consumer Waive Rights?", "Retirements", "CSO & CIO Retire", and "CEO Steps Down", each connected to a point on the timeline by a vertical dotted line.

 [REDACTED]
about an hour ago

When I die, I would like the people I did group projects with to lower me in to my grave so they can let me down one last time.

Unlike · Comment · Share

You, [REDACTED] and 41 others like this.

Outline

- Case Study – Maintainability Index
 - LOC, Halstead Volume, Cyclomatic Complexity
- Measurement
 - Difficulty, Validity
 - Correlation, Confounds
 - Streetlight Effect, McNamara Fallacy
 - Incentives and Warnings
 - Begel and Zimmermann Survey

Maintainability Index

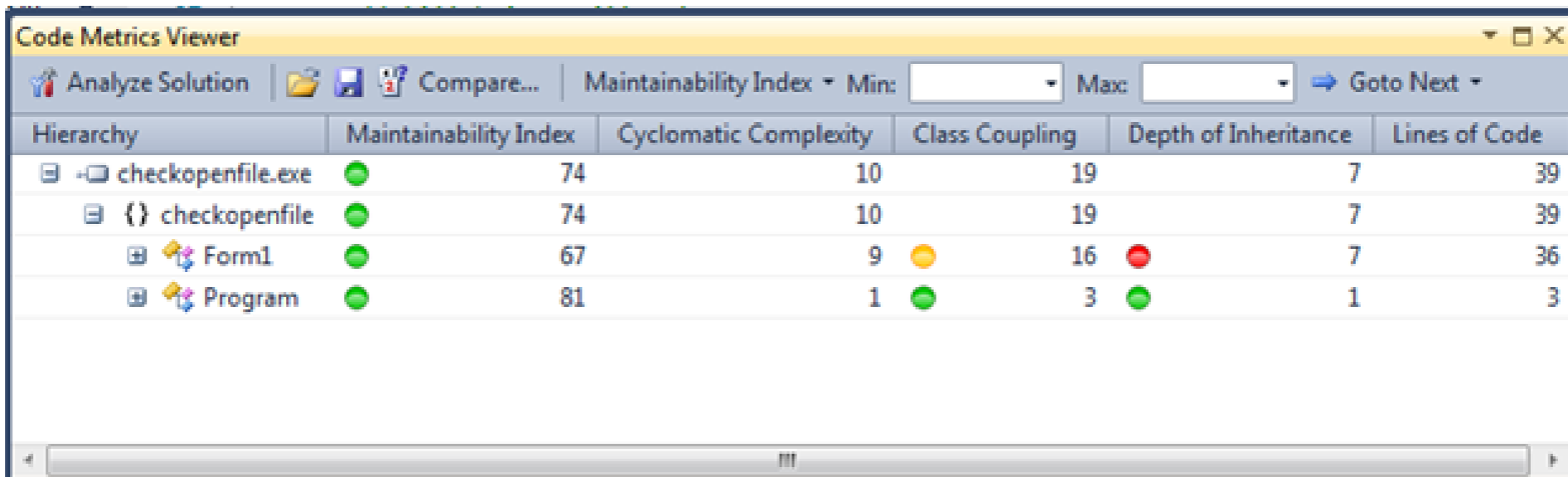
- In Visual Studio since 2007

“**Maintainability Index** calculates an index value between 0 and 100 that represents the relative ease of maintaining the code. A **high value means better maintainability**. Color coded ratings can be used to quickly identify trouble spots in your code. A **green rating is between 20 and 100** and indicates that the code has good maintainability. A **yellow rating is between 10 and 19** and indicates that the code is moderately maintainable. A **red rating is a rating between 0 and 9** and indicates low maintainability.”

Hierarchy	Maintainability Index	Cyclomatic Complexity	Class Coupling	Depth of Inheritance	Lines of Code
[-] -\ checkopenfile.exe	74	10	19	7	39
[-] {} checkopenfile	74	10	19	7	39
[-] Form1	67	9	16	7	36
[-] Program	81	1	3	1	3

Maintainability Index in a Nutshell

- Index between 0 and 100 representing the relative ease of maintaining the code.
- Higher is better. Color coded by number:
 - Green: between 20 and 100
 - Yellow: between 10 and 19
 - Red: between 0 and 9



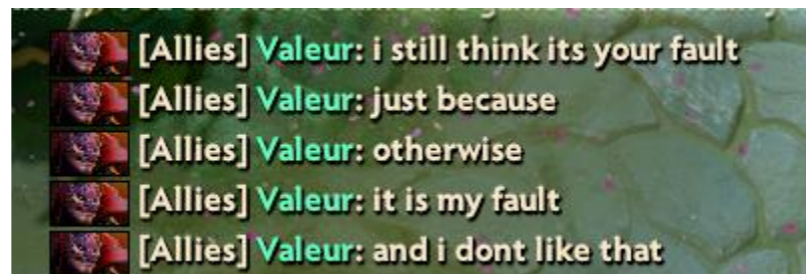
The screenshot shows the 'Code Metrics Viewer' window with a table of code metrics. The table has columns for Hierarchy, Maintainability Index, Cyclomatic Complexity, Class Coupling, Depth of Inheritance, and Lines of Code. The data is as follows:

Hierarchy	Maintainability Index	Cyclomatic Complexity	Class Coupling	Depth of Inheritance	Lines of Code
checkopenfile.exe	74 (Green)	10	19	7	39
checkopenfile	74 (Green)	10	19	7	39
Form1	67 (Green)	9	16 (Yellow)	7 (Red)	36
Program	81 (Green)	1	3 (Green)	1	3

Design Rationale

- "We noticed that as code tended toward 0 it was clearly hard to maintain code and the difference between code at 0 and some negative value was not useful."
- "The desire was that if the index showed red then we would be saying with a high degree of confidence that there was an issue with the code."

[<https://blogs.msdn.microsoft.com/codeanalysis/2007/11/20/maintainability-index-range-and-meaning/>]



The Magic Formula

Maintainability Index =

$$\max(0, (171 - 5.2 * \log(\text{Halstead Volume}) - 0.23 * (\text{Cyclomatic Complexity}) - 16.2 * \log(\text{Lines of Code})) * 100 / 171)$$

WHO WOULD WIN?

Lines of Code

Superficially easy to measure

```
wc -l file1 file2
```

450	Expression Evaluator
2.000	Sudoku, Functional Graph Library
40,000	OpenVPN
80-100,000	Berkeley DB, SQLite
150-300,000	Apache, HyperSQL, Busybox, Emacs, Vim, ArgoUML
500-800,000	gimp, glibc, mplayer, php, SVN
1,600,000	gcc
6,000,000	Linux, FreeBSD
45,000,000	Windows XP

a computer program with millions of lines of code



one CURLYBOY with no friend



Lines of Code: Normalized

- Common Practices:
 - Ignore comments and empty lines
 - Ignore lines with fewer than 2 characters
 - Pretty Print source code first

```
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");  
}
```



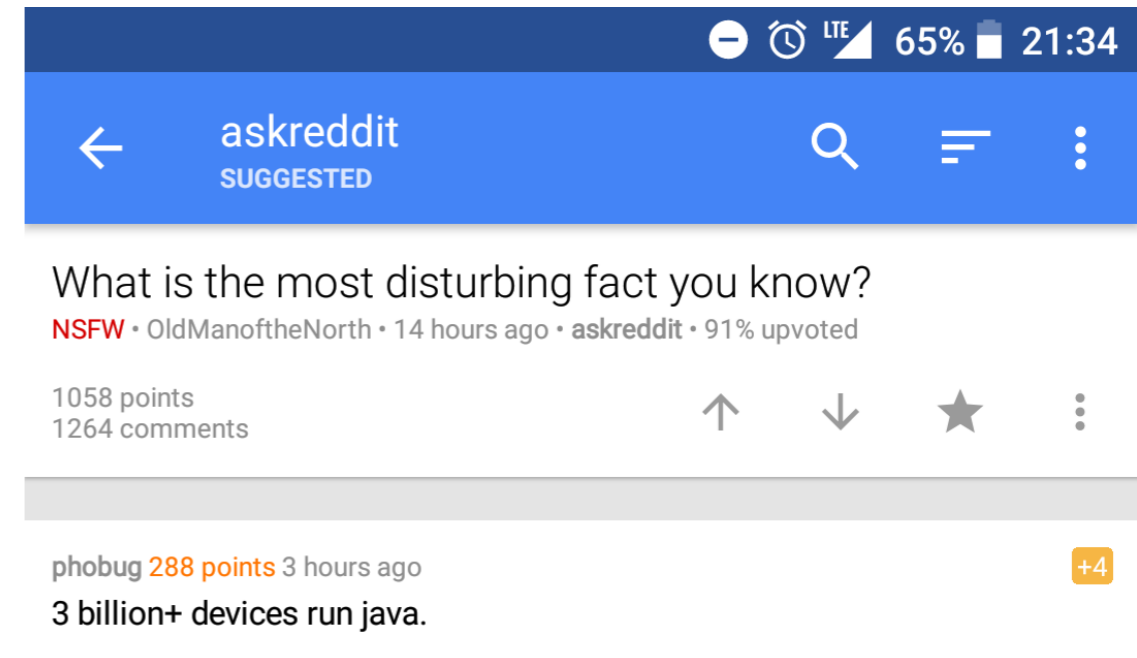
Languages: Normalized

- “Programmers working with **high-level languages** achieve better **productivity** and **quality** than those working with **lower-level languages**. Languages such as C++, Java, Smalltalk, and Visual Basic have been credited with improving productivity, reliability, and comprehensibility by **factors of 5 to 15** over low-level languages such as assembly and C (Brooks 1987, Jones 1998, Boehm 2000).”
- [Steve McConnell. *Code Complete: A Practical Handbook of Software Construction*, Second Edition. Microsoft.]

Languages: Normalized

- “... typical **ratios of source statements** in several high-level languages to the equivalent code in C. A **higher** ratio means that **each line of code** in the language listed accomplishes **more** than does each line of code in C.”

• C	1.0
• Fortran	2.0
• C++	2.5
• Java	2.5
• Visual Basic	4.5
• Perl	6.0
• Python	6.0
• Smalltalk	6.0



Halstead Volume

- Introduced by Maurice Halstead in 1977
 - “Halstead made the observation that **metrics** of the software should **reflect** the implementation or expression of **algorithms** in different languages, but be **independent** of their execution on a specific **platform**.”
- Halstead Volume =
number of operators / operands *
 $\log_2(\text{number of distinct operators / operands})$
- Approximates the size of elements and vocabulary

Halstead Example

```
main() {  
    int a, b, c, avg;  
    scanf("%d %d %d", &a, &b, &c);  
    avg = (a + b + c) / 3;  
    printf("avg = %d", avg); }
```

- The 12 unique operators (of 27) are:

`main`, `()`, `{}`, `int`, `scanf`, `&`, `=`, `+`, `/`, `printf`, `,`, `;`

- The 7 unique operands (of 17) are:

`a`, `b`, `c`, `avg`, `"%d %d %d"`, `3`, `"avg = %d"`

Cyclomatic Complexity

- Proposed by McCabe in 1976
- Based on control flow graphs (CFG), it measures **linearly independent paths** through a program
 - ~ “number of decisions”
 - ~ “tests to cover all branches”

(For more info: take a *Compilers* or *PL* class.)

How to know you're a sick programmer

```
152 |         wire = re.findall("\\((.*?)\\)", line)
153 |         if len(wire) > 1:
154 |             undefined_nets.append(wire[1])
~/scripts/comp.py
1 comp.py|119 col 1 C| 901 'program' is too complex (21) [mccabe]
```

[Location List] CodeCheck <comp.py> 1,1-65

```
if (c1) {
    f1 ();
} else {
    f2 ();
}
if (c2) {
    f3 ();
} else {
    f4 ();
}
```

Cyclomatic Complexity

- Based on control flow graphs, it measures **linearly independent paths** through a program
 - ~ “number of decisions”
 - ~ “tests to cover all branches”

(For more info: take a *Compilers* or *PL* class.)

Linearly independent path:

Any path through the program that introduces at least one new edge that is not included in any other linearly independent paths

Cyclomatic Complexity: $M = E - N + 2 * P$

E: # of edges *N*: # of nodes *P*: # of connected components

Connected component: a connected subgraph that is not part of any larger connected subgraph

```
if (c1) {
    f1 ();
} else {
    f2 ();
}
if (c2) {
    f3 ();
} else {
    f4 ();
}
```

Maintainability Index: Origins

- Developers rated a number of HP systems
- Statistical regression analysis to find key factors among 40 candidate metrics

$$171 - 5.2 \cdot \ln(\text{avgHV}) - 0.23 \cdot \text{avgCC}(g) - 16.2 \cdot \ln(\text{avgLOC}) + 50 \cdot \sin(\sqrt{2.4 \cdot \text{perCM}})$$

HV:	Halstead Volume	CC:	Cyclomatic Complexity
LOC:	lines of code	perCM:	% Comment Lines

[Oman and Hagemeister. *Metrics for Assessing a Software System's Maintainability*. ICSM 1992.]

Case Study Thoughts

- Metrics seem attractive, can be easy to compute, and seem to match our intuition
- Parameters can be arbitrary: calibrated from small study, few devs, unclear significance
 - Ex: original 1992 C/Pascal programs may be quite different from modern Java/JS/C# code
- Many of these metrics strongly correlate with size: just measure lines of code?

[cf. <https://avandeursen.com/2014/08/29/think-twice-before-using-the-maintainability-index/>]

Measurement for Decision Making in Software

- **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. [Craner, Bond, “Software Engineering Metrics: What Do They Measure and How Do We Know?”]
- A quantitatively expressed reduction of uncertainty based on one or more observations. [Hubbard, “How to Measure Anything ...”]

Software Quality Metric

- IEEE 1061 says:
- “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**.”

Measurement for Decision Making

- Fund project?
- More testing?
- Fast enough? Secure enough?
 - (“Should Equifax apply this webserver patch?”)
- Code quality sufficient?
- Which feature to focus on?
- Developer bonus?
- Time and cost estimation? Predictions reliable?

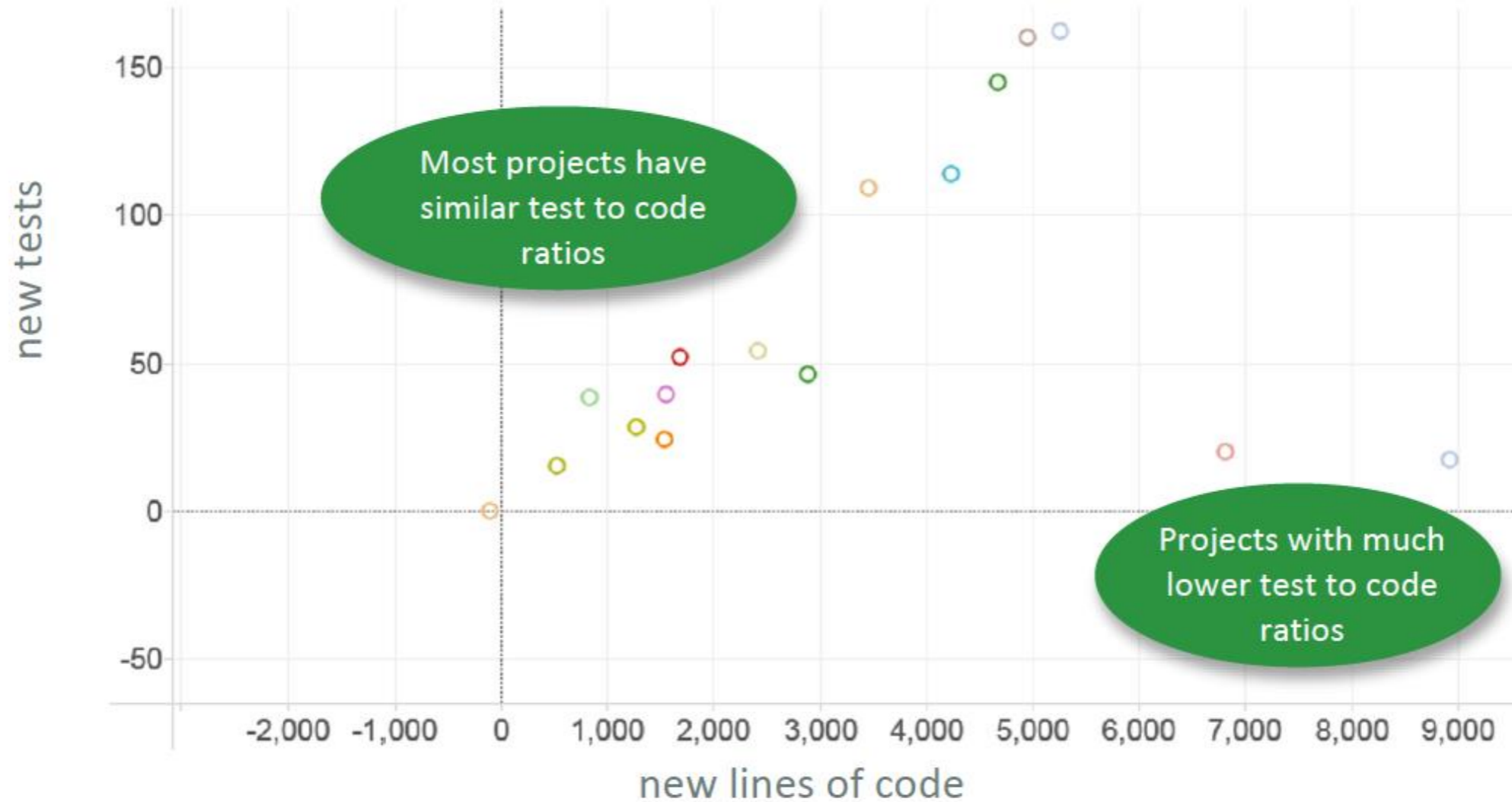
Software Qualities

- Scalability
- Security
- Extensibility
- Documentation
- Performance
- Consistency
- Portability
- Installability
- Maintainability
- Functionality (e.g., data integrity)
- Availability
- Ease of use
- Privacy
- Energy Efficiency

Process Qualities

- On-time release
 - Development speed
 - Meeting efficiency
 - Conformance to processes
 - Time spent on rework
 - Reliability of predictions
 - Fairness in decision making
- *Measure time, costs, actions, resources, and quality of work packages; compare with predictions*
 - *Use information from issue trackers, communication networks, team structures, etc.*
 - ...

Positive Example: Benchmark-Based Metrics

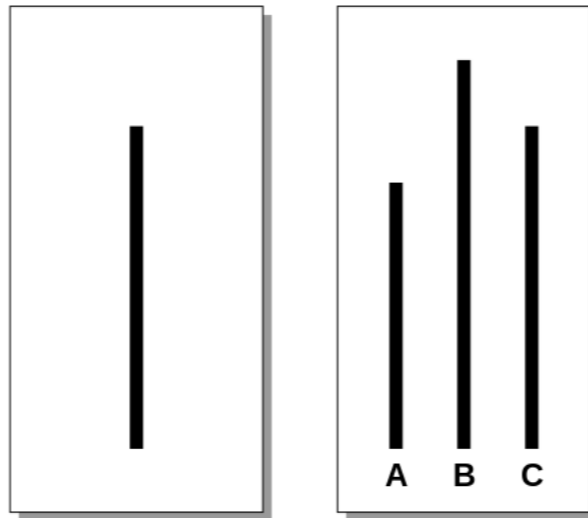


Measurement is Difficult

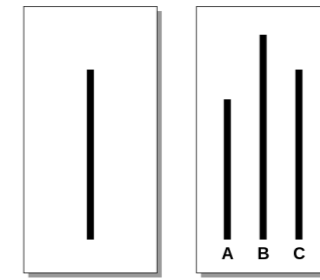


Psychology: “Perception”

You are participating in a perception study with other students. One by one you each say aloud which line in the second card is most like the line in the first card:



Psychology: “Perception”



- When you are alone, your accuracy is 100%
- When 7 of the 8 people ahead of you give the wrong answer, your accuracy drops to 63.2%
 - Overall, 75% of participants gave an [obviously!] incorrect answer at least one time out of twelve
- Most “yielders”: “I suspected about the middle – but tried to put it out of my mind”
- 12/50 had “distortion of perception”: expressed belief that the given answer was correct; were unaware that all were wrong

Psychology: Social Influence

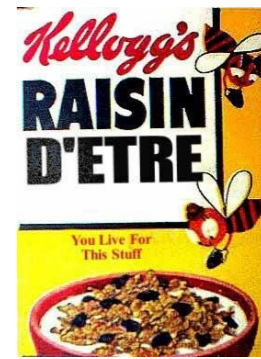
- This study is **Asch's Conformity Experiment**
- Individual differences were large, independence was frequent (e.g., 95% of subjects defied the majority at least once)
 - Still, 75% yielded to a falsehood at least once
- **Implications for SE:** What if you and your boss disagree on a measurement “before your eyes”? Also: dangers of groupthink.

[Asch, S.E. (1951). Effects of group pressure on the modification and distortion of judgments. In H. Guetzkow (Ed.), *Groups, leadership and men* (pp. 177–190).]

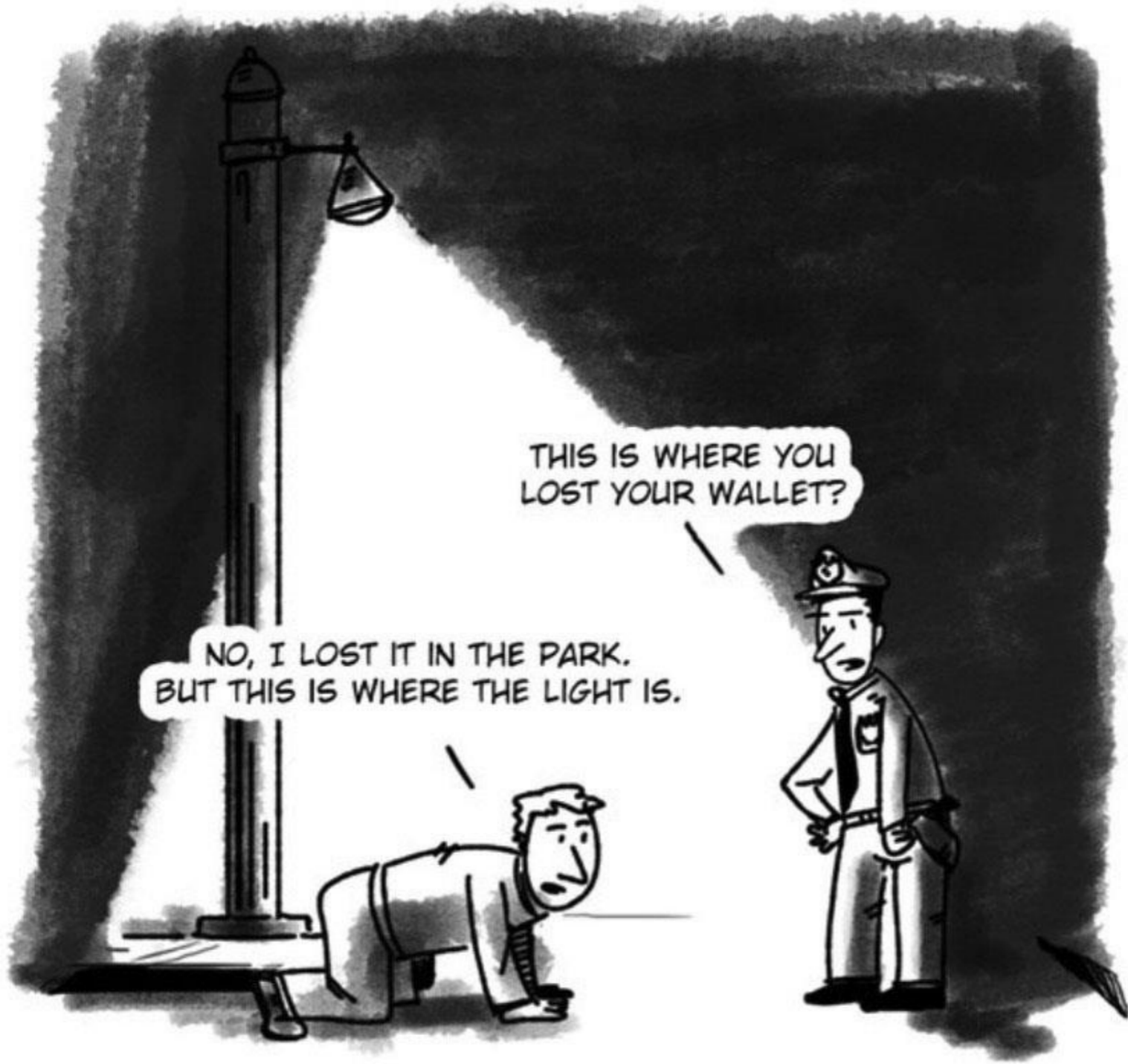
Validity

- **Construct Validity**: Are we measuring what we intended to measure?
- **Predictive Validity**: The extent to which the measurement can be used to explain some other characteristic of the entity being measured
- **External Validity**: Concerns the generalization of the findings to contexts and environments, other than the one studied

Everything is Measurable



- If **X** is something we care about, then **X**, by definition, 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.

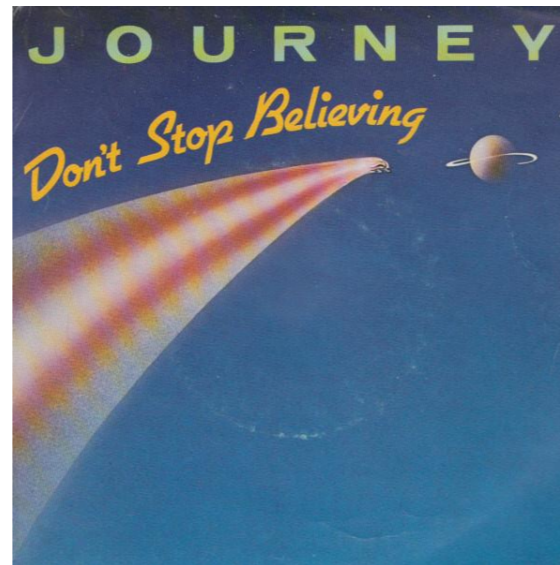


THIS IS WHERE YOU
LOST YOUR WALLET?

NO, I LOST IT IN THE PARK.
BUT THIS IS WHERE THE LIGHT IS.

Streetlight Effect

- The **streetlight effect** is a type of **observational bias** that occurs when people are searching for something and look only where it is easiest
- Despite this, don't lose faith in measurement: just work to avoid the bias

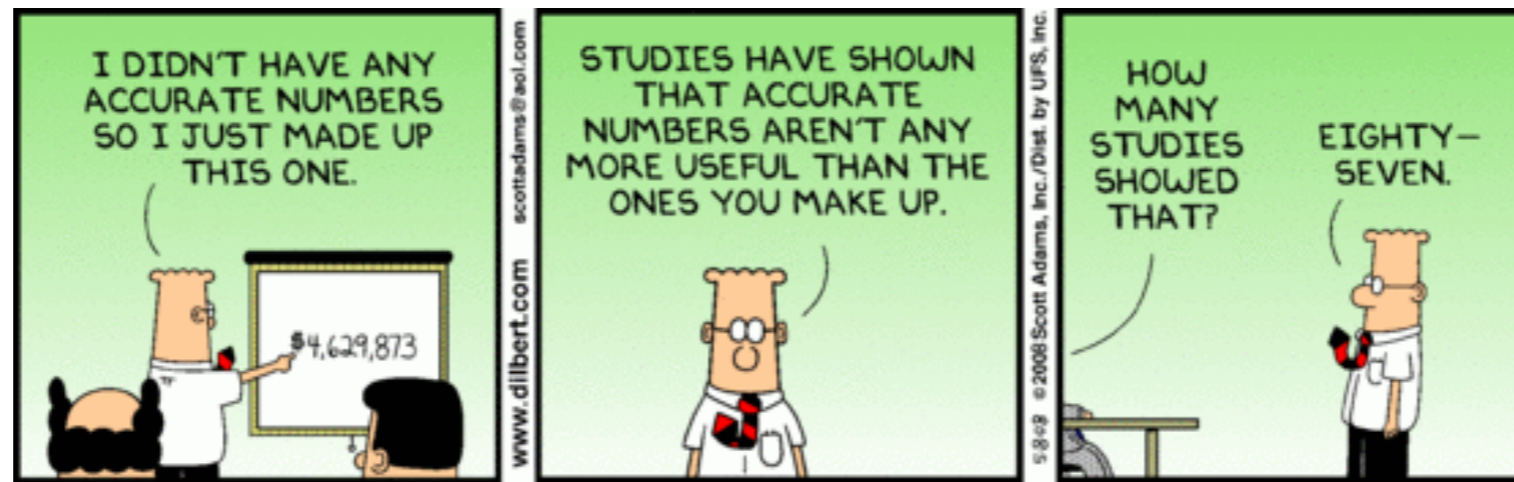


Dangers When Using Metrics

- Bad statistics: A basic misunderstanding of measurement theory and what is being measured.
- Bad decisions: The incorrect use of measurement data, leading to unintended side effects.
- Bad incentives: Disregard for the human factors, or how the cultural change of taking measurements will affect people.

Lies, damned lies, and ...

- A case study for your consideration:
- In 1995, the UK Committee on Safety of Medicines issued the following warning: "third-generation oral contraceptive pills increased the risk of potentially life-threatening blood clots in the legs or lungs **twofold** -- that is, by 100 percent"



... statistics

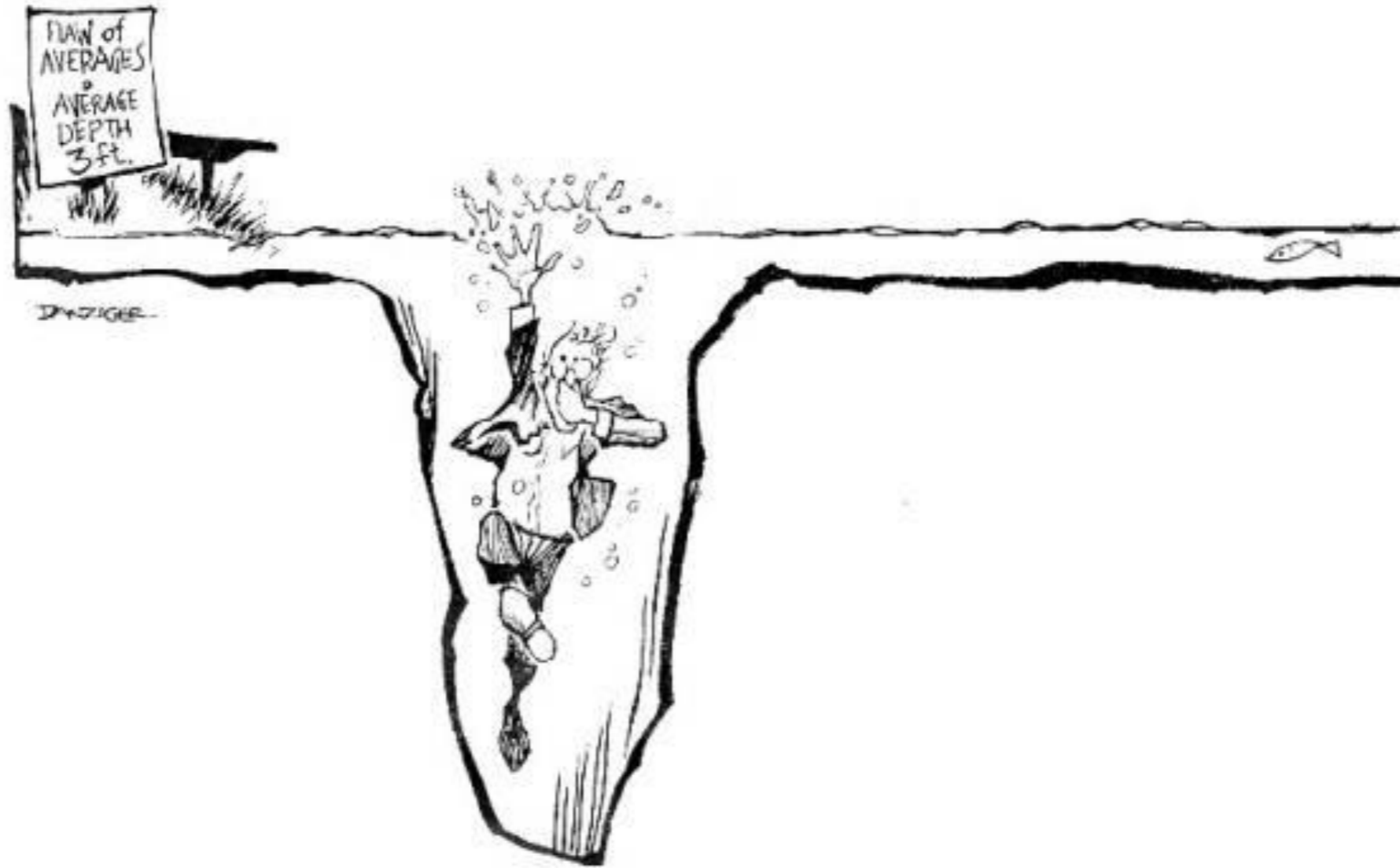
- “...of every 7,000 women who took the earlier, second-generation oral contraceptive pills, about one had a thrombosis; this number increased to two among women who took third-generation pills...”
- “...The **absolute risk increase was only one in 7,000**, whereas the relative increase (among women who developed blood clots) was indeed 100 percent.”

False Positive Paradox

- The **false positive paradox** is a statistical result where false positive tests are more probable than true positive tests, occurring when the overall population has a low incidence of a condition and the incidence rate is lower than the false positive rate.
- The probability of *actually being* infected after one is told that one is infected is only 29% ($20/20 + 49$) for a test that otherwise appears to be "95% accurate":

Number of people	Infected	Uninfected	Total
Test positive	20 (true positive)	49 (false positive)	69
Test negative	0 (false negative)	931 (true negative)	931
Total	20	980	1000

Understanding Data

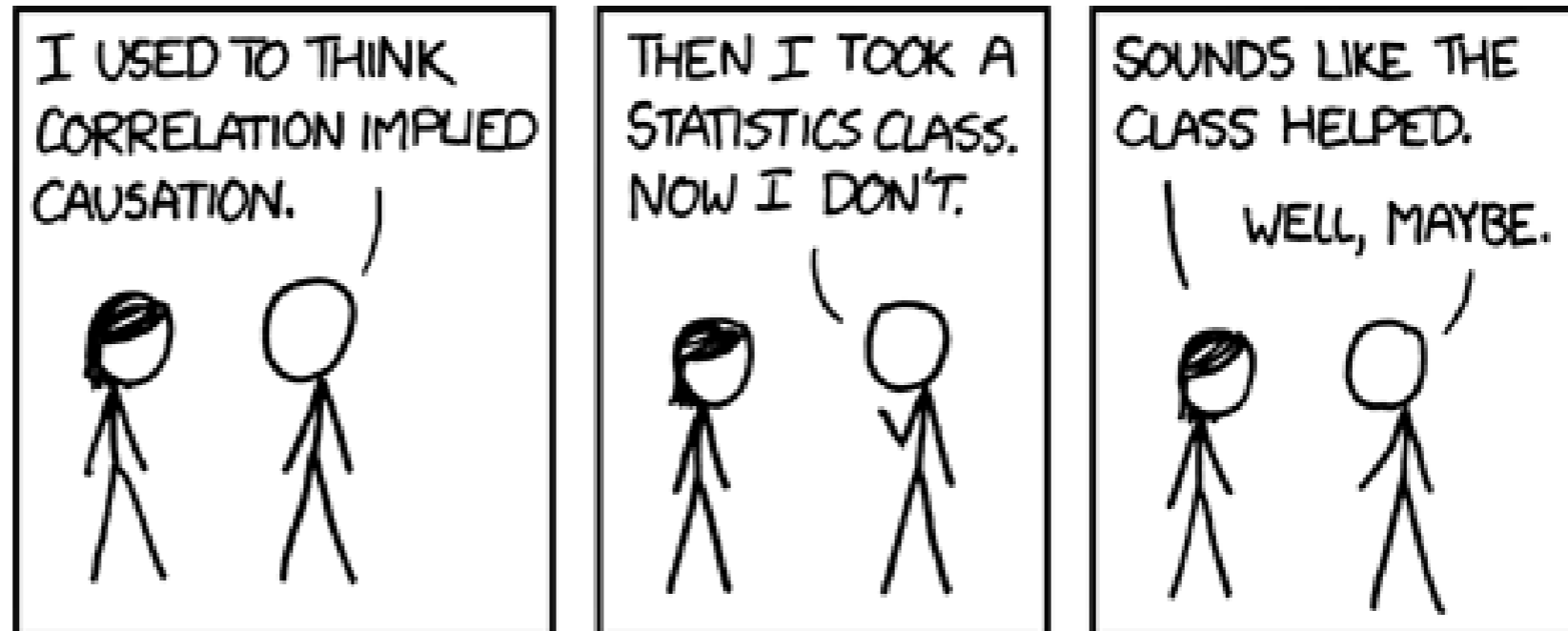


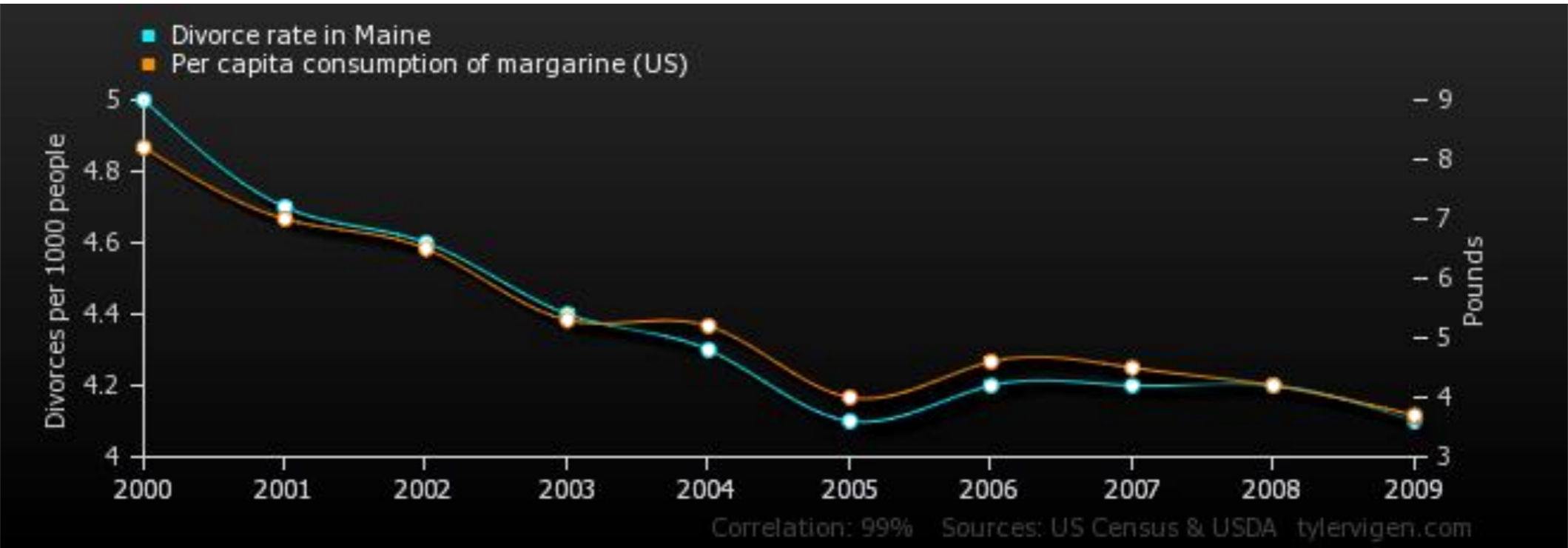
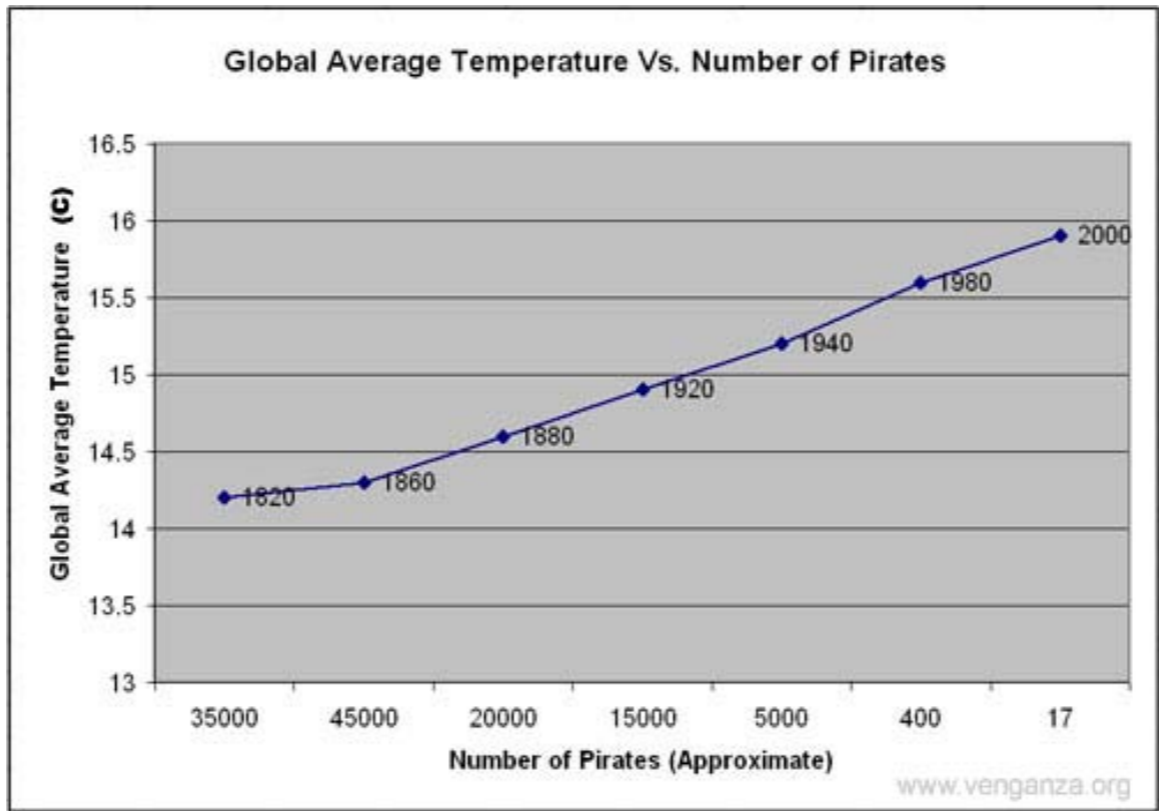
Measurement Scales

- **Scale**: the type of data being measured
- The scale dictates which analyses are legitimate or meaningful
- Common options:
 - **Nominal**: categories
 - **Ordinal**: order, but no magnitude (e.g., ranks)
 - Interval: order, magnitude, but no true zero (e.g., temperature)
 - Ratio: Order, magnitude, and true zero (e.g., height)

To Argue Causation

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





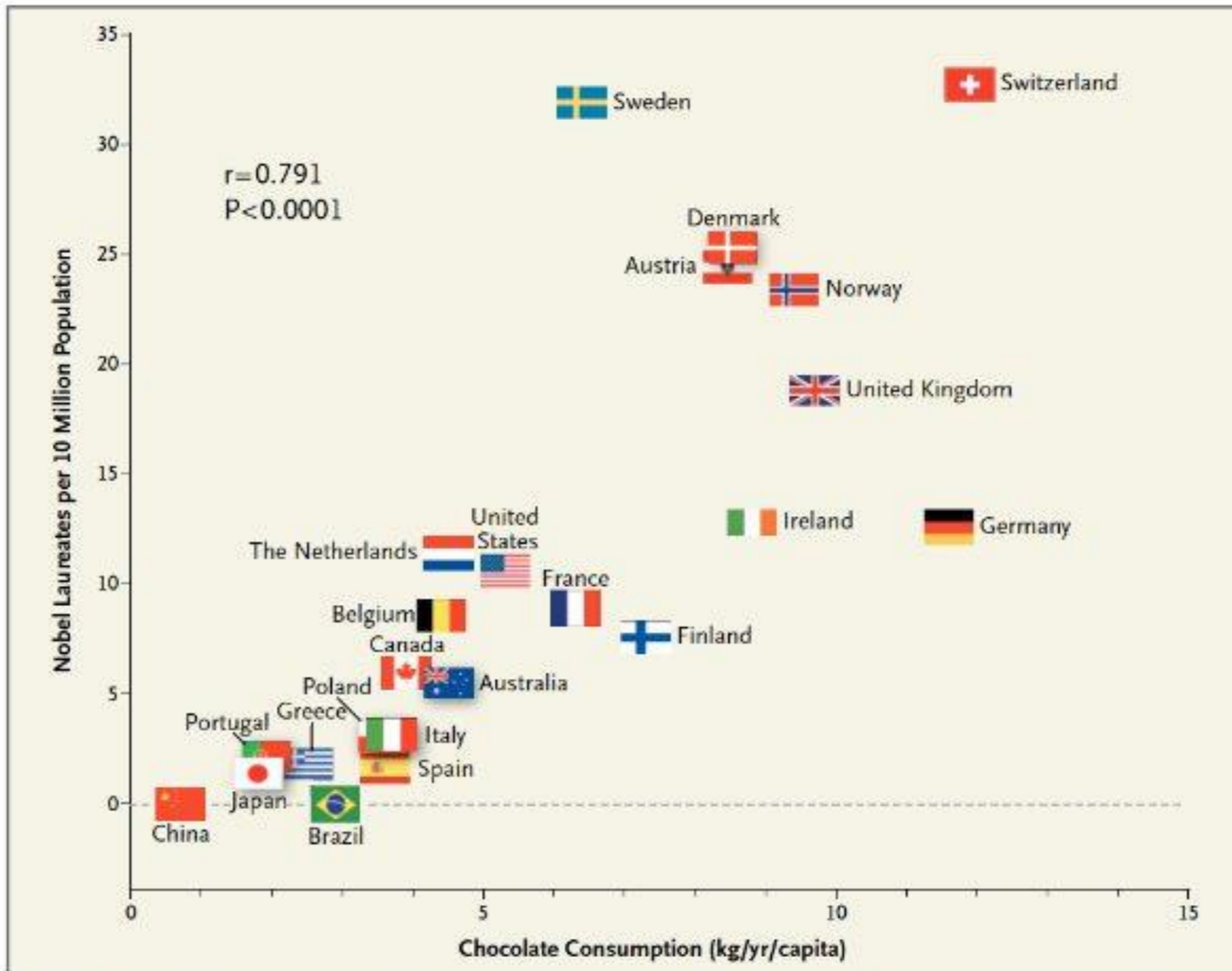
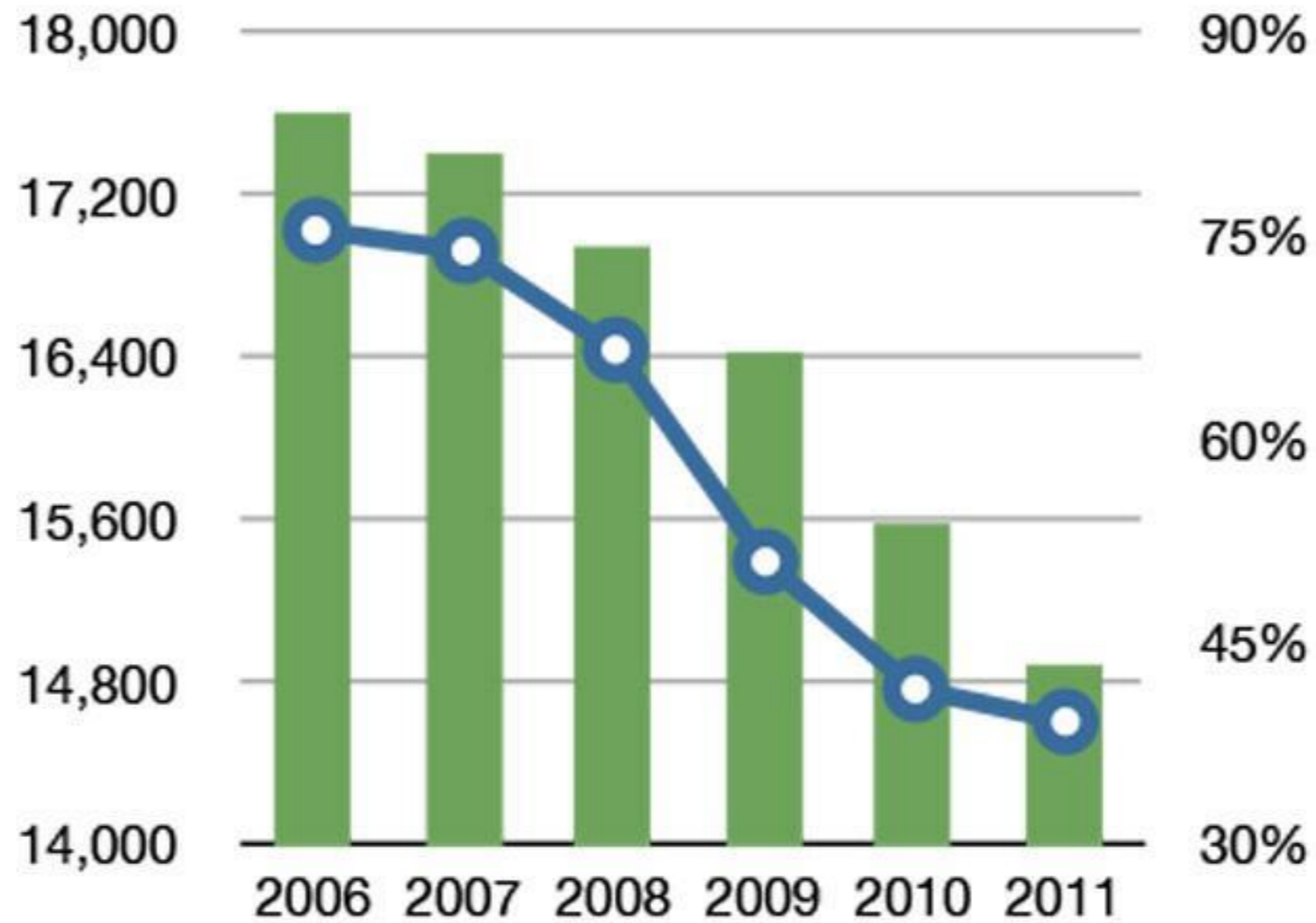


Figure 1. Correlation between Countries' Annual Per Capita Chocolate Consumption and the Number of Nobel Laureates per 10 Million Population.

Internet Explorer vs Murder Rate



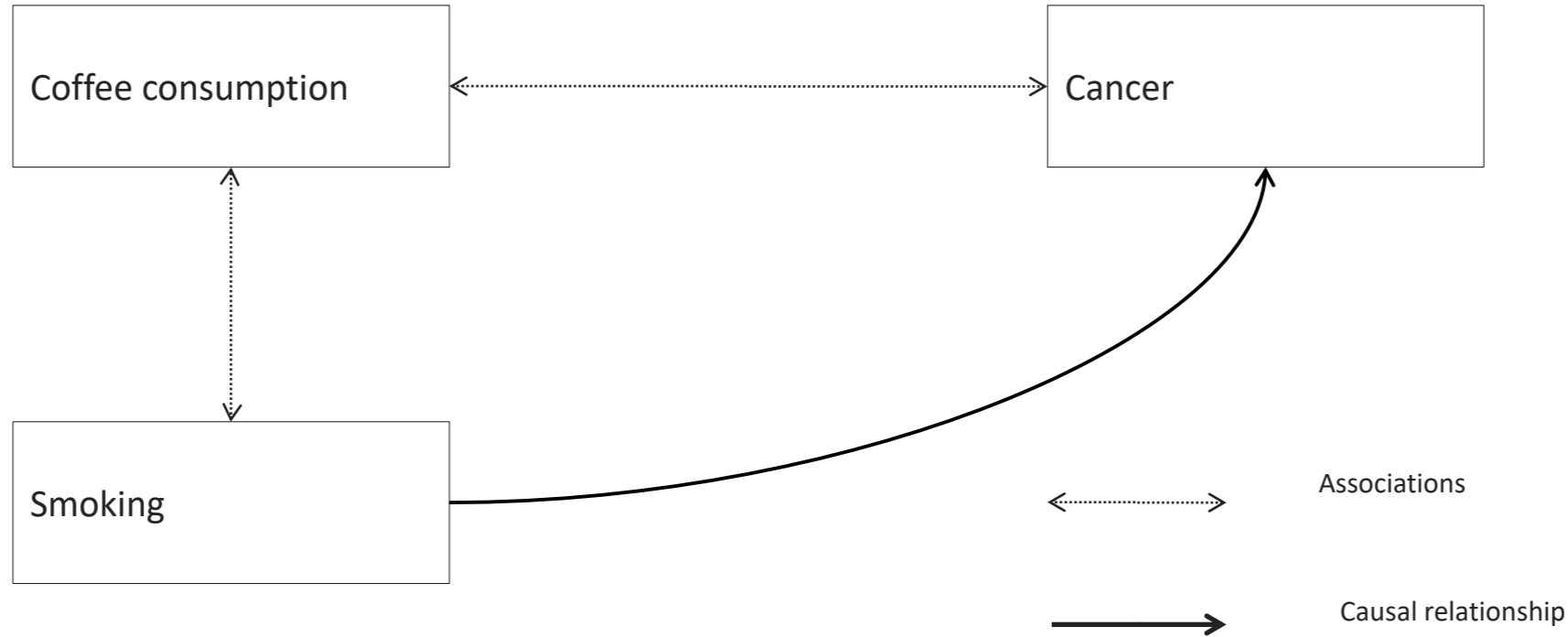
○ Murders in US ■ Internet Explorer Market Share

Confounding Variables



If we examine coffee consumption → cancer

Confounding Variables



If we examine coffee consumption → cancer, we end up with misleading results

Smoking is a **confounding** variable

Confounds in Software Analysis

- Earlier we considered that some metrics (e.g., Halstead, Cyclomatic) might be just “size” cleverly disguised
- In a study of twenty-four commonly-used object-oriented metrics, **only four were actually useful** in predicting the quality of a software module when the effect of the module size was accounted for

[El Emam et al. *The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics*. IEEE Transactions on Software Engineering 2001.]

McNamara Fallacy

- The **McNamara fallacy** (or **quantitative fallacy**), involves making a decision based **solely on quantitative** observations (or metrics) and ignoring all others.
 - The reason given is often that these other observations cannot be proven.
- “There seems to be a general misunderstanding to the effect that a mathematical model cannot be undertaken until every constant and functional relationship is known to high accuracy. ... to omit such variables is equivalent to saying that they have zero effect... Probably the only value known to be wrong ...” - J. W. Forrester

McNamara on Vietnam

The McNamara fallacy originates from the Vietnam War, in which **enemy body counts** were taken to be a precise and objective **measure of success**. **War was reduced to a mathematical model:** *by increasing enemy deaths and minimizing one's own, victory was assured.* ... The fallacy refers to McNamara's belief as to what led the United States to defeat in the Vietnam War—specifically, his quantification of success in the war (e.g. in terms of enemy body count), ignoring other variables.

Thought Experiment: Defect Metrics

- Defect Density = known bugs / line of code
- System Spoilage = time to fix post-release defects / total system development time
- Considerations:
 - Post-release vs. pre-release
 - What counts as a defect? Severity? Relevance?
 - What size metric is used?
- Little reference data is available (typically 2-10 defects / 1,000 lines of code)

Measurement Strategies

- Automated measures on code repositories
- Use or collect process data
- Instrument the program (e.g., in-field crash reports)
- Ask humans: surveys, interviews, controlled experiments, expert judgments
- Statistical analysis of sample

Metrics and Incentives



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 your code
 - Low number of open bugs in your code
 - High number of bugs fixed
 - Accuracy of time estimates

An Example Metric Incentive

- At a “large top-five public research university”, the engineering deans used “**research dollars expended per square foot**” as a ranking and incentive metric for departments.
 - A department with more “RDE/ft²” was doing better and would get more perks from the dean
- How would you arrive at this metric?
- What could go wrong?

Software Metric Warning

- Most software metrics are controversial
 - Usually based on plausibility arguments (not rigorous validation)
 - Cyclomatic Complexity was **repeatedly refuted and is still used**
 - “Similar to the attempt of measuring the intelligence of a person in terms of the weight or circumference of the brain.”

Software Metric Failure

- ROUGE score used to determine quality of machine-generated prose
- Commonly used to evaluate effectiveness of machine learning models that *automatically document code*
- However, developer productivity *did not correlate* with higher ROUGE scores

[Stapleton et. al. A Human Study of Comprehension and Code Summarization, ICPC 2020]

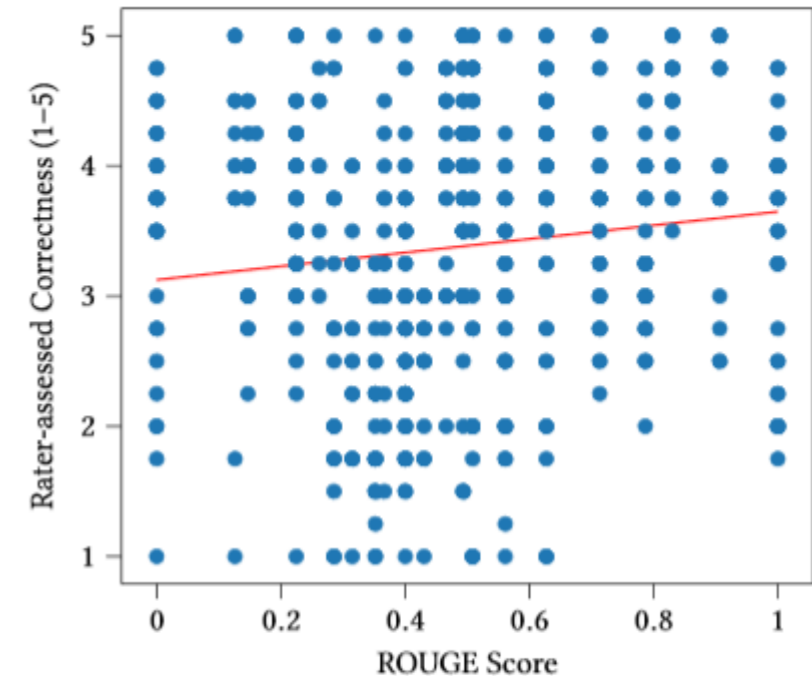


Figure 8: Scatter plot showing the distribution of ROUGE scores for each machine-generated code summary and the rater-assessed Correctness for participants who answered comprehension questions using that summary. We note a very weak correlation using Spearman's $\rho = 0.151$ ($p = 0.0004$). The best fit line is shown in red. High or low ROUGE scores do not appear to influence developer comprehension.

Software Metric Advice

- Use software metrics carefully
- Be careful about claims about human factors (e.g., readability) and quality, unless validated
- Calibrate metrics using your project history and the histories of other projects
- Metrics can be gamed: you get what you measure

Successful Measurement Programs

- Set solid measurement objectives and plans.
- Make measurement part of the process.
- Gain a thorough understanding of measurement.
- Focus on cultural issues.
- Create a safe environment to collect and report true data.
- Cultivate a predisposition to change.
- Develop a complementary suite of measures.

Questions when Choosing A Metric

- What is the purpose of this measure?
- What is the scope of this measure?
- What attribute are you trying to measure?
- What is the attribute's natural scale?
- What is the attribute's natural variability?
- What instrument are you using to measure the attribute, and what reading do you take from the instrument?
- What is the instrument's natural scale?
- What is the reading's natural variability (normally called measurement error)?
- What is the attribute's relationship to the instrument?
- What are the natural and foreseeable side effects of using this instrument?

[Cem Kaner and Walter P. Bond. "Software Engineering Metrics: What Do They Measure and How Do We Know?" 2004]

Beigel and Zimmermann Microsoft Survey

- “Suppose you could work with a team of data scientists and data analysts who specialize in studying how software is developed. Please list up to five questions you would like them to answer. Why do you want to know? What would you do with the answers?”

Top Questions (1/2)

- How do users typically use my application?
- What parts of a software product are most used and/or loved by customers?
- How effective are the quality gates we run at checkin?
- How can we improve collaboration and sharing between teams?
- What are best key performance indicators (KPIs) for monitoring services?
- What is the impact of a code change or requirements change to the project and tests?

Top Questions (2/2)

- What is the impact of tools on productivity?
- How do I avoid reinventing the wheel by sharing and/or searching for code?
- What are the common patterns of execution in my application?
- How well does test coverage correspond to actual code usage by our customers?
- What kinds of mistakes do developers make in their software? Which ones are the most common?
- What are effective metrics for ship quality?

Bottom Questions

- Which individual measures correlate with employee productivity (e.g., employee age, tenure, engineering skills, education, promotion velocity, IQ)?
- Which coding measures correlate with employee productivity (e.g., lines of code, time it take to build the software, a particular tool set, pair programming, number of hours of coding per day, language)?
- What metrics can be used to compare employees?
- How can we measure the productivity of a Microsoft employee?
- Is the number of bugs a good measure of developer effectiveness?
- Can I generate 100% test coverage?

Questions?

Next exciting episode:

Quality Assurance and Testing

