

Week: COMP 2120 / COMP 6120  
4 of 12  
METRICS

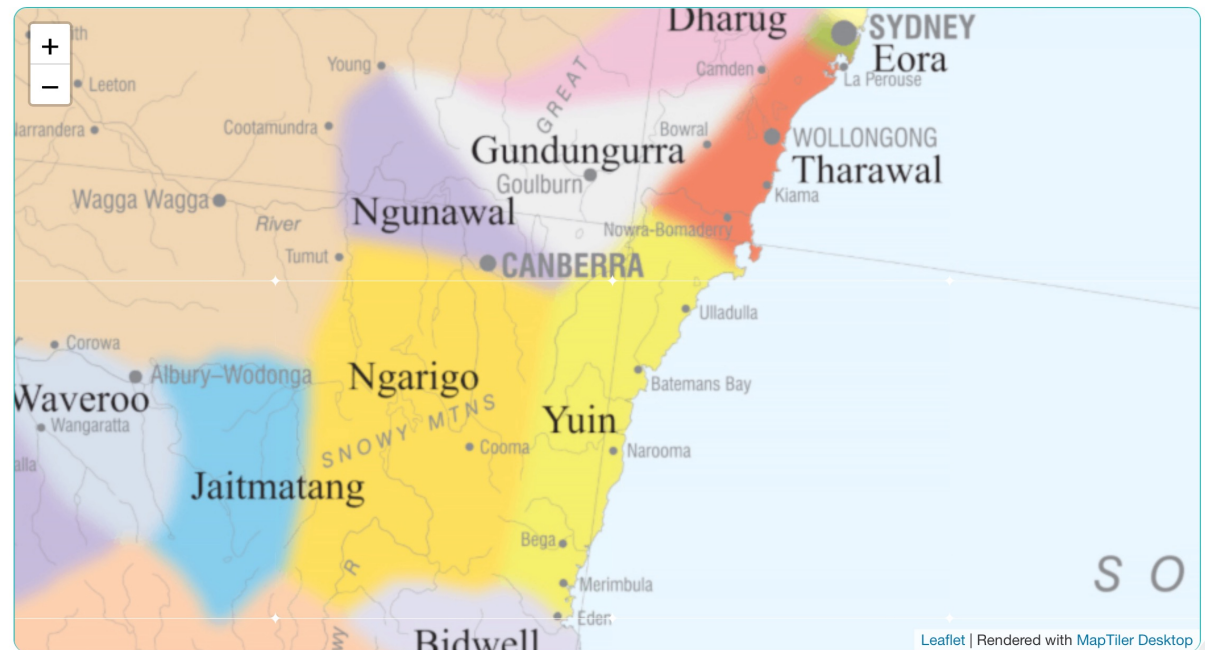
A/Prof Alex Potanin



# ANU Acknowledgment of Country



“We acknowledge and celebrate the First Australians on whose traditional lands we meet, and pay our respect to the elders past and present.”



<https://aiatsis.gov.au/explore/map-indigenous-australia>



# Today

- Understanding Large Systems
- Case Study: The Maintainability Index
- Case Study: Autonomous Vehicle Safety
- Measurement for Decision Making
- Understanding Your Data
- Metrics and Incentives
- Goals, Signals, Metrics

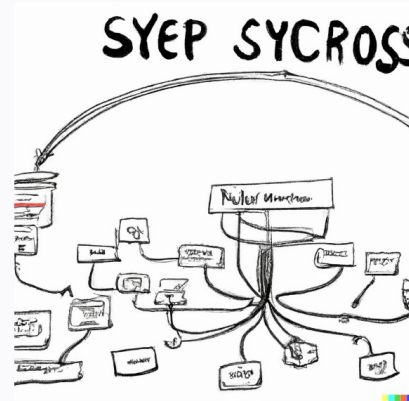
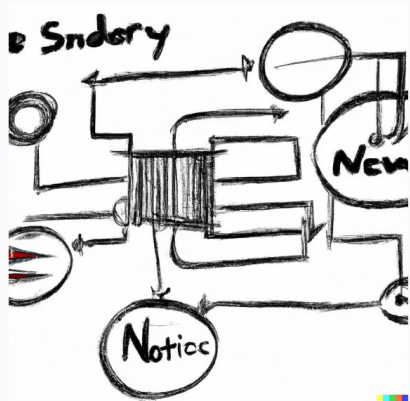


Edit the detailed description

Surprise me Upload →

pencil drawing of a large and complicated system with no words shown

Generate

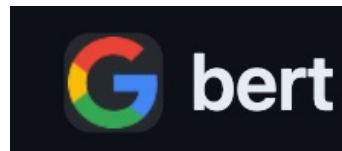


## Understanding Large Systems





Context: big ole pile of code.



...do something to it.

Like: Fix a bug, implement a feature, write a test...

**You cannot understand the  
entire system.**



# Goal



- *To develop and test a working model or set of working hypotheses about how (some part of) a system works.*
- Working model: an understanding of the pieces of the system (components), and the way they interact (connections).
- It is common in practice to consult documentation, experts.
- Prior knowledge/experience is also useful (see: frameworks, architectural patterns, design patterns).
- Today, we focus on individual information gathering via observation, probes, and hypothesis testing.



# Software constantly changes → Software is easy to change!



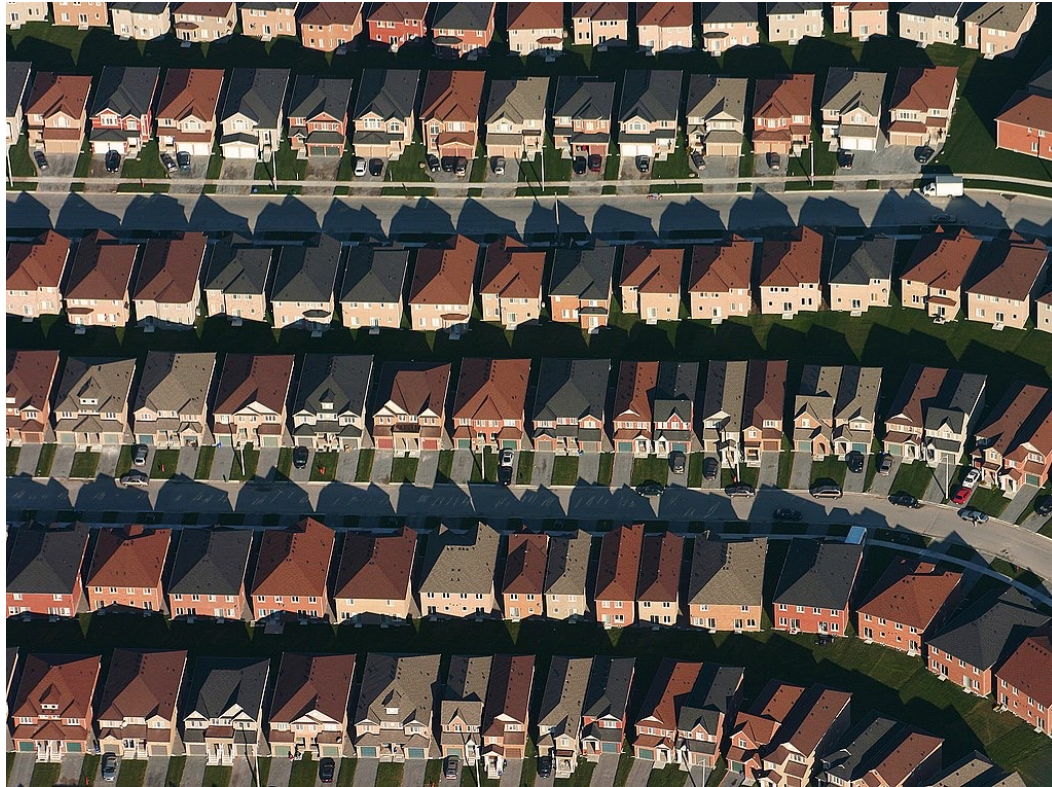
Guess so!

Is this wall  
load-  
bearing?

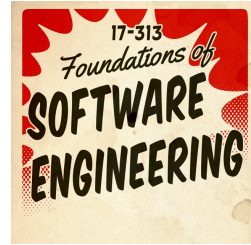




Software is a big redundant mess  
→ there's always something to copy  
as a starting point!

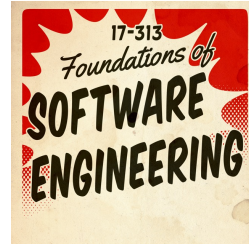


# 1. If code must run, it must have a beginning





## 2. If code must run, it must exist



```
0x08048416 <+16>: jg     DWORD PTR [ebp+0x8],0x1
0x08048419 <+18>: mov    0x804843c <main+56>
0x0804841b <+21>: mov    eax,DWORD PTR [ebp+0xc]
0x08048420 <+23>: mov    ecx,DWORD PTR [eax]
0x08048425 <+28>: mov    edx,0x8048520
0x08048429 <+33>: mov    eax,ds:0x8049648
0x0804842d <+37>: mov    DWORD PTR [esp+0x8],ecx
0x08048430 <+41>: mov    DWORD PTR [esp+0x4],edx
0x08048435 <+44>: call   DWORD PTR [esp],eax
0x0804843a <+49>: mov    eax,0x1
0x0804843c <+54>: jmp    0x8048459 <main+85>
0x0804843f <+59>: mov    eax,DWORD PTR [ebp+0xc]
0x08048442 <+62>: add    eax,0x4
0x08048444 <+64>: mov    eax,DWORD PTR [eax]
0x08048448 <+68>: mov    DWORD PTR [esp+0x4],eax
0x0804844c <+72>: lea   eax,[esp+0x10]
0x0804844f <+75>: mov    DWORD PTR
0x08048454 <+78>: call
```



# The Beginning: Entry Points



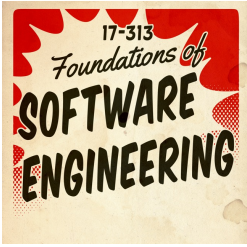
Some trigger that causes code to run.

- **Locally installed programs:** run cmd, OS launch, I/O events, etc.
- **Local applications in dev:** build + run, test, deploy (e.g. docker)
- **Web apps server-side:** Browser sends HTTP request (GET/POST)
- **Web apps client-side:** Browser runs JavaScript





# Code must exist. But where?

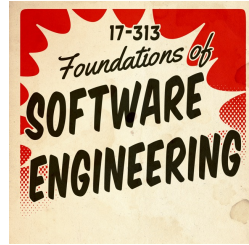


Helps to identify what's knowable and what's changeable

- **Locally installed programs:** run cmd, OS launch, I/O events, etc.
  - Binaries (machine code) on your computer
- **Local applications in dev:** build + run, test, deploy (e.g. docker)
  - Source code in repository (+ dependencies)
- **Web apps server-side:** Browser sends HTTP request (GET/POST)
  - Code runs remotely (you can only observe outputs)
- **Web apps client-side:** Browser runs JavaScript
  - Source code is downloaded and run locally (see: browser dev tools!)



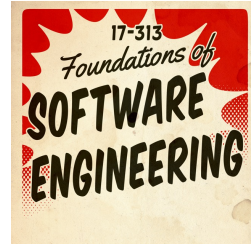
# Side note on build systems



- Basically the same across languages / platforms
  - Make, maven, gradle, grunt, bazel, etc.
- **Goal:** Source code + dependencies + config → runnables
- Common themes:
  - Dependency management (repositories, versions, etc)
  - Config management (platform-specific features, file/dir names, IP addresses, port numbers, etc)
  - Runnables (start, stop?, test)
  - Almost always have 'debug' mode and help ('-h' or similar)
  - Almost always have one or more "build" directories (= not part of source repo)



# Can running code be Probed/Understood/Edited?



Transparent



Source code built locally

(P+U+E)

Translucent



Binaries running locally

Open source

(P+U)

Closed source

(P)

Opaque



Server-side apps running remotely

Open source

(U)

Closed source

-



# Exercise Time

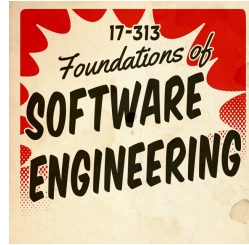
- NYTimes quiz: <http://bit.ly/problemQuiz>



**Beware of cognitive biases.**



# Beware of cognitive biases

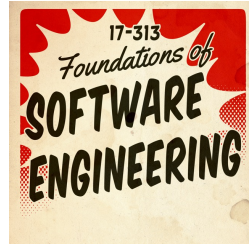


- **anchoring**
- **confirmation bias**
- **congruence bias**: The tendency to test hypotheses exclusively through direct testing, instead of testing possible alternative hypotheses
- conservatism (belief revision)
- curse of knowledge
- default effect
- expectation bias
- overconfidence effect
- plan continuation bias
- pro innovation bias
- recency illusion

[https://en.wikipedia.org/wiki/List\\_of\\_cognitive\\_biases](https://en.wikipedia.org/wiki/List_of_cognitive_biases)



# Static (+dynamic) information gathering



- **Basic needs:**

- **Code/file search and navigation**
- Code editing (probes)
- Execution of code, tests
- Observation of output (observation)

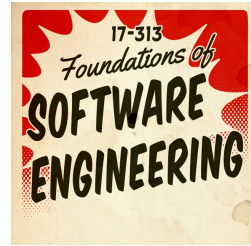
At the command line: **grep** and **find!**  
(Do a web search for tutorials)

- **Many choices here on tools! Depends on circumstance.**

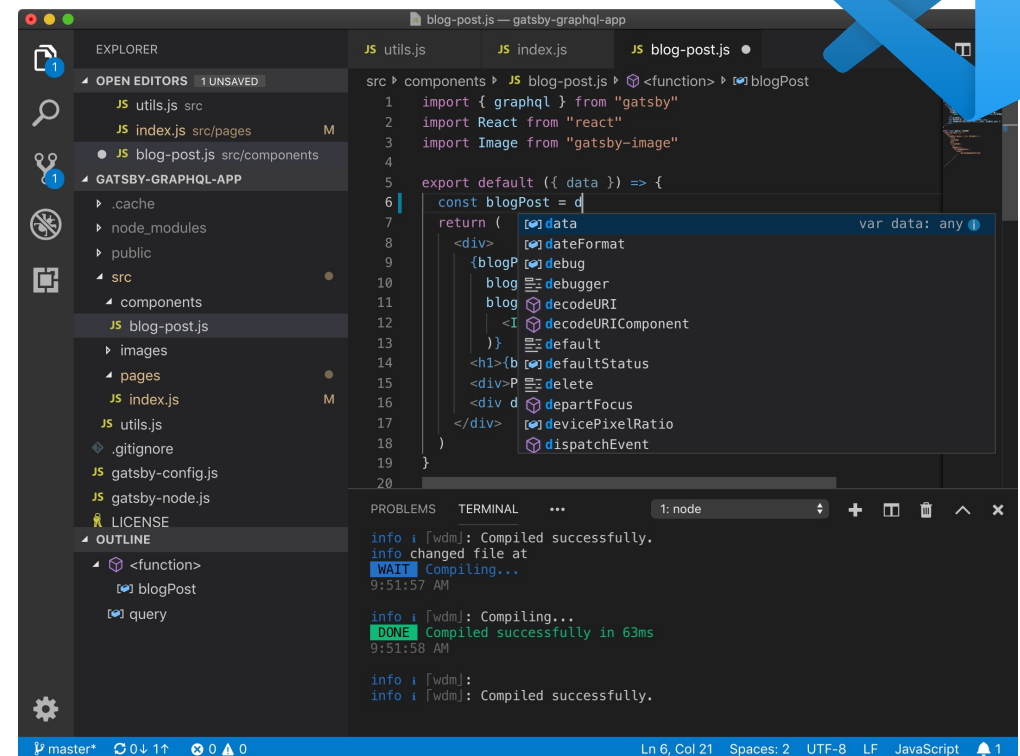
- grep/find/etc. Having a command on Unix tools is invaluable
- A decent IDE
- Debugger
- Test frameworks + coverage reports
- Google (or your favorite web search engine)



# Static Information Gathering



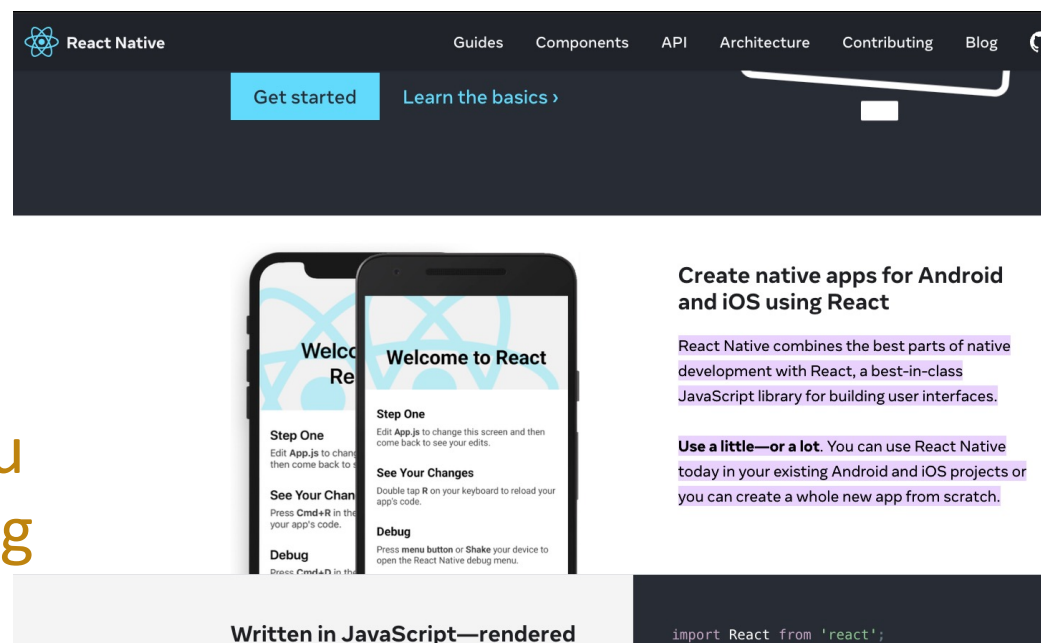
- Please configure and use a *legitimate* IDE.
- No favorites? We recommend VSCode and IntelliJ IDEA.
- Why?
  - “search all files”
  - “jump to definition”
  - “download dependency source”
- Remember: real software is too complicated to keep in your head.





# Consider documentation/tutorials judiciously

- Great for discovering entry points!
- Can teach you about general structure, architecture.
  - Forward-reference to architectural patterns!
- As you gain experience, you will recognize more of these, and you will immediately know something about how the program works.
- For example, next time you work on a mobile app...

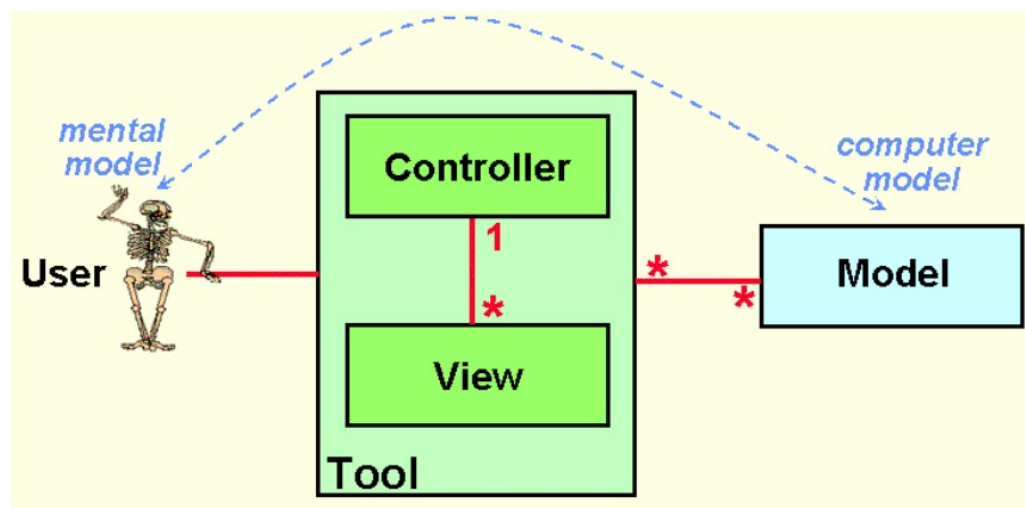


# Consider documentation/tutorials judiciously

## A bit of Model View Controller history

Trygve Reenskaug discovered MVC at Xerox PARC in 1978.

*The essential purpose of MVC is to bridge the gap between the human user's mental model and the digital model that exists in the computer [Trygve Reenskaug].*

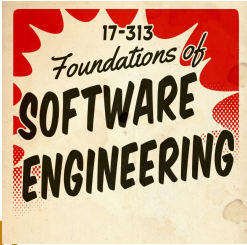


<http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>

<https://medium.com/swlh/elements-of-mvc-in-react-9382de427c09>

The term *mental-model* stuck with  211 |  represents the essence of our

# Dynamic Information Gathering



- Key principle 1: change is a useful primitive to inform mental models about a software system.
- Key principle 2: systems almost always provide some kind of starting point.
- Put simply:
  1. Build it.
  2. Run it.
  3. Change it.
  4. Run it again.
- Can provide information both *bottom up* or *top down*, depending on the situation.

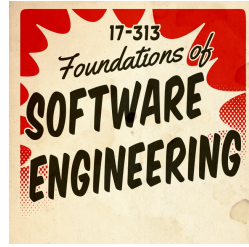


# Probes - Observe, control or “lightly” manipulate execution

- `printf(“here”)`
- Turning on automatic debug info logging
- Breakpoints
- Sophisticated debugging tools
  - Breakpoint, eval, step through / step over
  - (Some tools even support remote debugging)
- Delete debugging (equivalent of ``kill -9``)



# Step 0: sanity check basic model + hypotheses.



- Confirm that you can build and run the code.
  - Ideally *both* using the tests provided, *and* by hand.
- Confirm that the code you are running *is the code you built*.
- Confirm that you can make *an externally visible change*.
- How? Where? Starting points:
  - Run an existing test, change it.
  - Write a new test.
  - Change the code, write or rerun a test that should notice the change.
- **Make sure the changes persist if you want them to.**
  - Distinguish between source repository and build/deploy directories.







Software Engineering: Principles, practices (technical and non-technical) for confidently building high-quality software.

What does this mean?  
How do we know?  
→ *Measurement* and metrics are **key** concerns.



# Poll Everywhere Time!

Join by Web [PollEv.com/potantin](https://PollEv.com/potantin) Join by Text Send **potantin** to **22333** 


**What is the first thing you will try in 1-2 words when faced with a very large code base?**  0

Join by Web  
[PollEv.com/potantin](https://PollEv.com/potantin)

Join by Text  
Send **potantin** to **22333**

Join by QR code

Scan with your camera app







Edit the detailed description

Surprise me

Upload



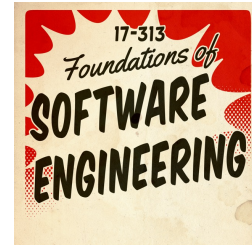
pencil drawing of maintenance worker with no written words

Generate



## Case Study: The Maintainability Index

# Visual Studio (since 2007)

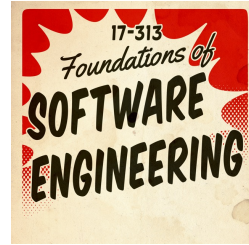


“Maintainability Index calculates an index value between 0 and 100 that represents the relative ease of maintaining the code. A **high value means better maintainability**. Color coded ratings can be used to quickly identify trouble spots in your code. A **green rating is between 20 and 100** and indicates 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



# Visual Studio (since 2007)



The screenshot shows the 'Code Metrics Viewer' window in Visual Studio. It displays a table with columns for 'Hierarchy', 'Maintainability Index', 'Cyclomatic Complexity', 'Class Coupling', 'Depth of Inheritance', and 'Lines of Code'. The 'Maintainability Index' column is highlighted with a black box. The table data is as follows:

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

- 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.



# Design rationale (from MSDN blog)



- "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."
- <http://blogs.msdn.com/b/codeanalysis/archive/2007/11/20/maintainability-index-range-and-meaning.aspx>



# The Index



Maintainability Index =

MAX(0,(171 –

5.2 \* log(Halstead Volume) –

0.23 \* (Cyclomatic Complexity) –

16.2 \* log(Lines of Code)

)\*100 / 171)

## Calculation [\[ edit \]](#)

For a given problem, let:

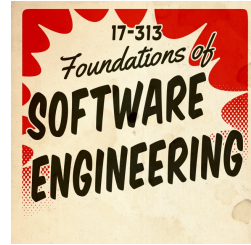
- $\eta_1$  = the number of distinct operators
- $\eta_2$  = the number of distinct operands
- $N_1$  = the total number of operators
- $N_2$  = the total number of operands

From these numbers, several measures can be calculated:

- Program vocabulary:  $\eta = \eta_1 + \eta_2$
- Program length:  $N = N_1 + N_2$
- Calculated estimated program length:  $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- **Volume:**  $V = N \times \log_2 \eta$
- Difficulty :  $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
- Effort:  $E = D \times V$



# Origins



- 1992 Paper at the International Conference on Software Maintenance by Paul Oman and Jack Hagemester

$$171 - 5.2 \ln(HV) - 0.23CC - 16.2 \ln(LOC) + 50.0 \sin \sqrt{2.46 * COM}$$

COM = percentage of comments

- Developers rated a number of HP systems in C and Pascal
- Statistical regression analysis to find key factors among 40 metrics



# The Index



Maintainability Index =

$\text{MAX}(0, (171 -$

$5.2 * \log(\text{Halstead Volume}) -$

$0.23 * (\text{Cyclomatic Complexity}) -$

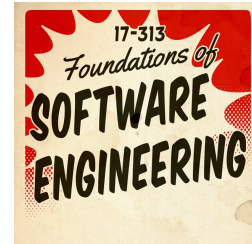
$16.2 * \log(\text{Lines of Code})$

$)* 100 / 171)$



# Thoughts?

- Metric seems attractive
- Easy to compute
- Often seems to match intuition
- Parameters seem almost arbitrary: code (few developers, unclear standards)
- All metrics related to size: just metrics
- Original 1992 C/Pascal programs  
Java/JS/C# code



## ARIE VAN DEURSEN

Software engineering in theory and practice

HOME ABOUT

29  
AUG  
2014

### Think Twice Before Using the “Maintainability Index”

posted in [Research](#) by [Arie van Deursen](#)

This is a quick note about the “Maintainability Index”, a metric aimed at assessing software maintainability, as I recently run into developers and researchers who are (still) using it.

Hierarchy	Maint.	Cyclo.	Depth.	Class C.	Lines of Code
One or more projects we					
Tailspin.SimplySoft.Repos	73	772	1	75	2,758
Tailspin.Model (Debug)	93	667	3	81	958
Tailspin.Admin.App (Deb)	83	217	2	29	436
Tailspin.Web (Debug)	77	179	4	101	426
Tailspin.Infrastructure (De	79	220	2	72	413
Tailspin.Test.Model (Deb	73	45	1	27	158

The Maintainability Index was introduced at the [International Conference on Software Maintenance](#) in 1992. To date, it is included in [Visual Studio](#) (since 2007), in the recent (2012) [JSComplexity](#) and [Radon](#) metrics reporters for Javascript and Python, and in older metric tool suites such as [verifysoft](#).

At first sight, this sounds like a great success of knowledge transfer from academic research to industry practice. Upon closer inspection, the Maintainability Index turns out to be problematic.


The Original Index






# Poll Everywhere Time!

Join by Web [PollEv.com/potanim](https://PollEv.com/potanim) Join by Text Send **potanim** to **22333**



**Would You Use Maintainability Index in Your Projects?**  0

Yes **(A)**

No **(B)**

I have no idea how! **(C)**



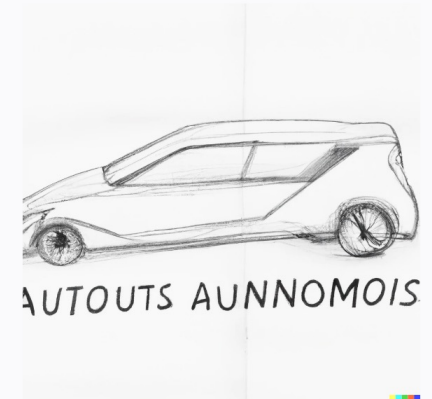
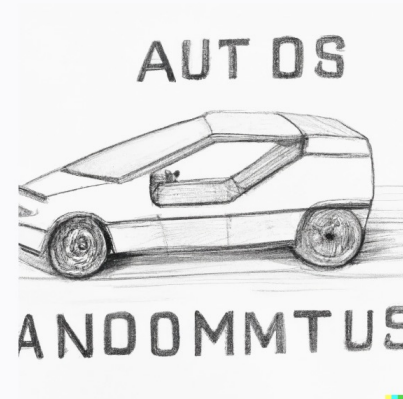
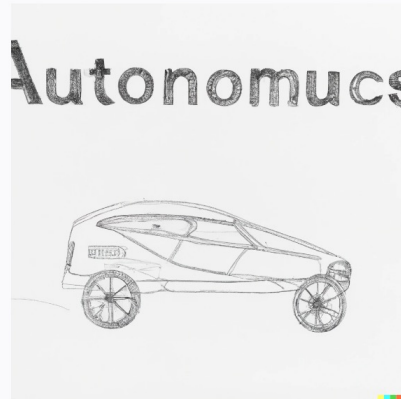
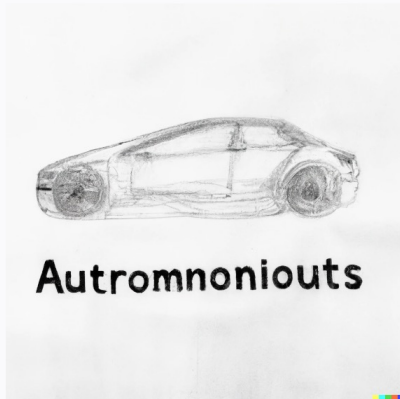


Edit the detailed description

Surprise me Upload →

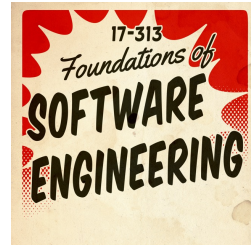
pencil drawing of an autonomous vehicle with no written words

Generate

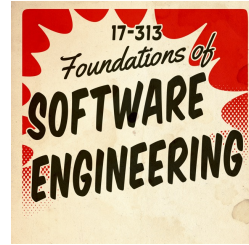


## Case Study: Autonomous Vehicle Safety

# How can we judge AV software quality (e.g. safety)?



# Test coverage

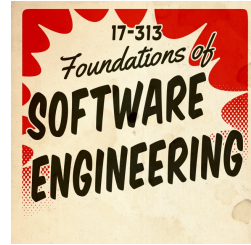


- Amount of code executed during testing.
- Statement coverage, line coverage, branch coverage, etc.
- E.g. 75% branch coverage → 3/4 if-else outcomes have been executed

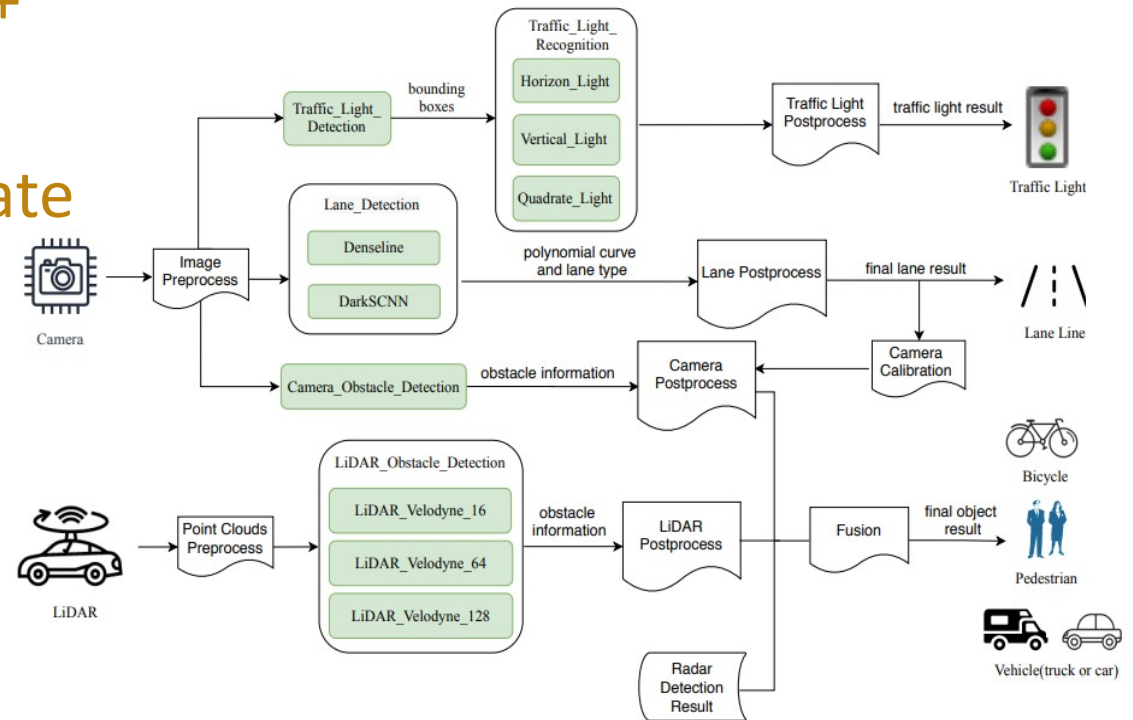
```
1698 : const TrajectoryPoint& StGraphData::init_point() const { return init_point_; }
2264 : const SpeedLimit& StGraphData::speed_limit() const { return speed_limit_; }
212736 : double StGraphData::cruise_speed() const {
[- + ]: 212736 : return cruise_speed_ > 0.0 ? cruise_speed_ : FLAGS_default_cruise_speed;
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]));
```



# Model Accuracy



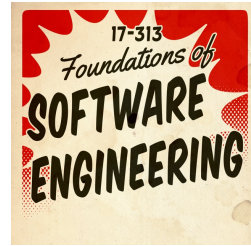
- Train machine-learning models on labelled data (sensor data + ground truth).
- Compute accuracy on a separate labelled test set.
- E.g. 90% accuracy implies that object recognition is right for 90% of the test inputs.





# Failure Rate

- Frequency of crashes/fatalities
- Per 1000 rides, per million miles, per month (in the news)



# Mileage

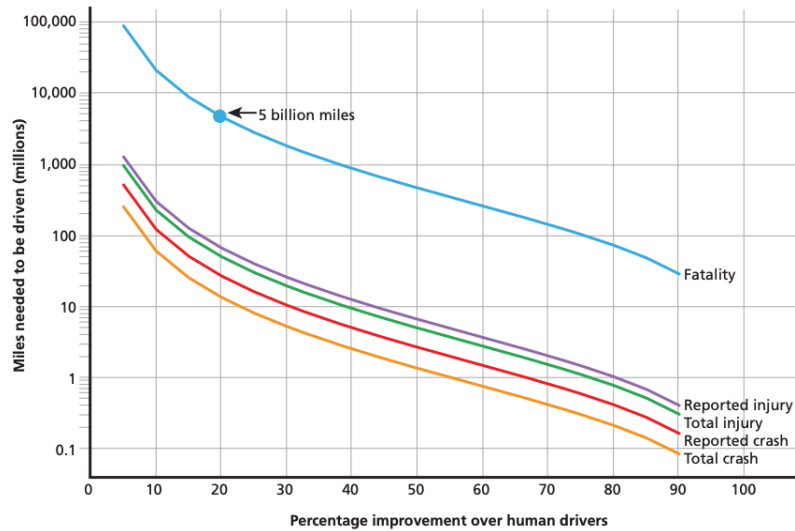


## Driving to Safety

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

Nidhi Kalra, Susan M. Paddock

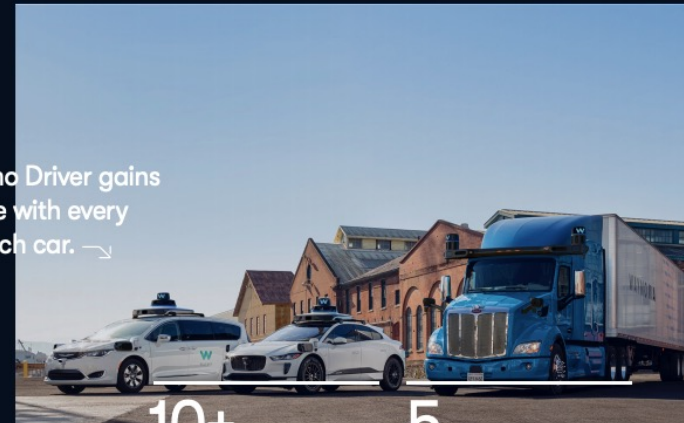
Figure 3. Miles Needed to Demonstrate with 95% Confidence that the Autonomous Vehicle Failure Rate Is Lower than the Human Driver Failure Rate



## Building the World's Most Experienced Driver™



The Waymo Driver gains experience with every mile, in each car.



10+

More than a Decade of Autonomous Driving in More than 10 States

5

Generations of Autonomously Driven Vehicles

15+

Billion Autonomously Driven Miles in Simulation

20+

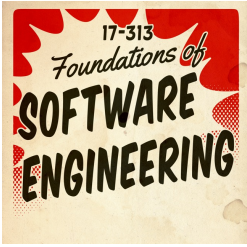
Million Real-World Miles on Public Roads

Source: waymo.com/safety (September 2021)





# Activity



Think of “pros” and “cons” for using various quality metrics to judge AV software.

- Test coverage
- Model accuracy
- Failure rate
- Mileage
- Size of codebase
- Age of codebase
- Time of most recent change
- Frequency of code releases
- Number of contributors
- Amount of code documentation




# STOP sign or 45 speed limit?




“Robust Physical-World Attacks on Deep Learning Models” by Kevin Eykholt et al. CVPR’18



# Poll Everywhere Time!

Join by Web [PollEv.com/potantin](https://PollEv.com/potantin) Join by Text Send **potantin** to **22333** 


**What metric would YOU use to judge the quality of AV software?**  0

Join by Web  
[PollEv.com/potantin](https://PollEv.com/potantin)

Join by Text  
Send **potantin** to **22333**

Join by QR code

Scan with your camera app





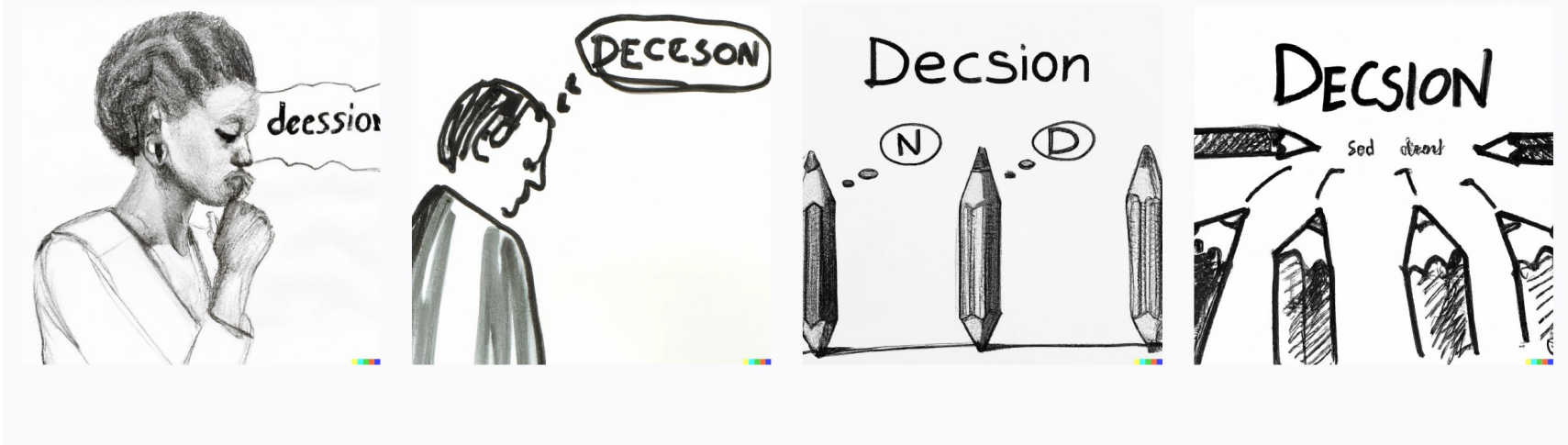
Edit the detailed description

Surprise me

Upload →

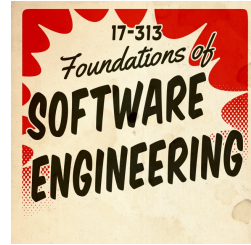
pencil drawing of decision makers with no words

Generate



## Measurement for Decision Making

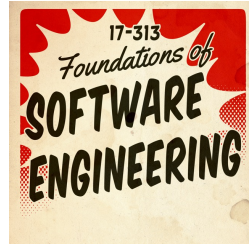
# What is Measurement?



- 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. – 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 ...”



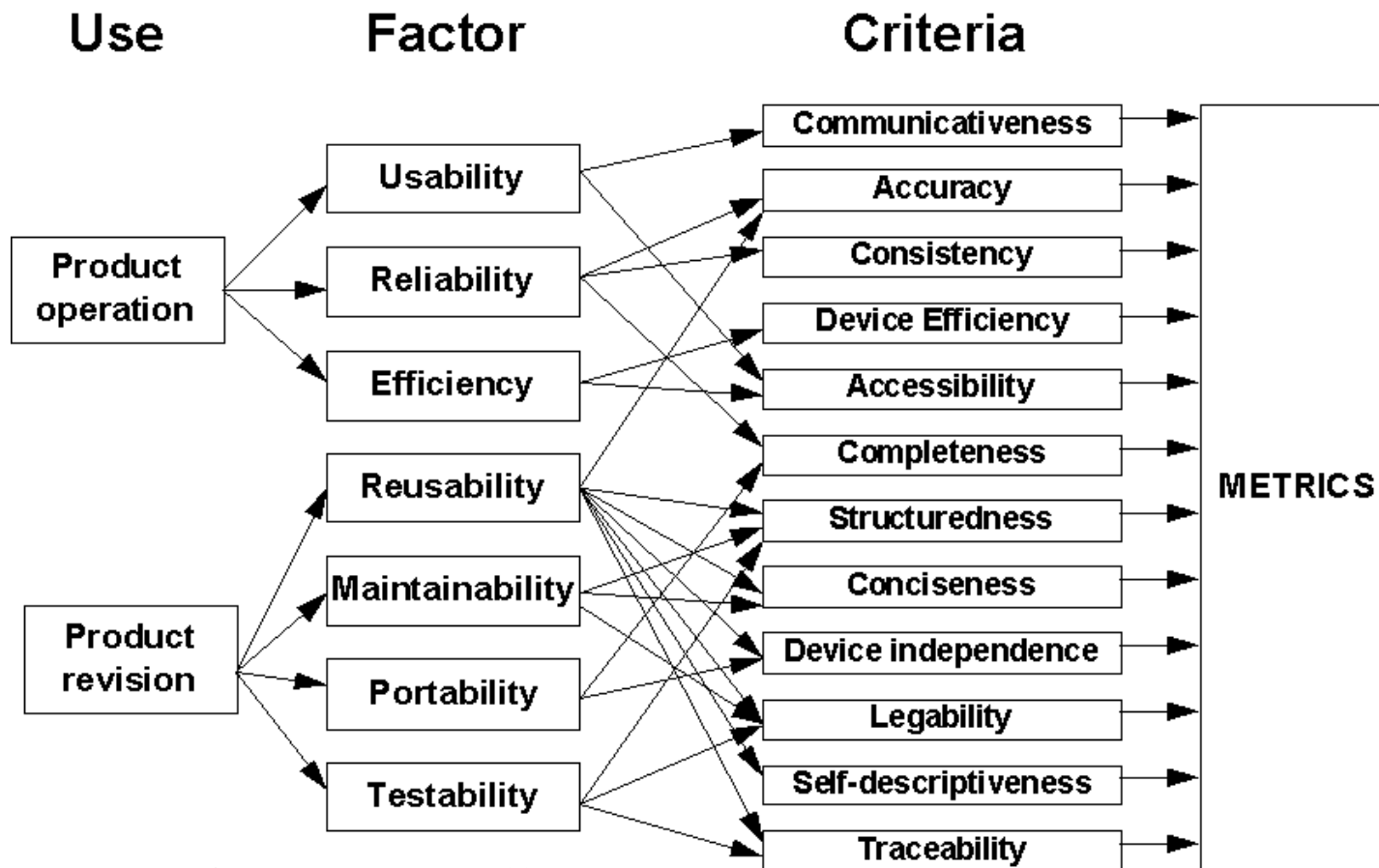
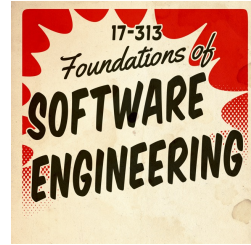
# Software Quality Metrics



- IEEE 1061 definition: “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.”
- Metrics have been proposed for many quality attributes; may define own metrics



# External attributes: Measuring Quality

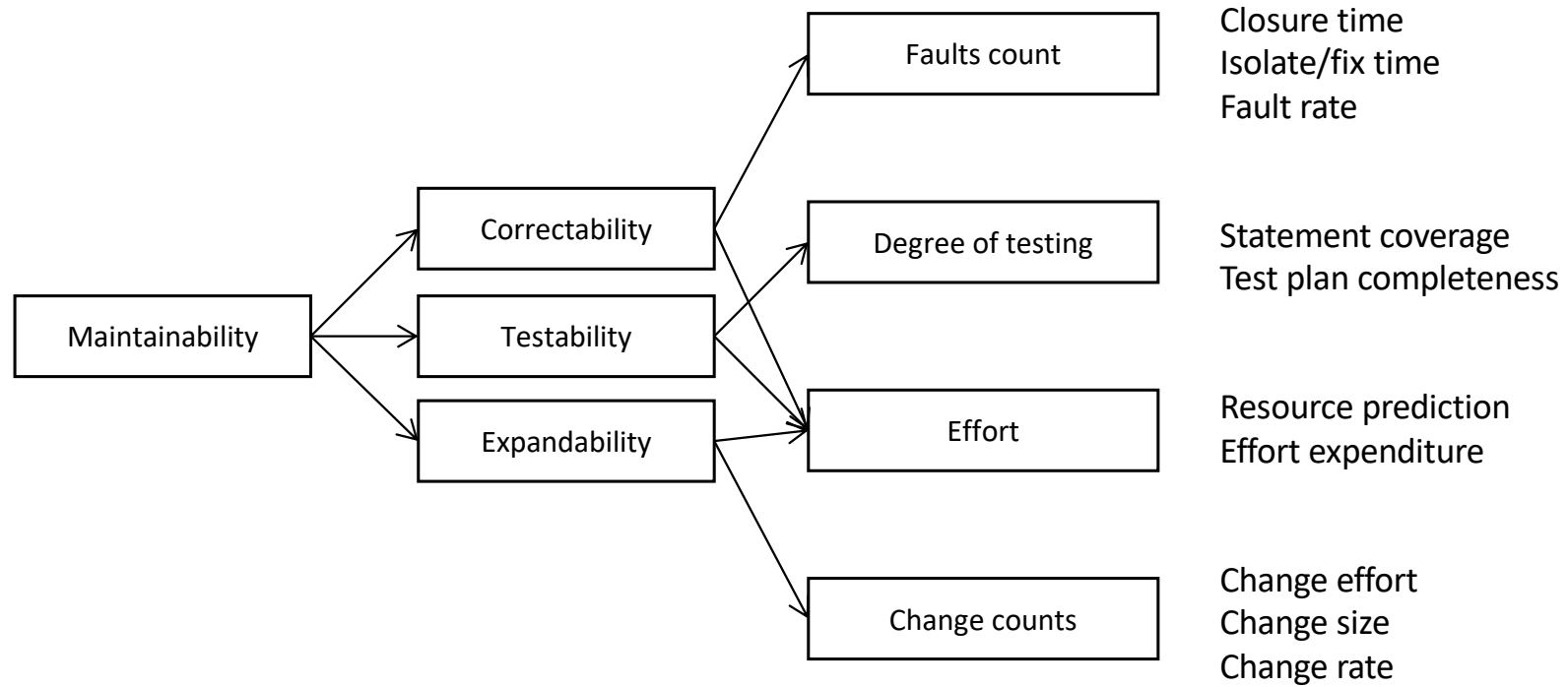
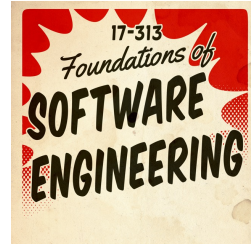


McCall model has 41 metrics to measure 23 quality criteria from 11 factors





# Decomposition of Metrics

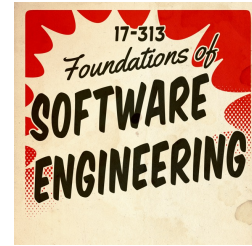


# Example: Code Complexity via Lines of Code

- Easy to measure

```
> wc -l file1 file2...
```

LOC	projects
450	Expression Evaluator
2,000	Sudoku
100,000	Apache Maven
500,000	Git
3,000,000	MySQL
15,000,000	gcc
50,000,000	Windows 10
2,000,000,000	Google (MonoRepo)



# Normalising 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

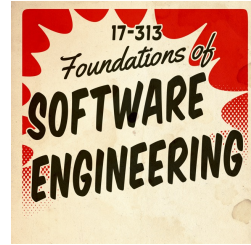
```
for (i = 0; i < 100; i += 1) printf("hello"); /* How many lines of code is this? */
```

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

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



# Normalisation per 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 Volume

- Introduced by Maurice Howard Halstead in 1977
- Halstead Volume =  
number of operators/operands \*  
 $\log_2(\text{number of distinct operators/operands})$
- Approximates size of elements and vocabulary

## Calculation [\[ edit \]](#)

For a given problem, let:

- $\eta_1$  = the number of distinct operators
- $\eta_2$  = the number of distinct operands
- $N_1$  = the total number of operators
- $N_2$  = the total number of operands

From these numbers, several measures can be calculated:

- Program vocabulary:  $\eta = \eta_1 + \eta_2$
- Program length:  $N = N_1 + N_2$
- Calculated estimated program length:  $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- **Volume:**  $V = N \times \log_2 \eta$
- Difficulty:  $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
- Effort:  $E = D \times V$



# Halstead Volume – Example (Do At Home)

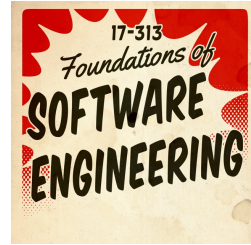


```
main() {  
    int a, b, c, avg;  
    scanf("%d %d %d", &a, &b, &c);  
    avg = (a + b + c) / 3;  
    printf("avg = %d", avg);  
}
```

Operators/Operands: main, (), {}, int, a, b, c, avg, scanf, (), "...", &, a, &, b, &, c, avg, =, a, +, b, +, c, (), /, 3, printf, (), "...", avg



# Cyclomatic Complexity



- Proposed by McCabe 1976
- Based on control flow graph, measures linearly independent paths through a program
  - $\sim$  = number of decisions
  - Number of test cases needed to achieve branch coverage

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

*"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 (aka CK Metrics)



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

Shyam R. Chidamber, Chris F. Kemerer.

A Metrics suite for Object Oriented design.

M.I.T. Sloan School of Management E53-315. 1993.

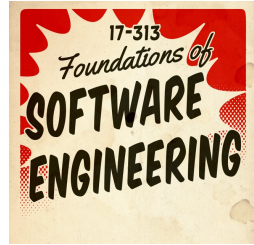
<http://uweb.txstate.edu/~mg43/CS5391/Papers/Metrics/OOMetrics.pdf>





# What software qualities do we care about? (examples)

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



# What process qualities do we care about? (examples)



- 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...



# 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.

D. Hubbard, How to Measure Anything, 2010



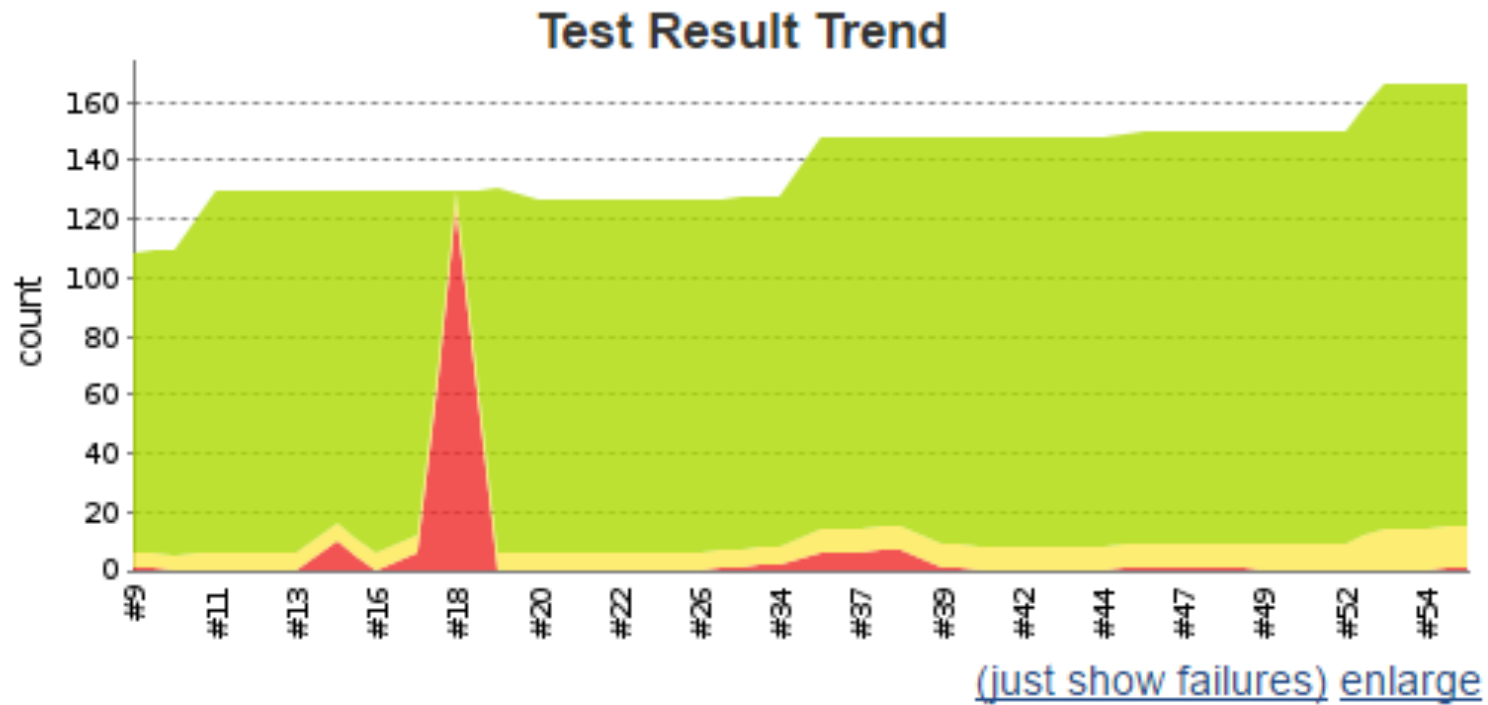
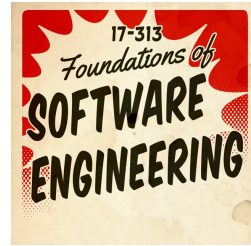
# Measurement for Decision Making



- Fund project?
- More testing?
- Fast enough? Secure enough?
- Code quality sufficient?
- Which feature to focus on?
- Developer bonus?
- Time and cost estimation? Predictions reliable?



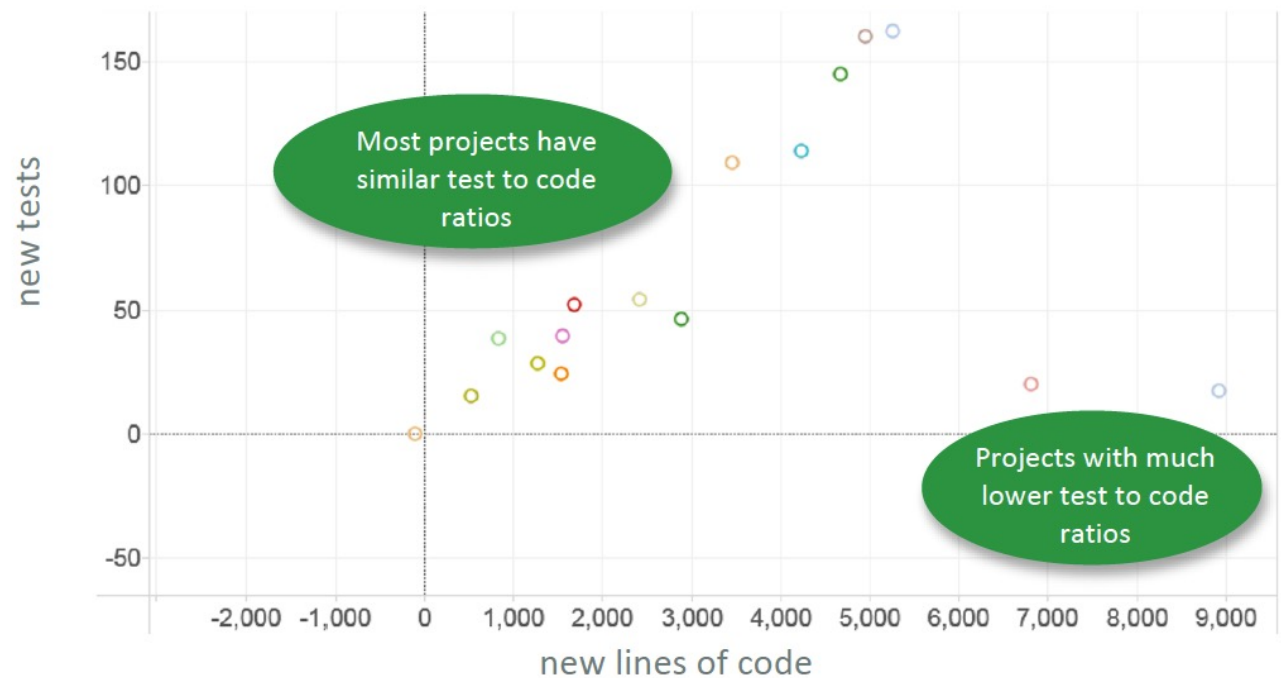
# Trend analyses



# Benchmark-Based Metrics



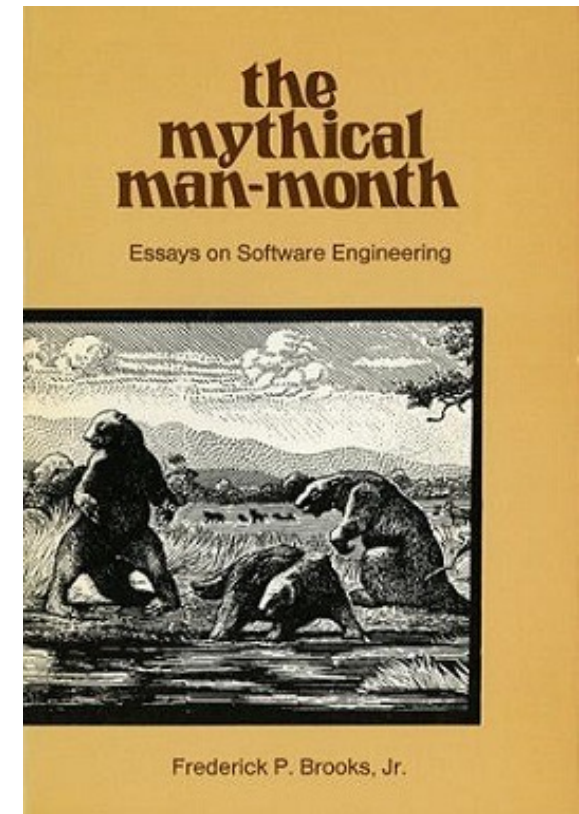
- Monitor many projects or many modules, get typical values for metrics
- Report deviations



# Example: Antipattern in effort estimation

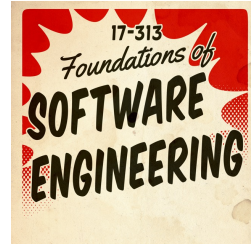


- IBM in the 60's: Would account in “person-months”  
e.g. Team of 2 working 3 months = 6 person-months
- LoC ~ Person-months ~ \$\$\$
- Brooks: *“Adding manpower to a late software project makes it later.”*





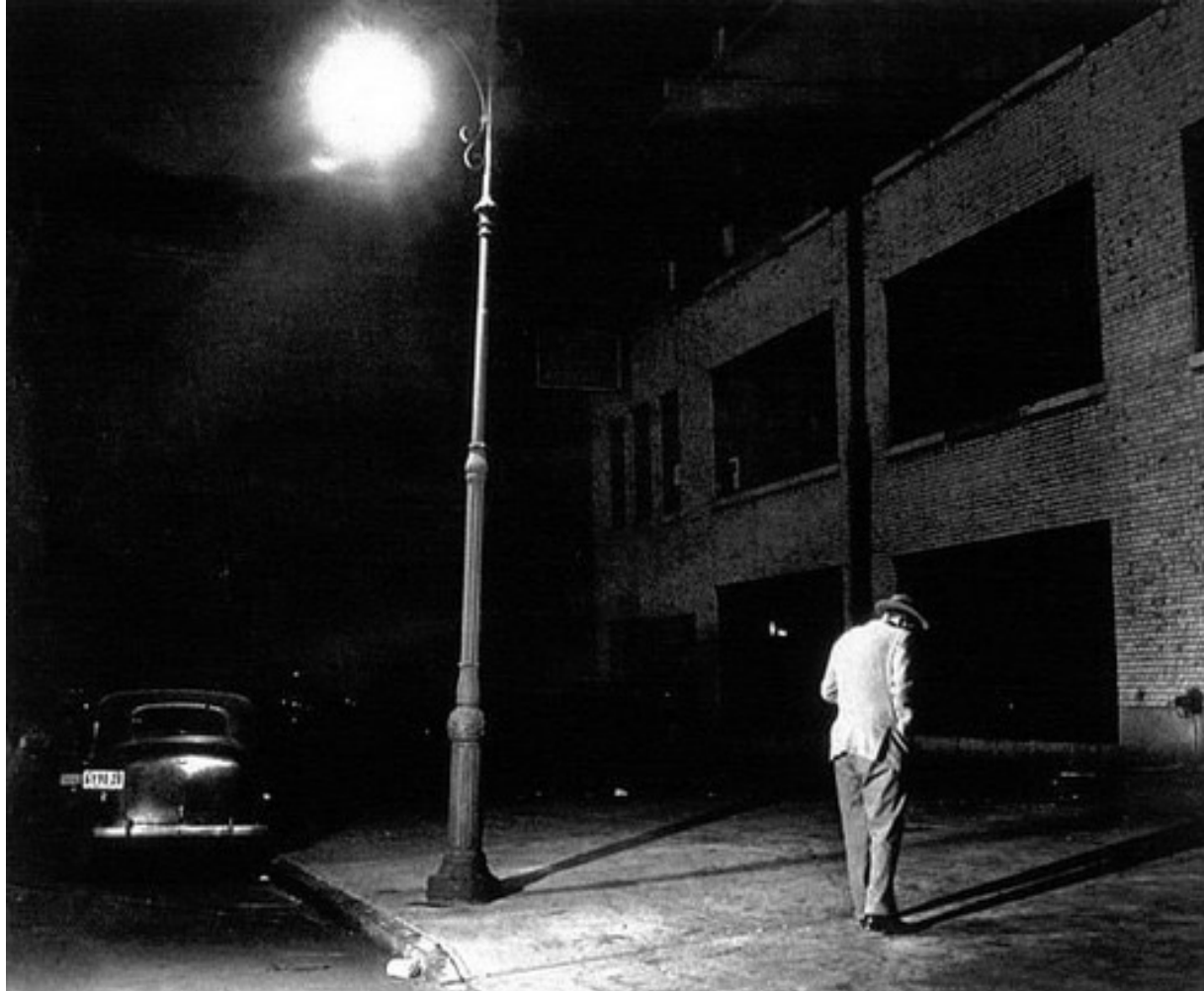
# Questions to consider



- What properties do we care about, and how do we measure it?
- 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? Check in gate? Once a month? Etc.
- What are potentially negative side effects or incentives?



# Measurement is Difficult



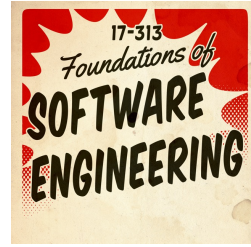
# The streetlight effect

- 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.





# What could possibly go wrong?



- **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...



- 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 oral contraceptive increased to two
- “...The absolute r relative increase indeed 100 perce

## COVID-19 Vaccines: Myth Versus Fact



### Featured Experts:



Gabor David Kelen, M.D.



Lisa Maragakis, M.D., M.P.H.

Updated on March 10, 2022

**N**ow that the U.S. Food and Drug Administration has authorized vaccines for COVID-19, and their distribution has begun, [Lisa Maragakis, M.D., M.P.H.](#), senior director of infection prevention, and [Gabor Kelen, M.D.](#),



eration  
amber  
pills...”  
s the  
s) was





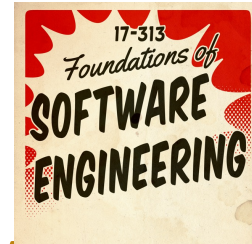
# Measurement scales



- Scale: the type of data being measured.
- The scale dictates what sorts of analysis/arithmetic is legitimate or meaningful.
- Your options are:
  - Nominal: categories
  - Ordinal: order, but no magnitude.
  - Interval: order, magnitude, but no zero.
  - Ratio: Order, magnitude, and zero.
  - Absolute: special case of ratio.



# Nominal/categorical scale

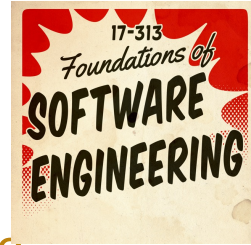


- Entities classified with respect to a certain attribute. Categories are jointly exhaustive and mutually exclusive.
  - No implied order between categories!
- Categories can be represented by labels or numbers; however, they do not represent a magnitude, arithmetic operation have no meaning.
- Can be compared for identity or distinction, and measurements can be obtained by counting the frequencies in each category. Data can also be aggregated.

Entity	Attribute	Categories
Application	Purpose	E-commerce, CRM, Finance
Application	Language	Java, Python, C++, C#
Fault	Source	assignment, checking, algorithm, function, interface, timing



# Ordinal scale

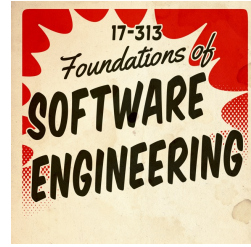


- Ordered categories: maps a measured attribute to an ordered set of values, but no information about the magnitude of the differences between elements.
- Measurements can be represented by labels or numbers, BUT: if numbers are used, *they do not represent a magnitude*.
  - Honestly, try not to do that. It eliminates temptation.
- You cannot: add, subtract, perform averages, etc (arithmetic operations are out).
- You can: compare with operators (like “less than” or “greater than”), create ranks for the purposes of rank correlations (Spearman’s coefficient, Kendall’s  $\tau$ ).

Entity	Attribute	Values
Application	Complexity	Very Low, Low, Average, High, Very High
Fault	Severity	1 – Cosmetic, 2 – Moderate, 3 – Major, 4 – Critical



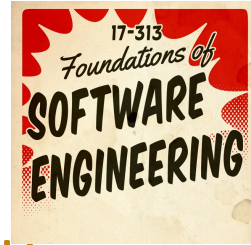
# Interval scale



- Has order (like ordinal scale) and magnitude.
  - The intervals between two consecutive integers represent equal amounts of the attribute being measured.
- Does NOT have a zero: 0 is an arbitrary point, and doesn't correspond to the absence of a quantity.
- Most arithmetic (addition, subtraction) is OK, as are mean and dispersion measurements, as are Pearson correlations. Ratios are not meaningful.
  - Ex: The temperature yesterday was 64 F, and today is 32 F. Is today twice as cold as yesterday?
- Incremental variables (quantity as of today – quantity at an earlier time) and preferences are commonly measured in interval scales.



# Ratio Scale

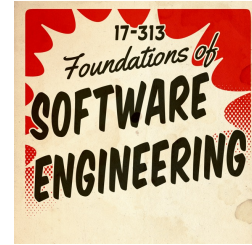


- An interval scale that has a true zero that actually represents the absence of the quantity being measured.
- All arithmetic is meaningful.
- Absolute scale is a special case, measurement simply made by counting the number of elements in the object.
  - Takes the form “number of occurrences of X in the entity.”

Entity	Attribute	Values
Project	Effort	Real numbers
Software	Complexity	Cyclomatic complexity



# Summary of Scales



Scale level	Examples	Operators	Possible analyses
<i>Quantitative scales</i>			
<b>Ratio</b>	size, time, cost	$*$ , $/$ , $\log$ , $\sqrt{\quad}$	geometric mean, coefficient of variation
<b>Interval</b>	temperature, marks, judgement expressed on rating scales	$+$ , $-$	mean, variance, correlation, linear regression, analysis of variance (ANOVA), ...
<i>Qualitative scales</i>			
<b>Ordinal</b>	complexity classes	$<$ , $>$	median, rank correlation, ordinal regression
<b>Nominal</b>	feature availability	$=$ , $\neq$	frequencies, mode, contingency tables



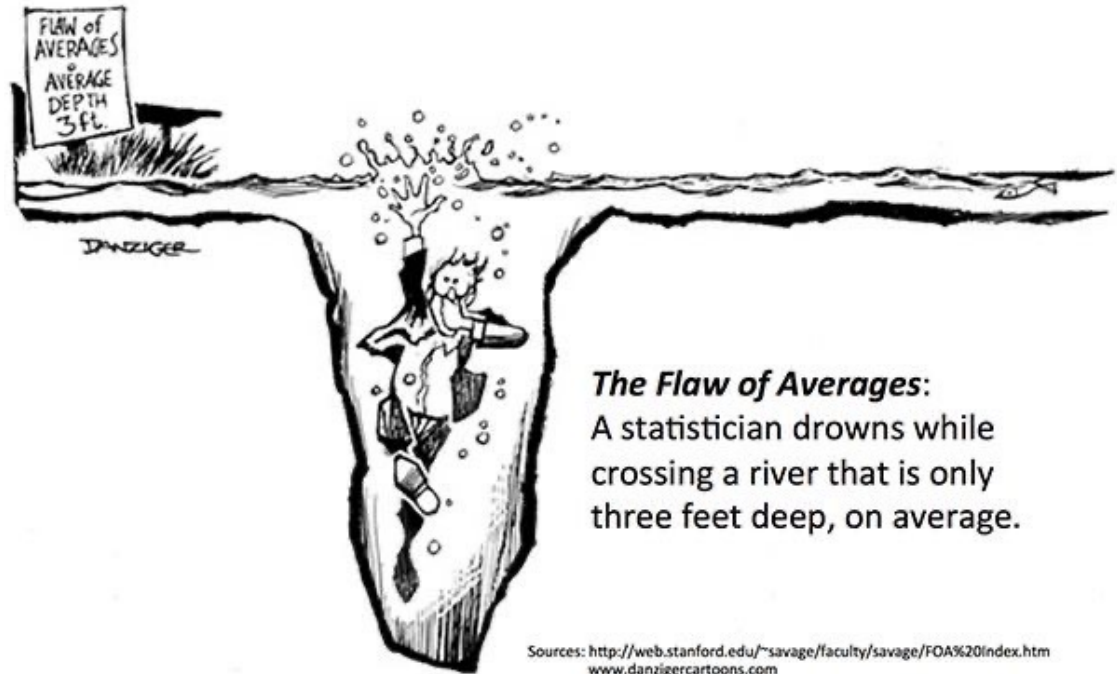
# Poll Everywhere Time!

The screenshot shows a mobile-style poll interface. At the top, there are two options to join: 'Join by Web' with the URL [PollEv.com/potanim](https://PollEv.com/potanim) and 'Join by Text' with the instruction 'Send **potanim** to **22333**'. A QR code is located in the top right corner. Below this is the poll question: 'What metric have you used the most before?' with a heart icon and the number '0' indicating zero votes. The interface is split into two columns. The left column contains the same 'Join by Web' and 'Join by Text' options. The right column is titled 'Join by QR code' and 'Scan with your camera app', featuring a large QR code for scanning.





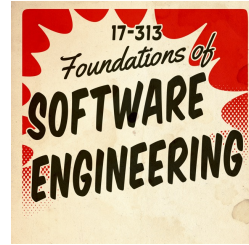




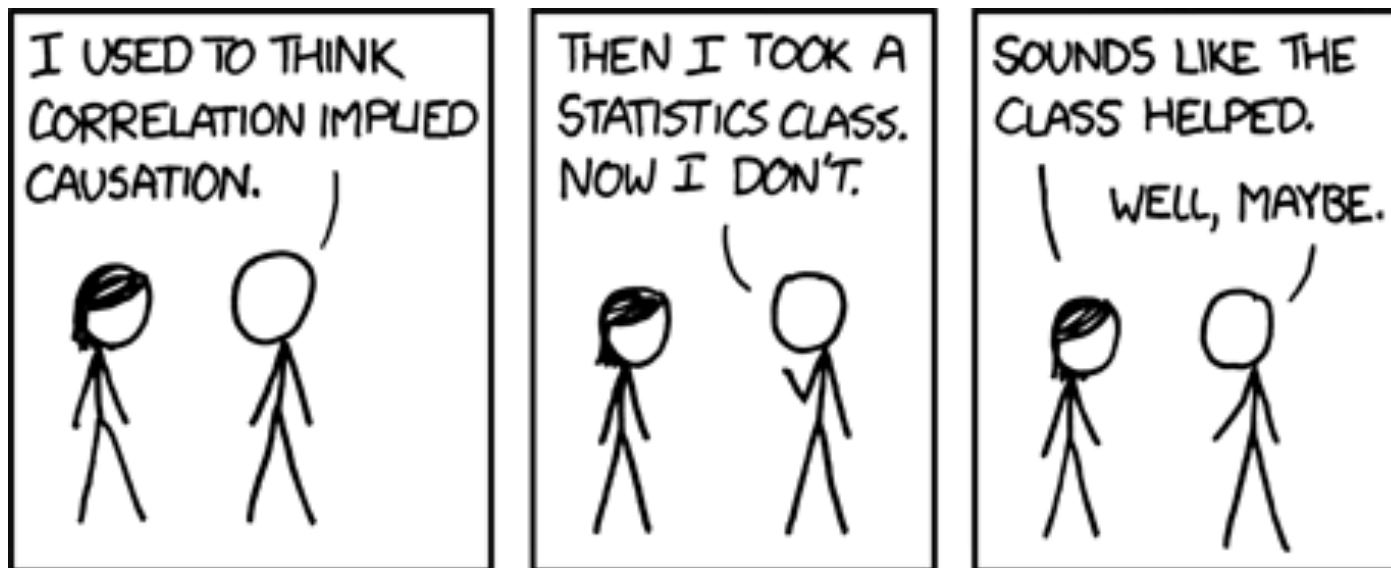
## Understanding Your Data



# For Causation



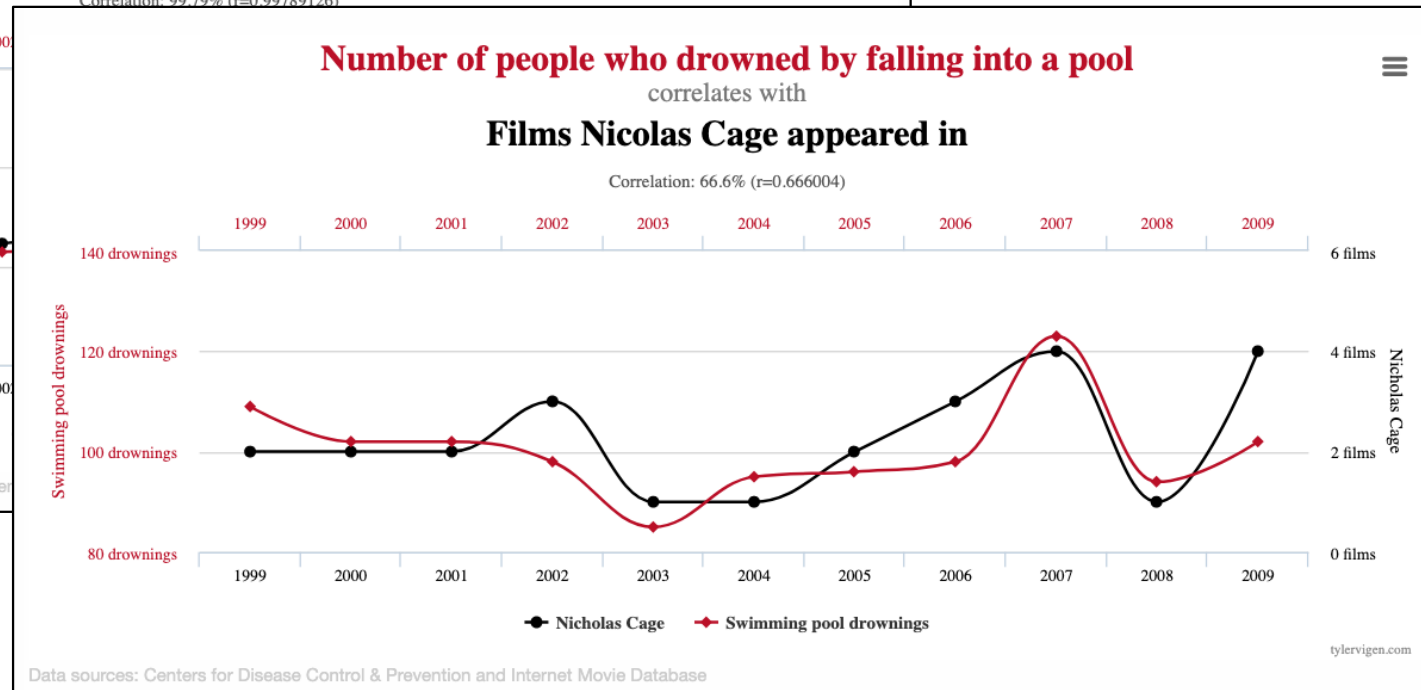
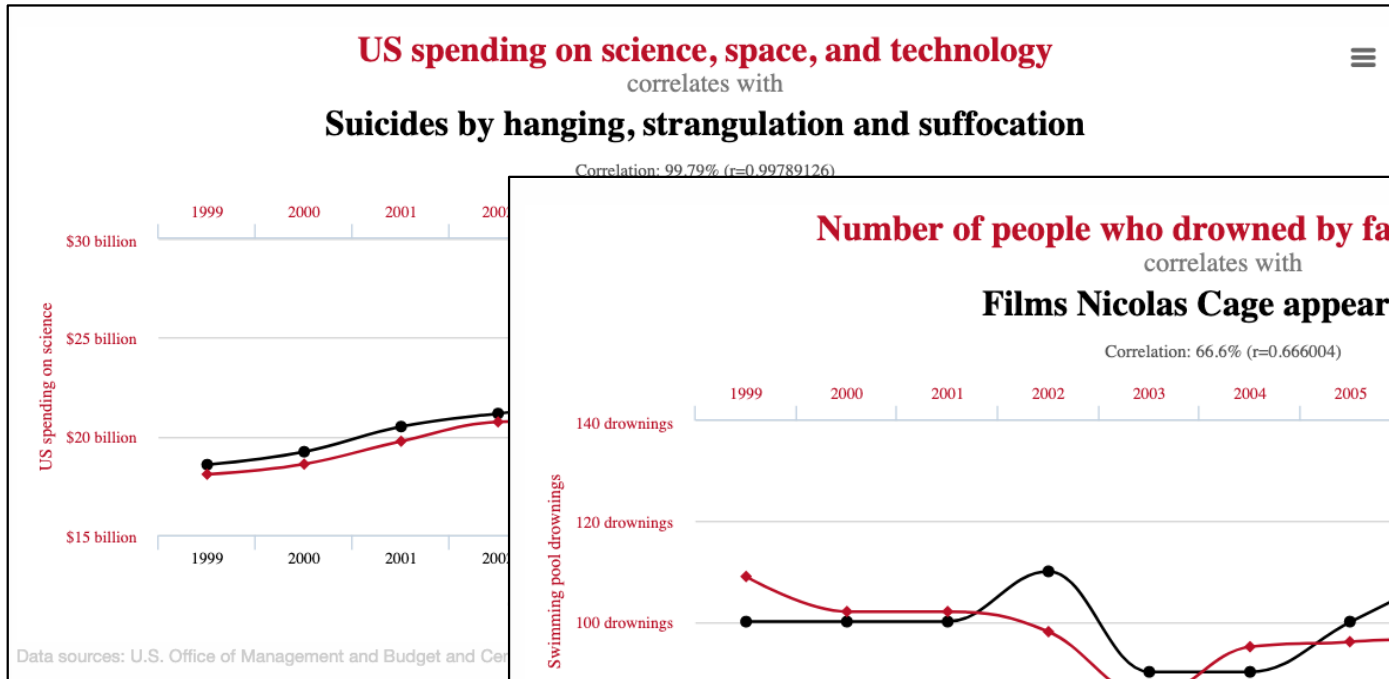
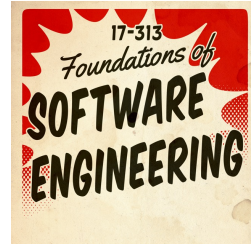
- Provide a theory (from domain knowledge, independent of data)
- Show correlation
- Demonstrate ability to predict new cases (replicate/validate)



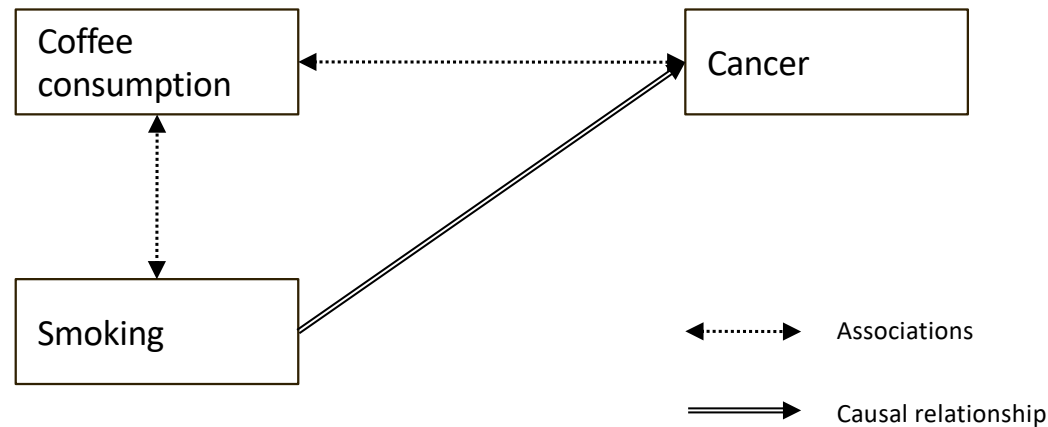
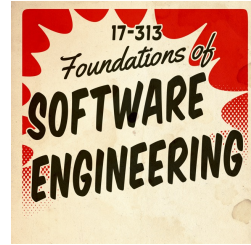
<http://xkcd.com/552/>



# Spurious Correlations



# Confounding variables

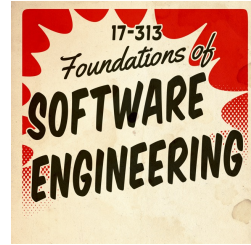


- If you look only at the coffee consumption → cancer relationship, you can get very misleading results
- Smoking is a confounder



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.”



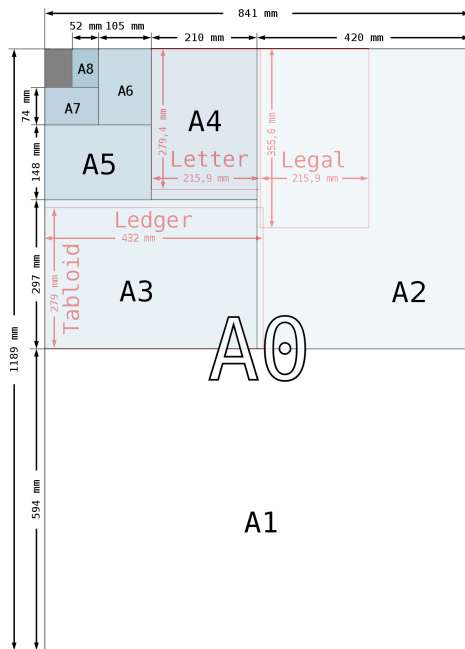
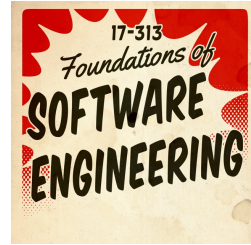
# Measurements validity



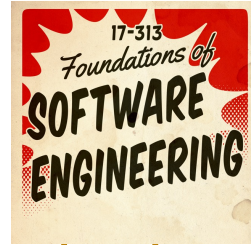
- *Construct validity* – Are we measuring what we intended to measure?
- *Internal validity* – The extent to which the measurement can be used to explain some other characteristic of the entity being measured
- *External validity* – Concerns the generalization of the findings to contexts and environments,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



# Measurements reliability



# Measurements 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



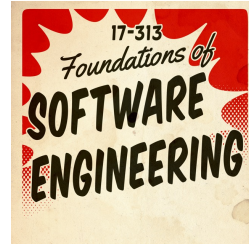


# The McNamara Fallacy



# The 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/>



# The McNamara Fallacy

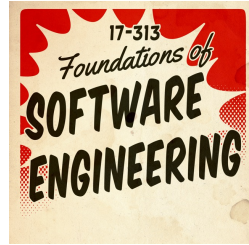


There seems to be a general misunderstanding to the effect that a mathematical model cannot be undertaken until every constant and functional relationship is known to high accuracy. This often leads to the omission of admittedly highly significant factors (most of the “intangibles” influences on decisions) because these are unmeasured or unmeasurable. To omit such variables is equivalent to saying that they have zero effect... Probably the only value known to be wrong...

J. W. Forrester, *Industrial Dynamics*, The MIT Press, 1961



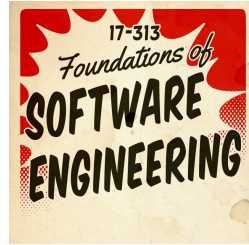
# Defect Density



- Defect density = Known bugs / line of code
- System spoilage = time to fix post-release defects / total system development time
- Post-release vs pre-release
- What counted as defect? Severity? Relevance?
- What size metric used?
- What quality assurance mechanisms used?
- Little reference data publicly available; typically 2-10 defects/1000 lines of code



# Example: Measuring usability.





- Automated measures on code repositories
- Use or collect process data
- Instrument program (e.g., in-field crash reports)
- Surveys, interviews, controlled experiments, expert judgment
- Statistical analysis of sample



# Poll Everywhere Time!

Join by Web [PollEv.com/potantin](https://PollEv.com/potantin) Join by Text Send **potantin** to **22333**



**Is Test Coverage Sufficient as Metric for Quality of your Unit Test Suite?**  0

Yes **(A)**

No **(B)**

I don't know **(C)**

Can you repeat the question? **(D)**







Edit the detailed description

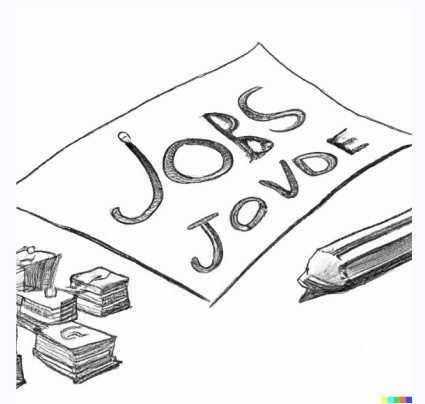
Surprise me

Upload



pencil drawing of job incentives with no words drawn

Generate

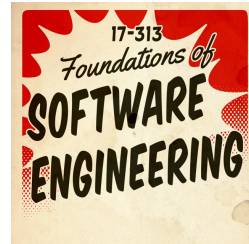


## Metrics and Incentives



# Goodhart's Law

“When a measure becomes a target, it ceases to be a good measure.”



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



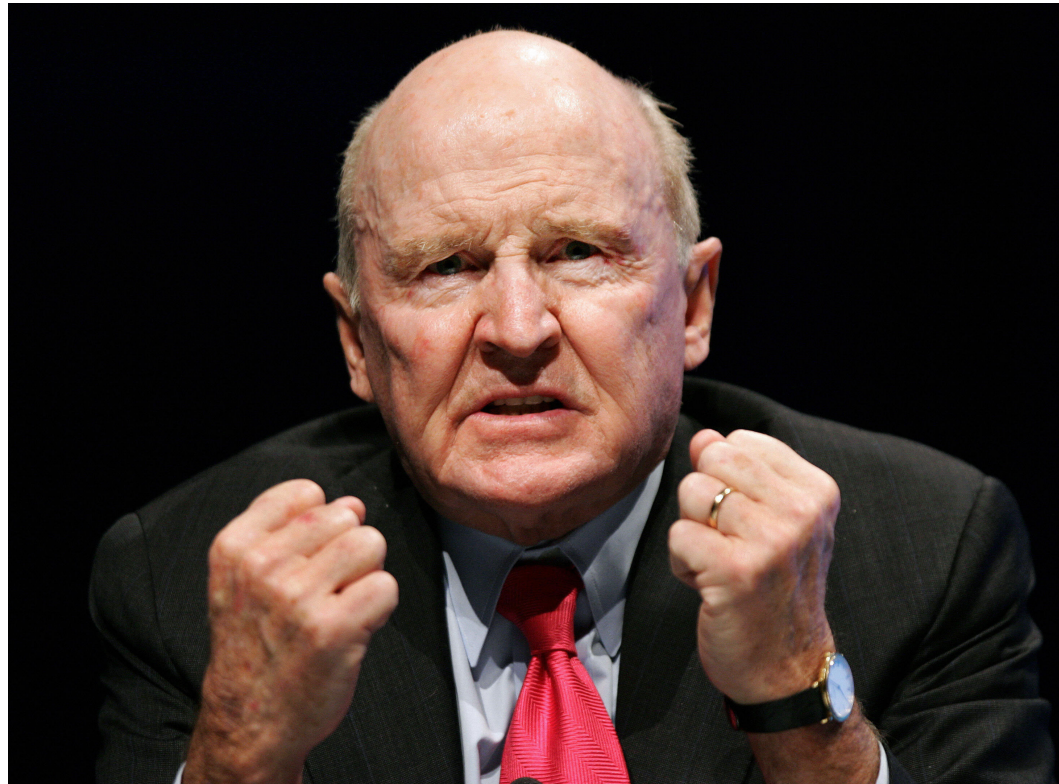
# Productivity Metrics



- Lines of code per day?
  - Industry average 10-50 lines/day
  - Debugging + rework ca. 50% of time
- Function/object/application points per month
- Bugs fixed?
- Milestones reached?



# Stack Ranking

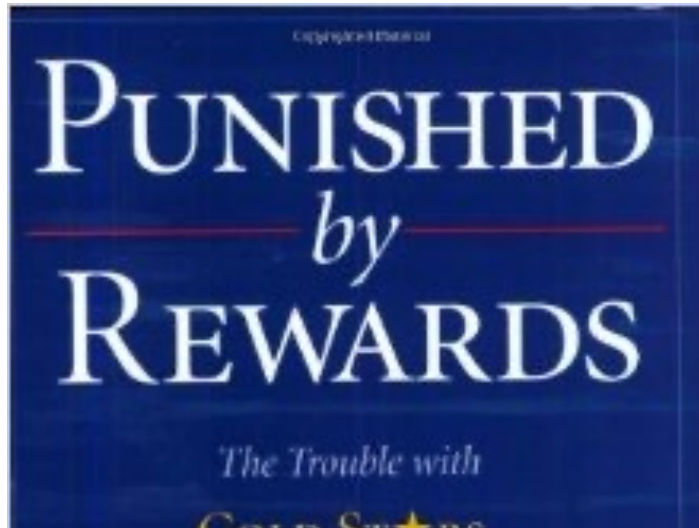


# Incentivizing Productivity

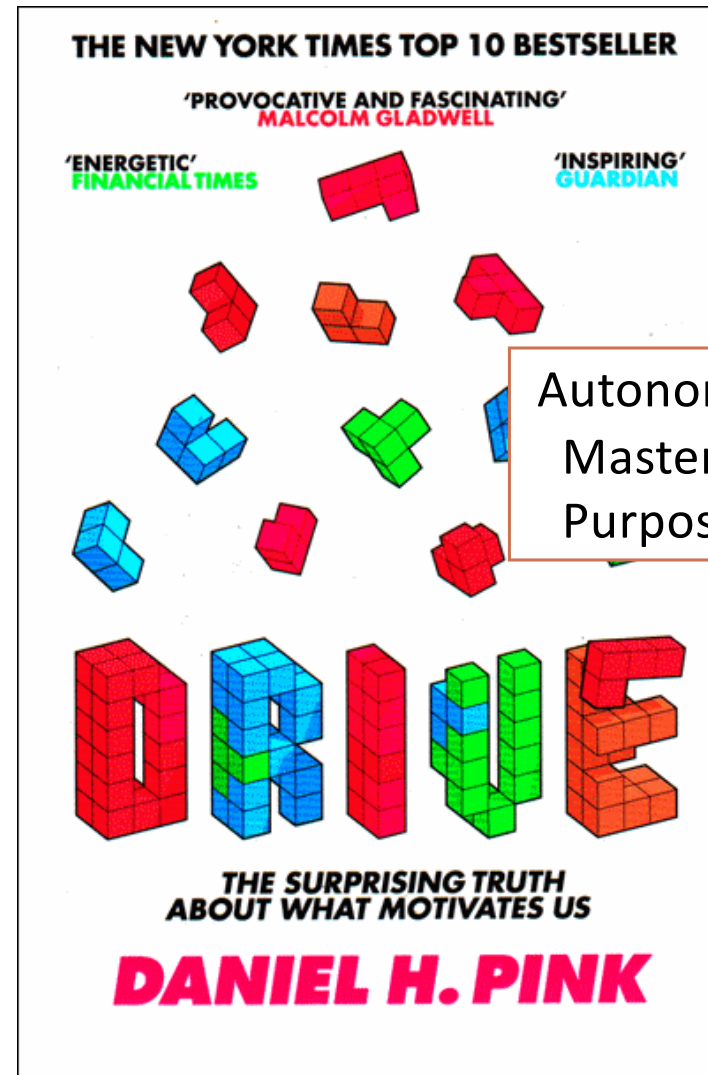


- What happens when developer bonuses are based on
  - Lines of code per day?
  - Amount of documentation written?
  - Low number of reported bugs in their code?
  - Low number of open bugs in their code?
  - High number of fixed bugs?
  - Accuracy of time estimates?





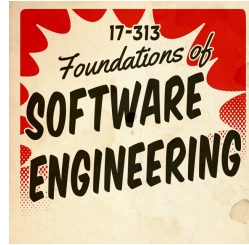
- Can extinguish intrinsic motivation
- Can diminish performance
- Can crush creativity
- Can crowd out good behavior
- Can encourage cheating, shortcuts, and unethical behavior
- Can become addictive
- Can foster short-term thinking



Autonomy  
Mastery  
Purpose



# Warning

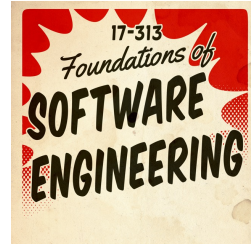


- Most software metrics are controversial
  - Usually only plausibility arguments, rarely rigorously validated
  - Cyclomatic complexity was repeatedly refuted and is still used
  - “Similar to the attempt of measuring the intelligence of a person in terms of the weight or circumference of the brain”
- Use carefully!
- Code size dominates many metrics
- Avoid claims about human factors (e.g., readability) and quality, unless validated
- Calibrate metrics in project history and other projects
- Metrics can be gamed; you get what you measure





# (Some) strategies

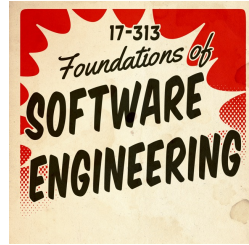


- Metrics tracked using tools and processes (process metrics like time, or code metrics like defects in a bug database).
- Expert assessment or human-subject experiments (controlled experiments, talk-aloud protocols).
- Mining software repositories, defect databases, especially for trend analysis or defect prediction.
  - Some success e.g., as reported by Microsoft Research
- Benchmarking (especially for performance).



# Factors in a successful measurement program

- 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.

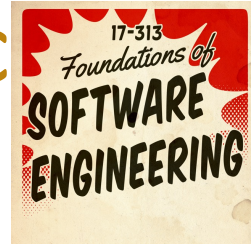


Carol A. Dekkers and Patricia A. McQuaid,  
“The Dangers of Using Software Metrics to  
(Mis)Manage”, 2002.





# Kaner's questions when choosing a metric

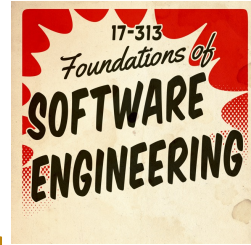


1. What is the purpose of this measure?
2. What is the scope of this measure?
3. What attribute are you trying to measure?
4. What is the attribute's natural scale?
5. What is the attribute's natural variability?
6. What instrument are you using to measure the attribute, and what reading do you take from the instrument?
7. What is the instrument's natural scale?
8. What is the reading's natural variability (normally called measurement error)?
9. What is the attribute's relationship to the instrument?
10. 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



# Further Reading on Metrics




- Sommerville. Software Engineering. Edition 7/8, Sections 26.1, 27.5, and 28.3
- Hubbard. How to measure anything: Finding the value of intangibles in business. John Wiley & Sons, 2014. Chapter 3
- Kaner and Bond. Software Engineering Metrics: What Do They Measure and How Do We Know? METRICS 2004
- Fenton and Pfleeger. Software Metrics: A rigorous & practical approach. Thomson Publishing 1997



# Poll Everywhere Time!

Join by Web [PollEv.com/potantin](https://PollEv.com/potantin) Join by Text Send **potantin** to **22333**



## Are Grades in a Course a Stack Ranking?

0

Yes **(A)**

No **(B)**

I don't know **(C)**

Can you repeat the question? **(D)**



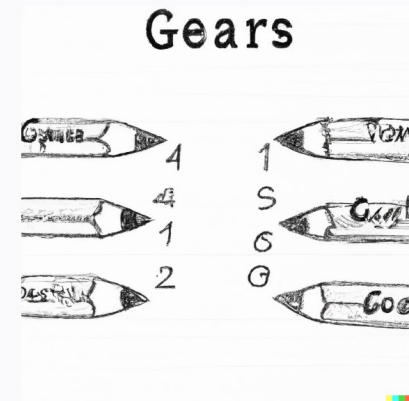
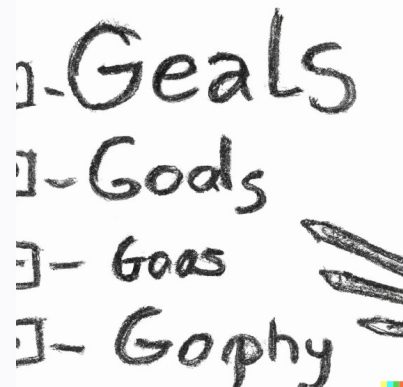
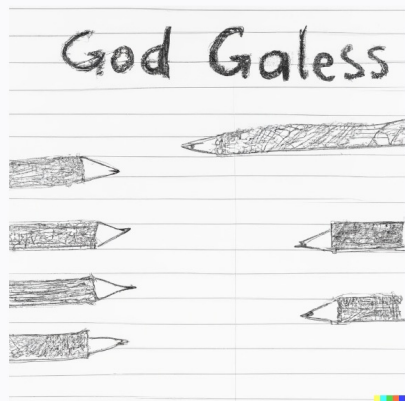


Edit the detailed description

Surprise me Upload →

pencil drawing of goals, signals, and metrics with no words

Generate



## Goals, Signals, Metrics

# Notes on Measuring Engineering Productivity

- Collecting and analysing data on the *human side of things*
- As organisations grow in size *linearly*, communication costs grow *quadratically* (see The Mythical Man-Month or even Amdahl's Law in Computer Architecture 😊)
- Could try to make each individual more productive?
- How to *measure* individual productivity and identify inefficiencies without taking up too many resources?
- Google has a team of researchers dedicated to *engineering productivity*



# Notes on Measuring Engineering Productivity

- Building on *social sciences*, allows to study human side like personal motivations, incentives, and strategies for complex tasks
- What *should* we measure?
- *How* to use metrics to track improvements and productivity?
- Case Study around the process of C++ and Java language teams around *Code Readability*
- *Is the time spent on the readability process worthwhile?*



# Notes on Measuring Engineering Productivity

- Is It Even Worth Measuring?
- Triage Questions:
  1. What result are you expecting, and why?
  2. If the data supports your expected result, what action will be taken?
  3. If we get a negative result, will appropriate action be taken?
  4. Who is going to decide to take action on the result, and when would they do it?
- Reasons NOT to measure can be:
  - You can't afford to change the process/tools right now
  - Any results will soon be invalidated by other factors
  - The results will be used only as vanity metrics to support something you were going to do anyway
  - The only metrics available are not precise enough to measure the problem and can be confounded by other factors





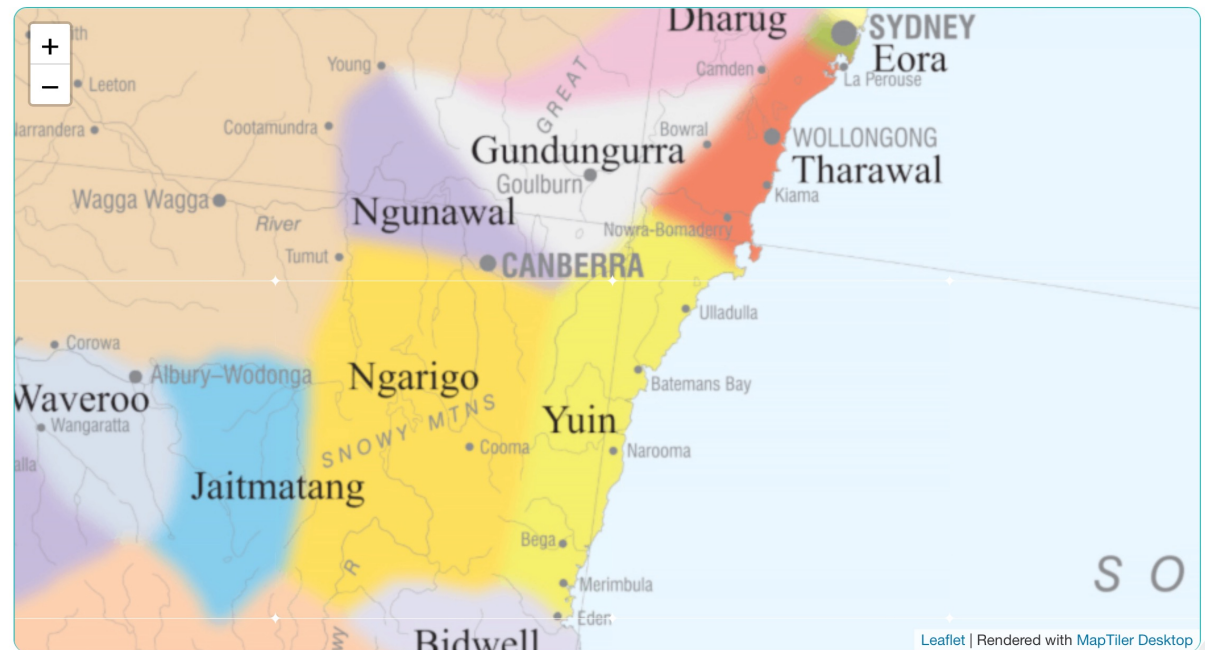
Week 4 Lecture Ended Here, so Resumed in Week 5



# ANU Acknowledgment of Country



“We acknowledge and celebrate the First Australians on whose traditional lands we meet, and pay our respect to the elders past and present.”



<https://aiatsis.gov.au/explore/map-indigenous-australia>



# Notes on Measuring Engineering Productivity

- At Google they use Goals/Signals/Metrics (GSM) framework to guide metrics creation:
  - A *goal* is a desired end result. It's phrased in terms of what you want to understand at a high level and should not contain references to specific ways to measure it.
  - A signal is how you might know that you've achieved the end result. Signals are things we would *like* to measure, but they might not be measurable themselves.
  - A *metric* is a proxy for a signal. It is the thing we actually can measure. It might not be the ideal measurement, but it is something that we believe is close enough.
- GSM encourages us to select metrics based on their ability to measure the original goals



# Goals (Capturing Productivity Trade Offs)

## *Quality of the code*

What is the quality of the code produced? Are the test cases good enough to prevent regressions? How good is an architecture at mitigating risk and changes?

## *Attention from engineers*

How frequently do engineers reach a state of flow? How much are they distracted by notifications? Does a tool encourage engineers to context switch?

## *Intellectual complexity*

How much cognitive load is required to complete a task? What is the inherent complexity of the problem being solved? Do engineers need to deal with unnecessary complexity?

## *Tempo and velocity*

How quickly can engineers accomplish their tasks? How fast can they push their releases out? How many tasks do they complete in a given timeframe?

## *Satisfaction*

How happy are engineers with their tools? How well does a tool meet engineers' needs? How satisfied are they with their work and their end product? Are engineers feeling burned out?



# Goals (Readability Case Study)

## *Quality of the code*

Engineers write higher-quality code as a result of the readability process; they write more consistent code as a result of the readability process; and they contribute to a culture of code health as a result of the readability process.

## *Attention from engineers*

We did not have any attention goal for readability. This is OK! Not all questions about engineering productivity involve trade-offs in all five areas.

## *Intellectual complexity*

Engineers learn about the Google codebase and best coding practices as a result of the readability process, and they receive mentoring during the readability process.

## *Tempo and velocity*

Engineers complete work tasks faster and more efficiently as a result of the readability process.

## *Satisfaction*

Engineers see the benefit of the readability process and have positive feelings about participating in it.



# Signals (Readability Case Study)

*Table 7-1. Signals and goals*

<b>Goals</b>	<b>Signals</b>
Engineers write higher-quality code as a result of the readability process.	Engineers who have been granted readability judge their code to be of higher quality than engineers who have not been granted readability. The readability process has a positive impact on code quality.
Engineers learn about the Google codebase and best coding practices as a result of the readability process.	Engineers report learning from the readability process.
Engineers receive mentoring during the readability process.	Engineers report positive interactions with experienced Google engineers who serve as reviewers during the readability process.
Engineers complete work tasks faster and more efficiently as a result of the readability process.	Engineers who have been granted readability judge themselves to be more productive than engineers who have not been granted readability. Changes written by engineers who have been granted readability are faster to review than changes written by engineers who have not been granted readability.
Engineers see the benefit of the readability process and have positive feelings about participating in it.	Engineers view the readability process as being worthwhile.



# Metrics (Readability Case Study)

<b>QUANTS</b>	<b>Goal</b>	<b>Signal</b>	<b>Metric</b>
Quality of the code	Engineers write higher-quality code as a result of the readability process.	Engineers who have been granted readability judge their code to be of higher quality than engineers who have not been granted readability.	Quarterly Survey: Proportion of engineers who report being satisfied with the quality of their own code
		The readability process has a positive impact on code quality.	Readability Survey: Proportion of engineers reporting that readability reviews have no impact or negative impact on code quality



# Metrics (Readability Case Study)

QUANTS	Goal	Signal	Metric
			Readability Survey: Proportion of engineers reporting that participating in the readability process has improved code quality for their team
	Engineers write more consistent code as a result of the readability process.	Engineers are given consistent feedback and direction in code reviews by readability reviewers as a part of the readability process.	Readability Survey: Proportion of engineers reporting inconsistency in readability reviewers' comments and readability criteria.
	Engineers contribute to a culture of code health as a result of the readability process.	Engineers who have been granted readability regularly comment on style and/or readability issues in code reviews.	Readability Survey: Proportion of engineers reporting that they regularly comment on style and/or readability issues in code reviews
Attention from engineers	n/a	n/a	n/a





# Metrics (Readability Case Study)

<b>Intellectual</b>	<b>Engineers learn about the Google codebase and best coding practices as a result of the readability process.</b>	<b>Engineers report learning from the readability process.</b>	<b>Readability Survey: Proportion of engineers reporting that they learned about four relevant topics</b>
			<b>Readability Survey: Proportion of engineers reporting that learning or gaining expertise was a strength of the readability process</b>
	<b>Engineers receive mentoring during the readability process.</b>	<b>Engineers report positive interactions with experienced Google engineers who serve as reviewers during the readability process.</b>	<b>Readability Survey: Proportion of engineers reporting that working with readability reviewers was a strength of the readability process</b>



# Metrics (Readability Case Study)

Tempo/velocity	Engineers are more productive as a result of the readability process.	Engineers who have been granted readability judge themselves to be more productive than engineers who have not been granted readability.	Quarterly Survey: Proportion of engineers reporting that they're highly productive
		Engineers report that completing the readability process positively affects their engineering velocity.	Readability Survey: Proportion of engineers reporting that <i>not</i> having readability reduces team engineering velocity
		Changelists (CLs) written by engineers who have been granted readability are faster to review than CLs written by engineers who have not been granted readability.	Logs data: Median review time for CLs from authors with readability and without readability



# Metrics (Readability Case Study)

QUANTS	Goal	Signal	Metric
		CLs written by engineers who have been granted readability are easier to shepherd through code review than CLs written by engineers who have not been granted readability.	Logs data: Median shepherding time for CLs from authors with readability and without readability
		CLs written by engineers who have been granted readability are faster to get through code review than CLs written by engineers who have not been granted readability.	Logs data: Median time to submit for CLs from authors with readability and without readability
		The readability process does not have a negative impact on engineering velocity.	Readability Survey: Proportion of engineers reporting that the readability process negatively impacts their velocity
			Readability Survey: Proportion of engineers reporting that readability reviewers responded promptly
			Readability Survey: Proportion of engineers reporting that timeliness of reviews was a strength of the readability process



# Metrics (Readability Case Study)

Satisfaction	Engineers see the benefit of the readability process and have positive feelings about participating in it.	Engineers view the readability process as being an overall positive experience.	Readability Survey: Proportion of engineers reporting that their experience with the readability process was positive overall
		Engineers view the readability process as being worthwhile	Readability Survey: Proportion of engineers reporting that the readability process is worthwhile
			Readability Survey: Proportion of engineers reporting that the quality of readability reviews is a strength of the process
			Readability Survey: Proportion of engineers reporting that thoroughness is a strength of the process
		Engineers do not view the readability process as frustrating.	Readability Survey: Proportion of engineers reporting that the readability process is uncertain, unclear, slow, or frustrating

Quarterly Survey: Proportion of engineers reporting that they're satisfied with their own engineering velocity




# Case Study on Readability Outcome


- Study showed that it was overall worthwhile:
  - Engineers who had achieved readability were satisfied with the process and felt they learned from it
  - Logs showed that they also had their code reviewed faster and submitted it faster, even accounting for no longer needing as many reviewers
  - Study also showed places for improvement with the process: engineers identified pain points
- The language teams improved the tooling and process based on the results



# Poll Everywhere Time!

When poll is active respond at [PollEv.com/potantin](https://PollEv.com/potantin) Send **potantin** to **22333**



**Should I keep using Monty Python Video Dividers in the Lectures?**  0

Yes

No

