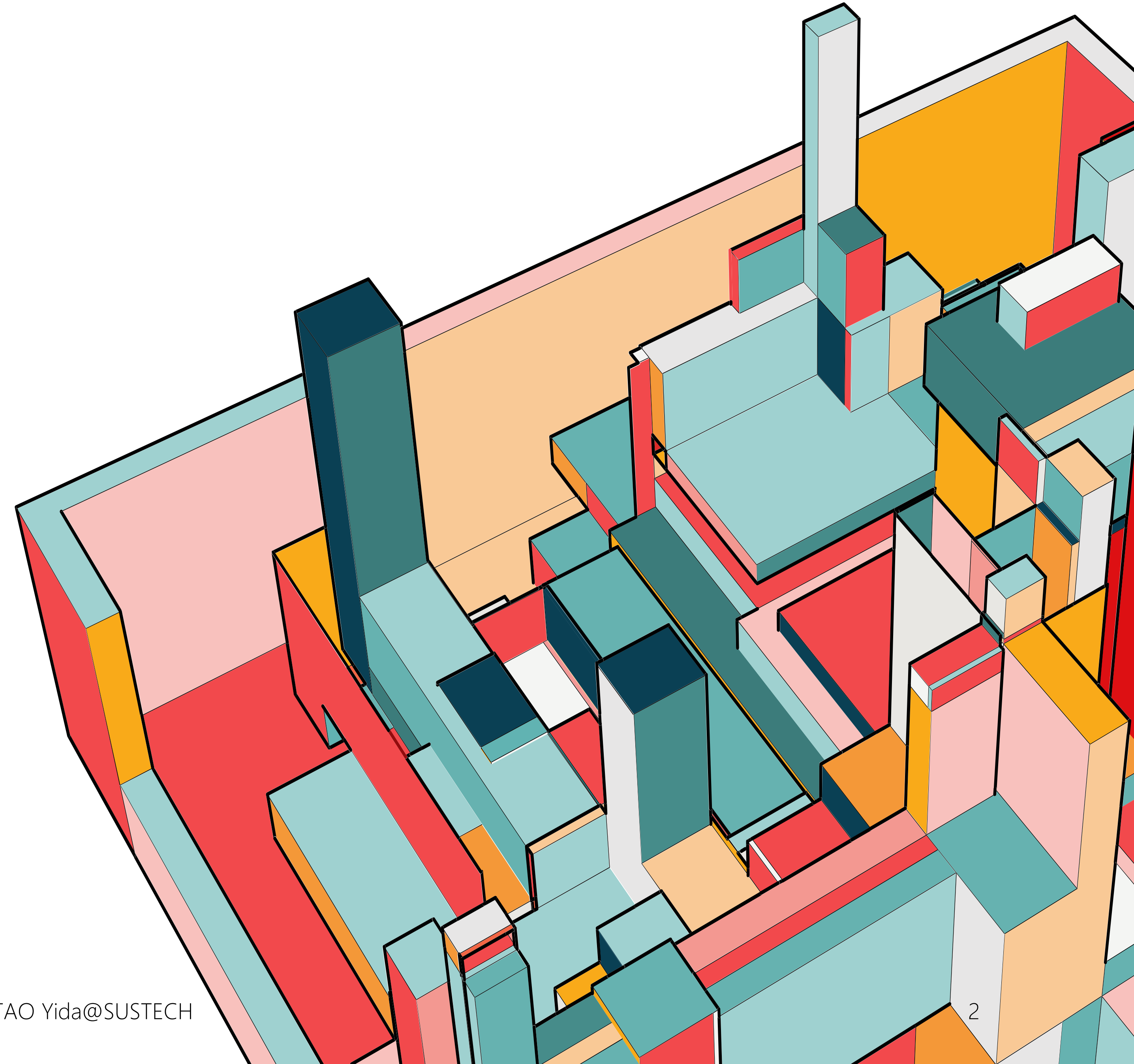
CS304 SOFTWARE ENGINEERING

Yida Tao

taoyd@sustech.edu.cn

LECTURE 8

- Measurement and Metrics

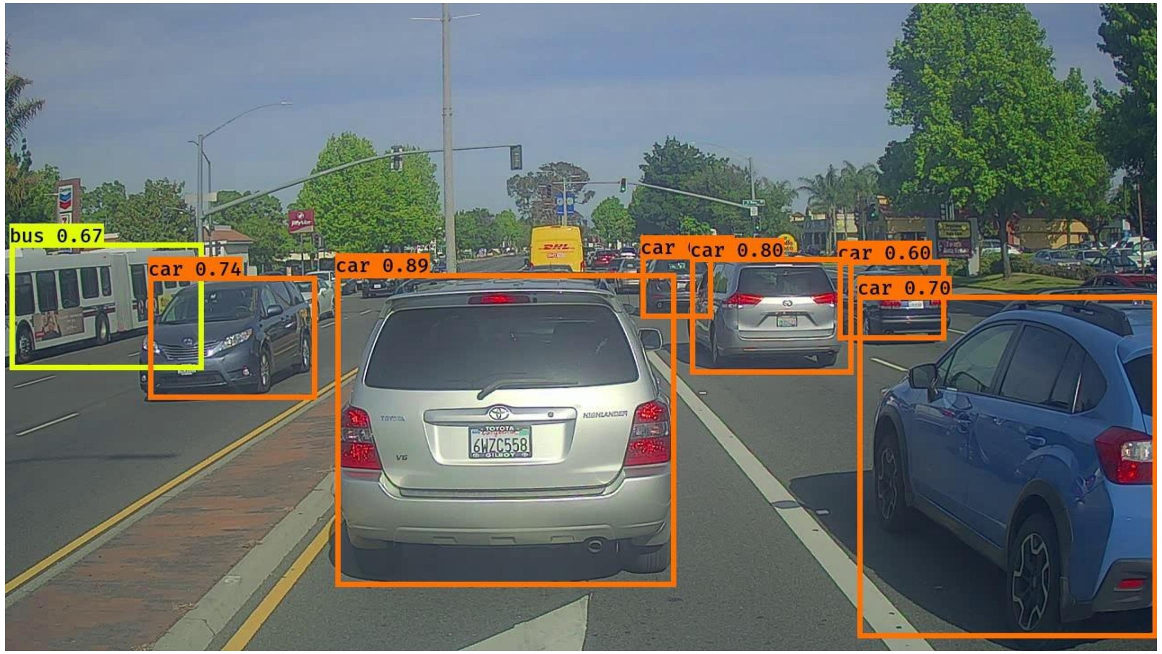


HOW DO WE JUDGE THE SOFTWARE QUALITY OF AUTONOMOUS VEHICLES ?

Case study adapted from https://cmu-313.github.io/assets/pdfs/02-measurement.pdf

MODEL ACCURACY

- Train machine learning models on labelled data
- Compute accuracy on a separate labelled test set
- For example, 90% accuracy implies that object recognition is correct for 90% of the test inputs

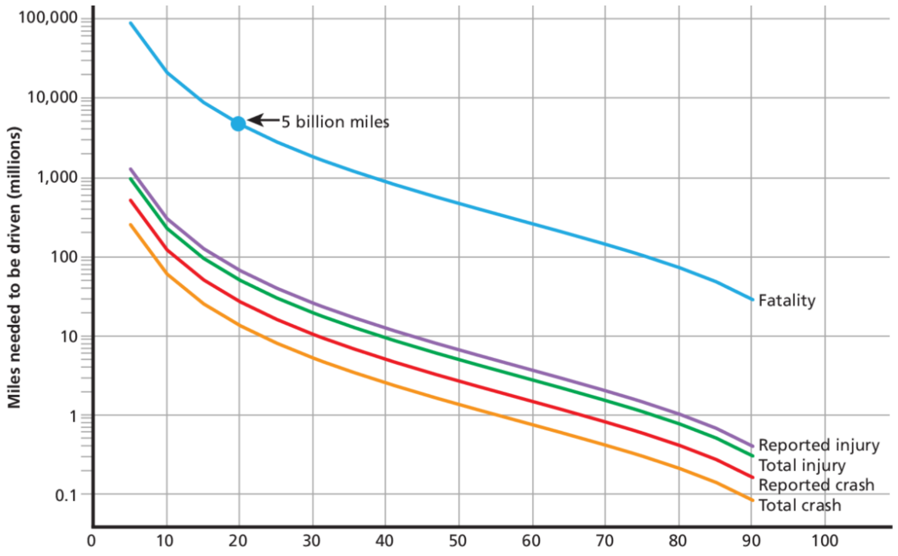


https://miro.medium.com/max/1400/1*5gzTV45WMA71QSZYelq9JQ.jpeg

FAILURE RATE

- Frequency of crashes / fatalities
- Failure rate could be measured per 1000 rides, per million miles, per month, etc.

MILEAGE



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

https://waymo.com/intl/zh-cn/safety/

MEASUREMENTS AND METRICS

- Software measurement **quantifies** the characteristics of a software product or the software process.
- Software measurement process includes
  - **Formulation**: The **derivation** of software measures and metrics appropriate for the representation of the software that is being considered.
  - **Collection**: The mechanism used to **accumulate** data required to derive the formulated metrics.
  - **Analysis**: The **computation** of metrics and the application of mathematical tools.
  - **Interpretation**: The evaluation of metrics resulting in **insight** into the quality of the representation.
  - **Feedback**: **Recommendation** derived from the interpretation of product metrics transmitted to the software team.

https://www.geeksforgeeks.org/software-measurement-and-metrics/

SOFTWARE QUALITY METRICS

- Software quality metrics are a subset of software metrics that focus on the quality aspects of the product, process, and project.

- Metrics have been proposed for many quality attributes

- We may also define own metrics

*Even if a metric is not a measurement (measurement is the act of collecting data, while metrics are derived from one or more measures to track a process or assess a particular goal), often the two terms are used as synonyms (i.e., 软件度量).

WHAT SOFTWARE QUALITIES DO WE CARE ABOUT?

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

https://cmu-313.github.io/assets/pdfs/02-measurement.pdf

WHAT PROCESS QUALITIES DO WE CARE ABOUT?

- 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…

https://cmu-313.github.io/assets/pdfs/02-measurement.pdf

TYPES OF METRICS

Product Metrics

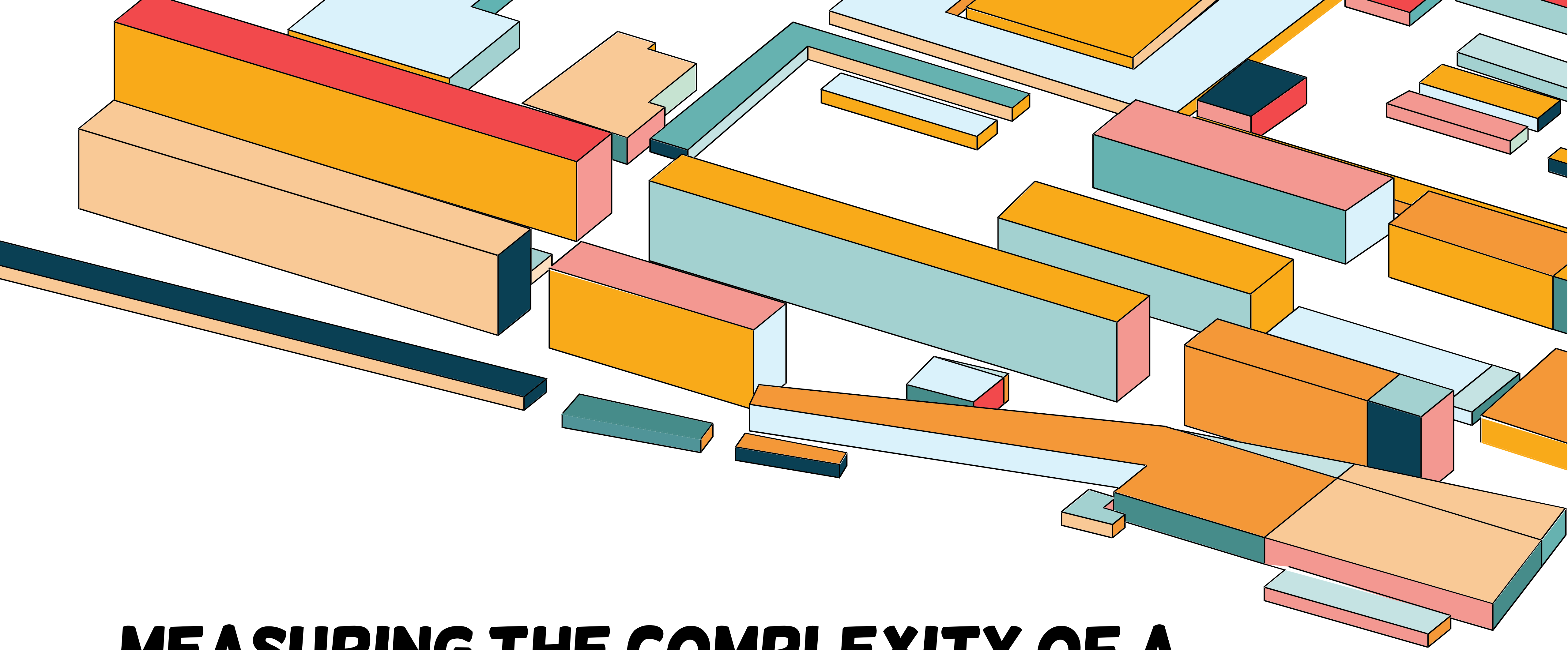
Describe the characteristics of the product/software, e.g., size, complexity, performance, etc.

Process Metrics

Used for describing and improving software development & maintenance process, e.g., # bugs found, bug fixing time, # features added

Project Metrics

Describe the project characteristics and execution, e.g., # of developers, money spent, time spent



MEASURING THE COMPLEXITY OF A SYSTEM

LINES OF CODE

- LoC is easy to measure (`> wc -l command`)
- All software products produce LoC

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)

Potential problems with the LoC metric?

LINES OF CODE - NORMALIZING

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

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

- LoC needs to be **normalized** to be meaningful and comparable

- Ignore comments and empty lines
- Count statement (logical lines) instead of textual lines
- See cLoc (https://github.com/AlDanial/cloc)

LINES OF CODE - LANGUAGE MATTERS

Higher-level languages are more expressive than lower-level languages. Each line of code says more.

- Assembly code may be 2-3X longer than C code
- C code may be 2-3X longer than Java code
- Java code may be 2-3X longer than …

LINES OF CODE - LANGUAGE MATTERS

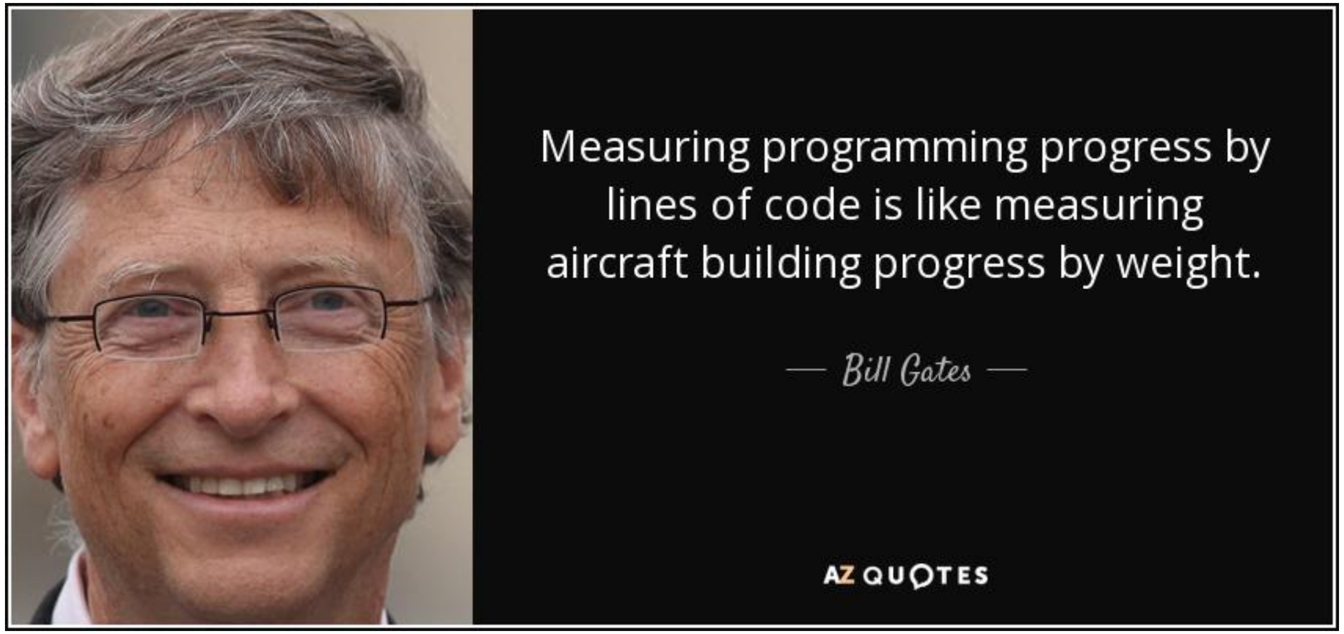
- The table shows typical ratios of source statements in several high-level languages to the equivalent code in C.

- A higher ratio means that each line of code in the language listed accomplishes what x lines of code in C accomplish.

https://blog.codinghorror.com/are-all-programming-languages-the-same/

Language	Level Relative to C
C	1
C++	2.5
Fortran	2
Java	2.5
Perl	6
Python	6
Smalltalk	6
MS Visual Basic	4.5

LINES OF CODE



LoC is a valid metric when

- Use within the same programming language
- Code measured use standard, consistent formatting
- Code has been reviewed first

Cyclomatic Complexity

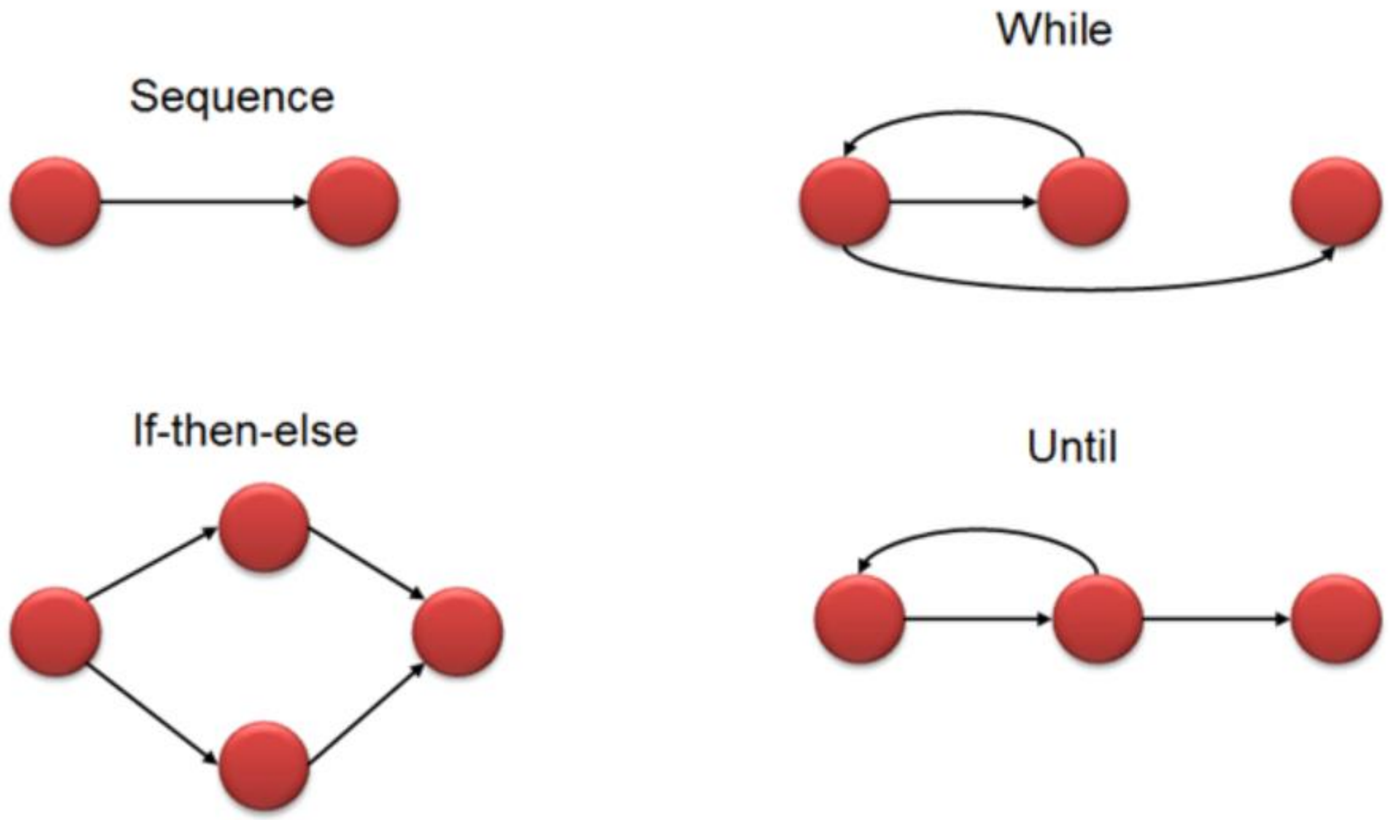
- Cyclomatic complexity (圈复杂度) is a software metric used to indicate the logic complexity of a program.
- It is a quantitative measure of the number of linearly independent paths through a program's source code.
- It was developed by Thomas J. McCabe, Sr. in 1976.

Core idea: the complexity of code depends on the number of decisions in the code (if, while, for)

Control Flow Graph

- Cyclomatic complexity is computed using the control-flow graph (控制流图) of the program
- The nodes of the control flow graph correspond to indivisible groups of commands of a program (e.g., blocks), and a directed edge connects two nodes if the second command might be executed immediately after the first command.

CONTROL FLOW GRAPH



https://www.guru99.com/cyclomatic-complexity.html

Decisions are **cycles**

CALCULATE CYCLOMATIC COMPLEXITY

Approach 1: `V (G) = P + 1`

P: Number of branch nodes (e.g., if, for, while, case)

Approach 2: `V(G) = E - N + 2`

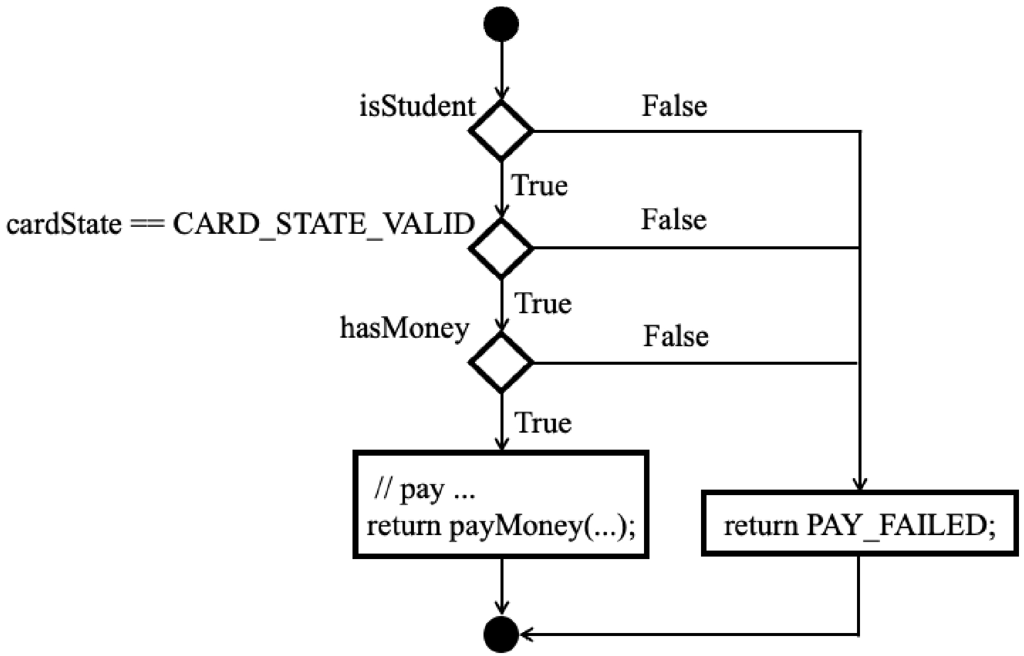
E: Number of edges

N: Number of nodes

https://www.guru99.com/cyclomatic-complexity.html

CALCULATE CYCLOMATIC COMPLEXITY

```
if (isStudent) {
    if (cardState == CARD_STATE_VALID) {
        if (hasMoney) {
            // pay ...
            return payMoney(...);
        }
    }
}
return PAY_FAILED;
```



Cyclomatic complexity = 3 + 1 = 4

CALCULATE CYCLOMATIC COMPLEXITY

Original paper is not clear about how to derive the control flow graph

- Different implementations gives different values for the same code.
- For example, the following code is reported with complexity 2 by the Eclipse Metrics Plugin , with 4 by GMetrics , and with complexity 5 by SonarQube

```
int foo (int a, int b) {
    if (a > 17 && b < 42 && a + b < 55) {
        return 1;
    }
    return 2;
}
```

https://www.cqse.eu/en/news/blog/mccabe-cyclomatic-complexity/

INTERPRETING CYCLOMATIC COMPLEXITY

- Cyclomatic complexity is the number of independent paths through the procedure
- Gives an upper bound on the number of tests necessary to execute every edge of control graph
- So, in the context of testing, cyclomatic complexity can be used to estimate the required effort for writing tests.

https://www.cqse.eu/en/news/blog/mccabe-cyclomatic-complexity/

INTERPRETING CYCLOMATIC COMPLEXITY

Cyclomatic complexity	Code	Testability	Maintenance cost
1-10	Structured and well written	High	Low
10-20	Complex	Medium	Medium
20-40	Very complex	Low	High
>40	Unreadable	Very low	Very high

https://www.guru99.com/cyclomatic-complexity.html

PITFALLS ON CYCLOMATIC COMPLEXITY

High cyclomatic complexity means poor readability?

- The code got the cyclomatic complexity of 14
- Seems readable

```
String getMonthName (int month) {
    switch (month) {
        case 0: return "January";
        case 1: return "February";
        case 2: return "March";
        case 3: return "April";
        case 4: return "May";
        case 5: return "June";
        case 6: return "July";
        case 7: return "August";
        case 8: return "September";
        case 9: return "October";
        case 10: return "November";
        case 11: return "December";
        default:
            throw new IllegalArgumentException();
    }
}
```

https://www.cqse.eu/en/news/blog/mccabe-cyclomatic-complexity/

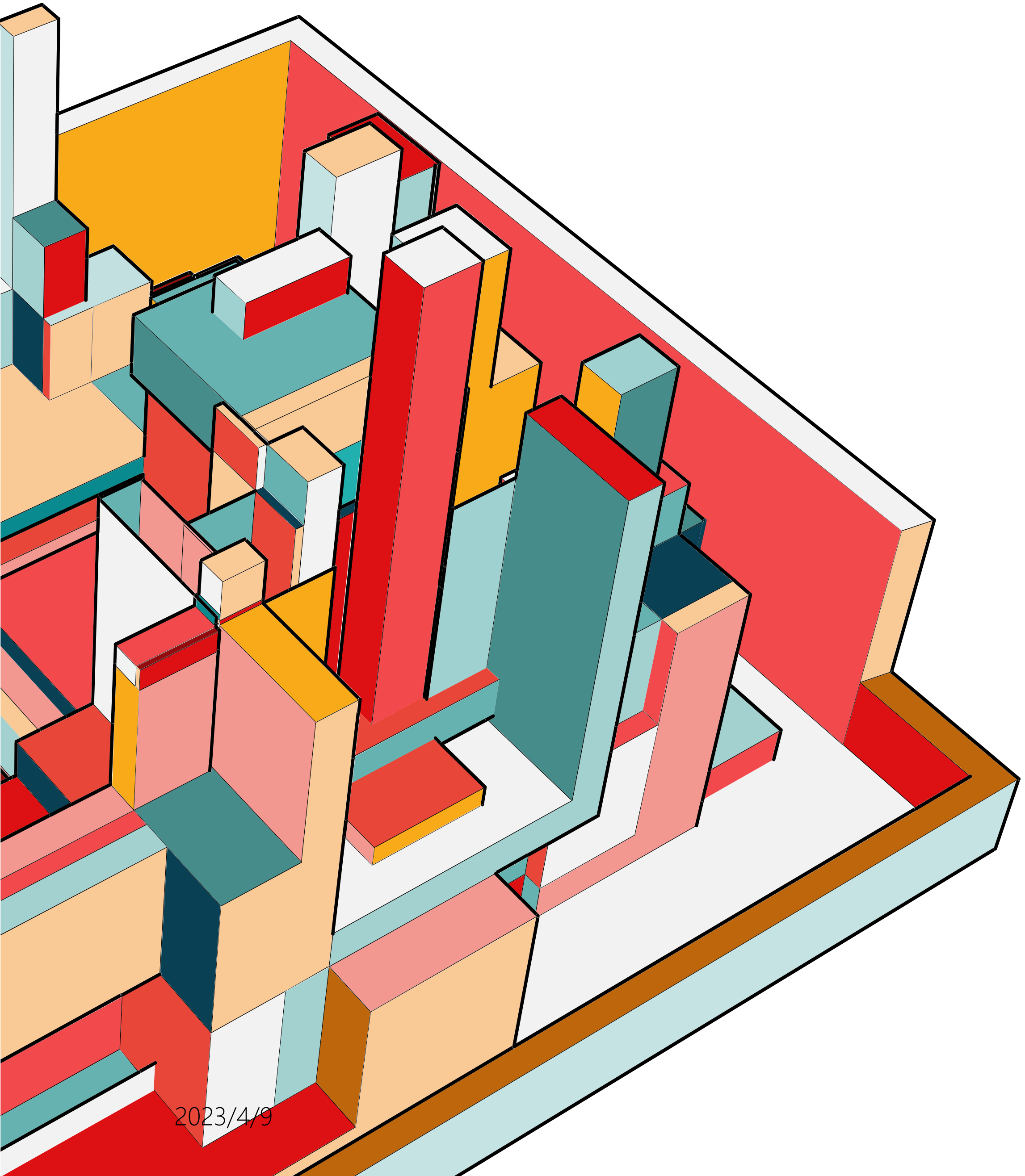
PITFALLS ON CYCLOMATIC COMPLEXITY

```java
String getWeight(int i) {
    if (i <= 0) {
        return "no weight";
    }
    if (i < 10) {
        return "light";
    }
    if (i < 20) {
        return "medium";
    }
    if (i < 30) {
        return "heavy";
    }
    return "very heavy";
}
```

```java
int sumOfNonPrimes(int limit) {
    int sum = 0;
    OUTER: for (int i = 0; i < limit; ++i) {
        if (i <= 2) {
            continue;
        }
        for (int j = 2; j < i; ++j) {
            if (i % j == 0) {
                continue OUTER;
            }
        }
        sum += i;
    }
    return sum;
}
```

Both have a cyclomatic complexity of 5. Same readability & maintainability?

https://www.cqse.eu/en/news/blog/mccabe-cyclomatic-complexity/



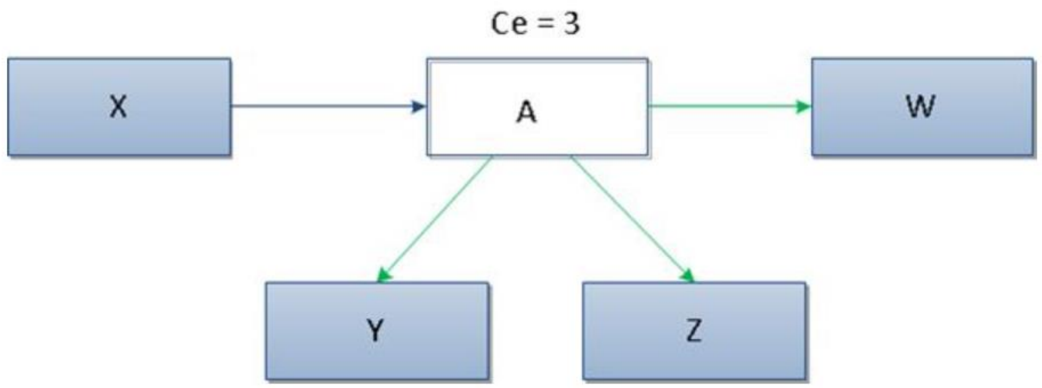
COUPLING AND COHESION

- Dependences
  - Call methods, refer to classes, share variables

- Coupling
  - Dependences among modules (bad)
- Cohesion
  - Dependences within modules (good)

MARTIN'S COUPLING METRICS

- **Efferent coupling (Ce):** A class's efferent couplings is a measure of how many different classes are used by the specific class.
- Ce measures outgoing dependencies (who do you depend on)
- Measures the vulnerability of the class to changes in other classes that it depends on



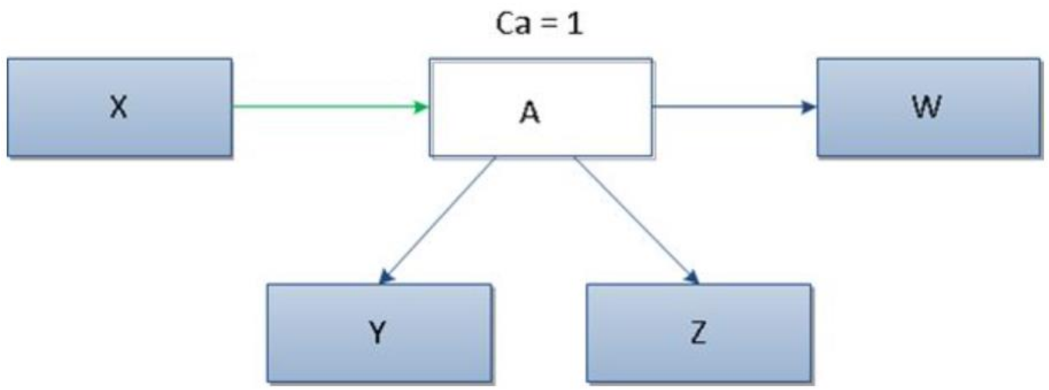
Pic. 1 – Outgoing dependencies

Ce> 20 indicates instability of a package

https://kariera.future-processing.pl/blog/object-oriented-metrics-by-robert-martin/

MARTIN'S COUPLING METRICS

- **Afferent coupling (Ca)**: A class's afferent couplings is a measure of how many other classes use the specific class.
- Ca measures incoming dependencies (who depends on you)
- Measures the sensitivity of remaining classes to changes in the analyzed class



Pic. 2 – Incoming dependencies

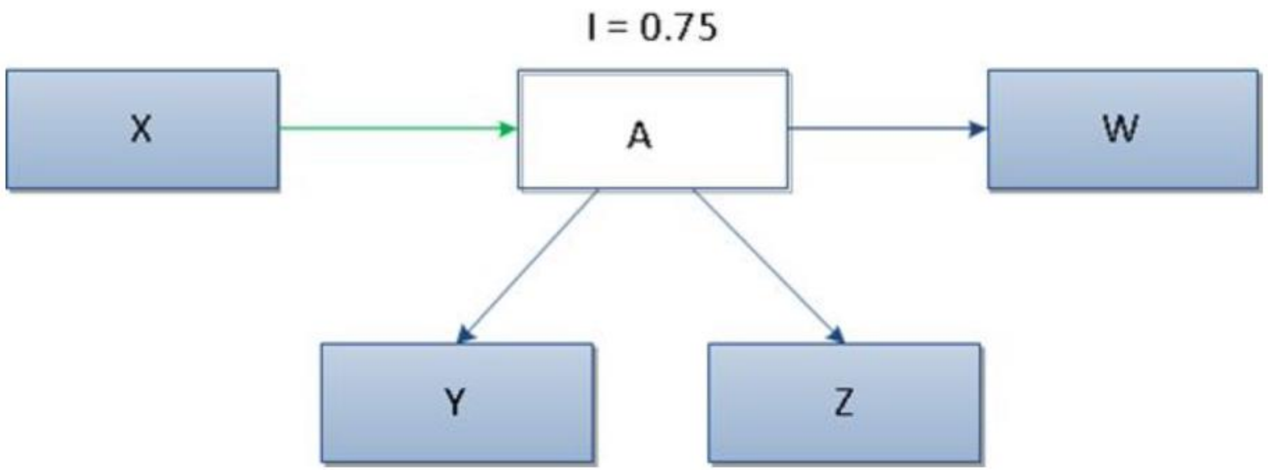
High values of metric Ca usually suggest high component stability

https://kariera.future-processing.pl/blog/object-oriented-metrics-by-robert-martin/

MARTIN'S COUPLING METRICS

- **Instability:** measure the relative susceptibility of class to changes

$$I = \frac{Ce}{Ce + Ca}$$



Pic. 3 – Instability

https://kariera.future-processing.pl/blog/object-oriented-metrics-by-robert-martin/

MARTIN'S COUPLING METRICS

- The ones having many outgoing dependencies and not many of incoming ones (instability is close to 1): rather unstable due to the possibility of easy changes to these packages

- The ones having many incoming dependencies and not many of outgoing ones (instability is close to 0): rather more difficult in modifying due to their greater responsibility.

$$I = \frac{Ce}{Ce + Ca}$$

https://kariera.future-processing.pl/blog/object-oriented-metrics-by-robert-martin/

OO METRICS ckjm – Chidamber and Kemerer Java Metrics

- Weighted Methods per Class (WMC)
- Depth of Inheritance Tree (DIT)
- Number of Children (NOC)
- Coupling between Object Classes (CBO)
- Response for a Class (RFC)
- Lack of Cohesion in Methods (LCOM)
- ……

WEIGHTED METHODS PER CLASS

- This metric is the sum of complexities of methods defined in a class.
- It therefore represents the complexity of a class as a whole

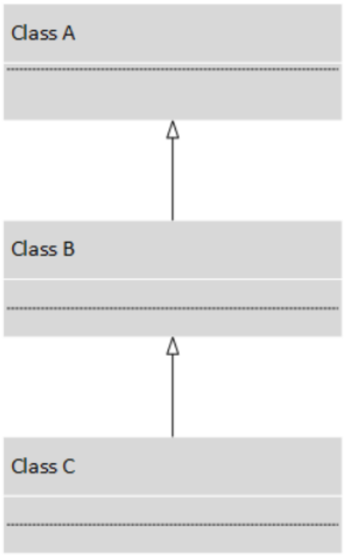
- Possible method complexities
  - 1 (# of methods)
  - LoC
  - # of method calls
  - Cyclomatic complexity

WEIGHTED METHODS PER CLASS

- This measure can be used to indicate the development and maintenance effort for the class.
- The larger the number of methods in a class, the greater the potential impact on children
- Classes with large numbers of methods are more likely to be application specific and less reusable

DEPTH OF INHERITANCE TREE

DIT measures the maximum length between a node and the root node in a class hierarchy



DIT = 0

DIT = 1

DIT = 2

DEPTH OF INHERITANCE TREE

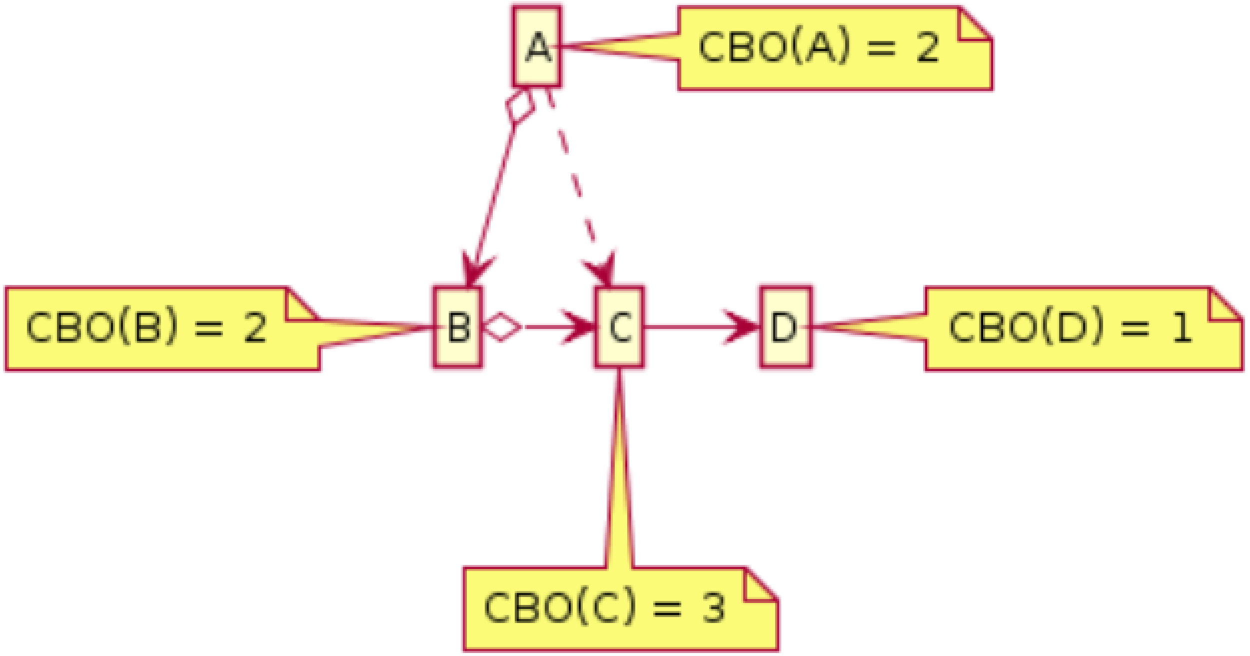
- The deeper a class is in the hierarchy, the more methods it inherits and so it is harder to predict its behavior
- The deeper a class is in the hierarchy, the more methods it reuses
- Deeper trees are more complex

NUMBER OF CHILDREN

- NOC indicates the number of immediate subclasses
- A class with a large NOC is probably very important and needs a lot of testing

COUPLING BETWEEN OBJECT CLASSES

- CBO represents the number of classes coupled to a given class.
- This coupling can occur through method calls, field accesses, inheritance, arguments, return types, and exceptions.
- CBO doesn't care about the direction of a dependency



https://stackoverflow.com/questions/27515541/cbo-coupling-between-object

RESPONSE FOR A CLASS

- RFC measures the number of different methods that can be executed when an object of that class receives a message (when a method is invoked for that object).

- Ideally, we would want to find for each method of the class, the methods that class will call, and repeat this for each called method, calculating what is called the transitive closure of the method's call graph, which can however be both expensive and quite inaccurate.

- In ckjm, we calculate a rough approximation to the response set by simply inspecting method calls within the class's method bodies.

RESPONSE FOR A CLASS

- If a large number of methods can be invoked in response to a message, testing becomes more complicated
- The more methods that can be invoked from a class, the greater the complexity of the class

LACK OF COHESION IN METHODS

- LCOM counts the sets of methods in a class that are **not related** through the sharing of some of the class's fields.

- Considers all pairs of a class's methods:
  - Q: In some pairs, both methods access at least one common field of the class
  - P: In other pairs, the two methods do not share any common field accesses

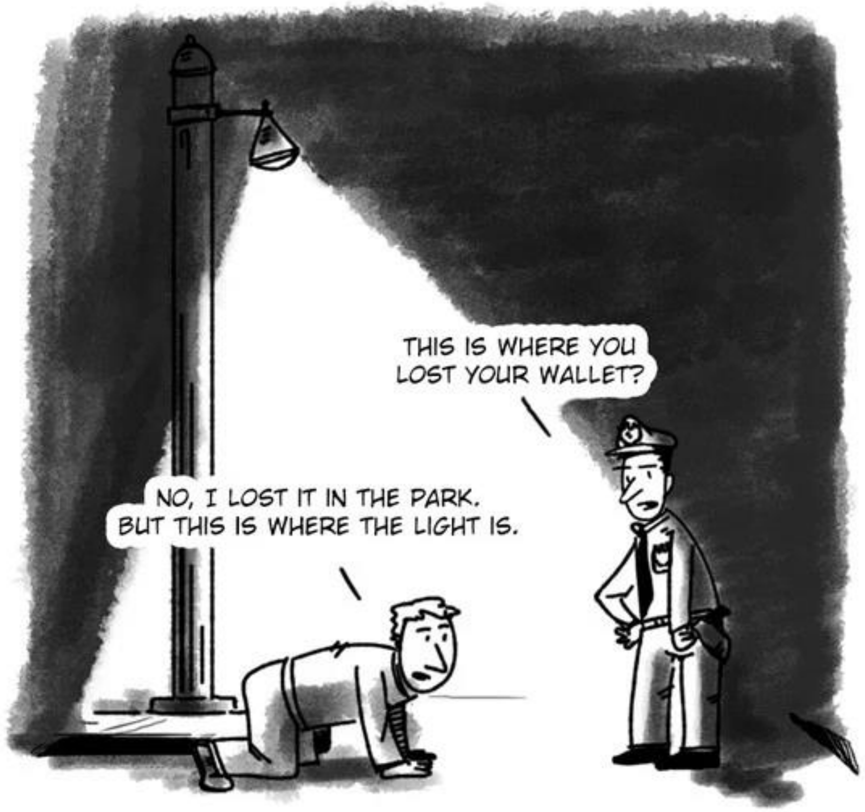
```
LCOM = P - Q
```

LACK OF COHESION IN METHODS

- Cohesiveness of methods is a sign of encapsulation
- Lack of cohesion implies that classes should be split

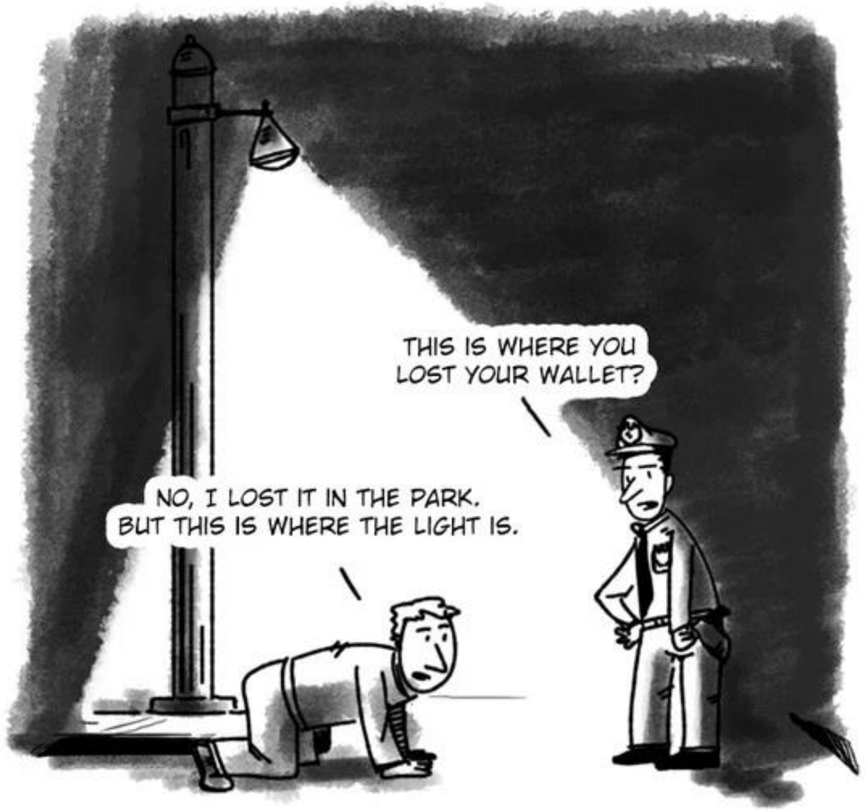
MEASUREMENT IS DIFFICULT

THE STREETLIGHT EFFECT



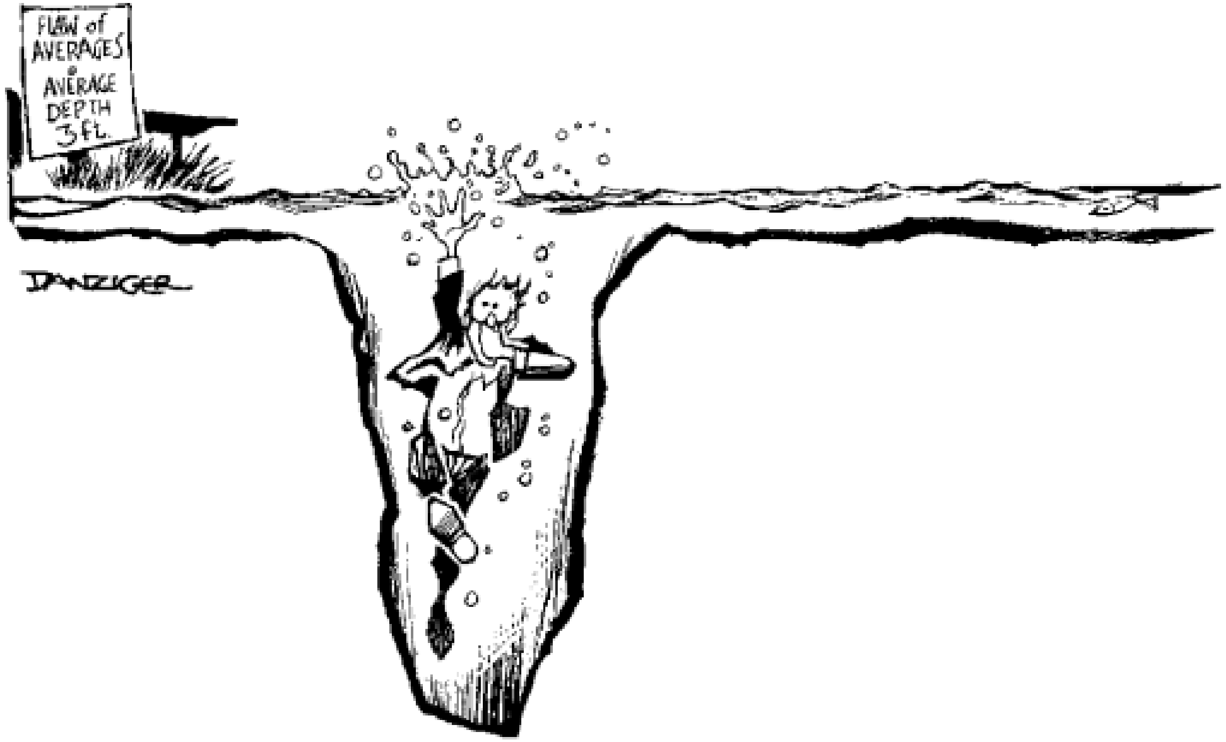
- A known observational bias
- People tend to look for something only where it's easiest to do so
  - If you drop your wallet at night, you'll tend to look for it under streetlights

THE STREETLIGHT EFFECT



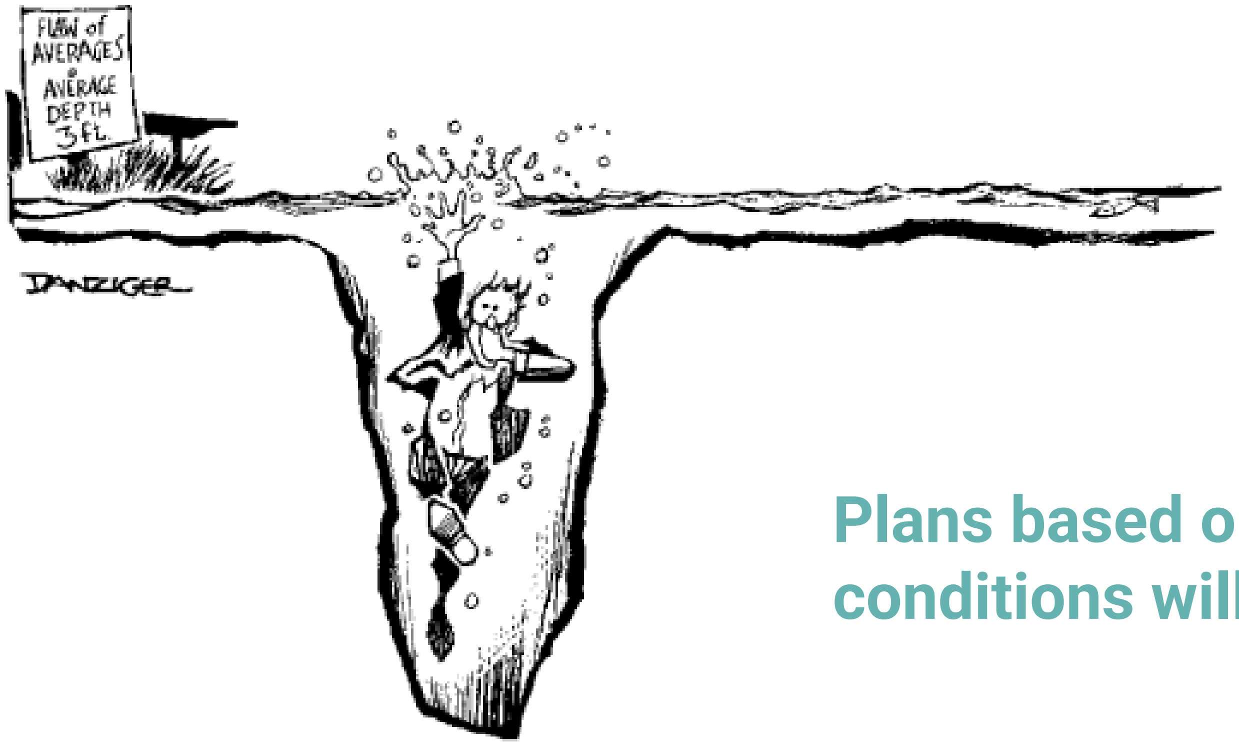
- Are we defining and measuring the right thing(s) to obtain understanding?
- Was only the easy and accessible measured because the important could not be measured?
- What was not measured?

THE FLAW OF AVERAGES



- A statistician drowns while crossing a river that is only 3-feet deep, on average
- A statistician put his head in the oven and his feet in the freezer so that on average he felt comfortable

THE FLAW OF AVERAGES

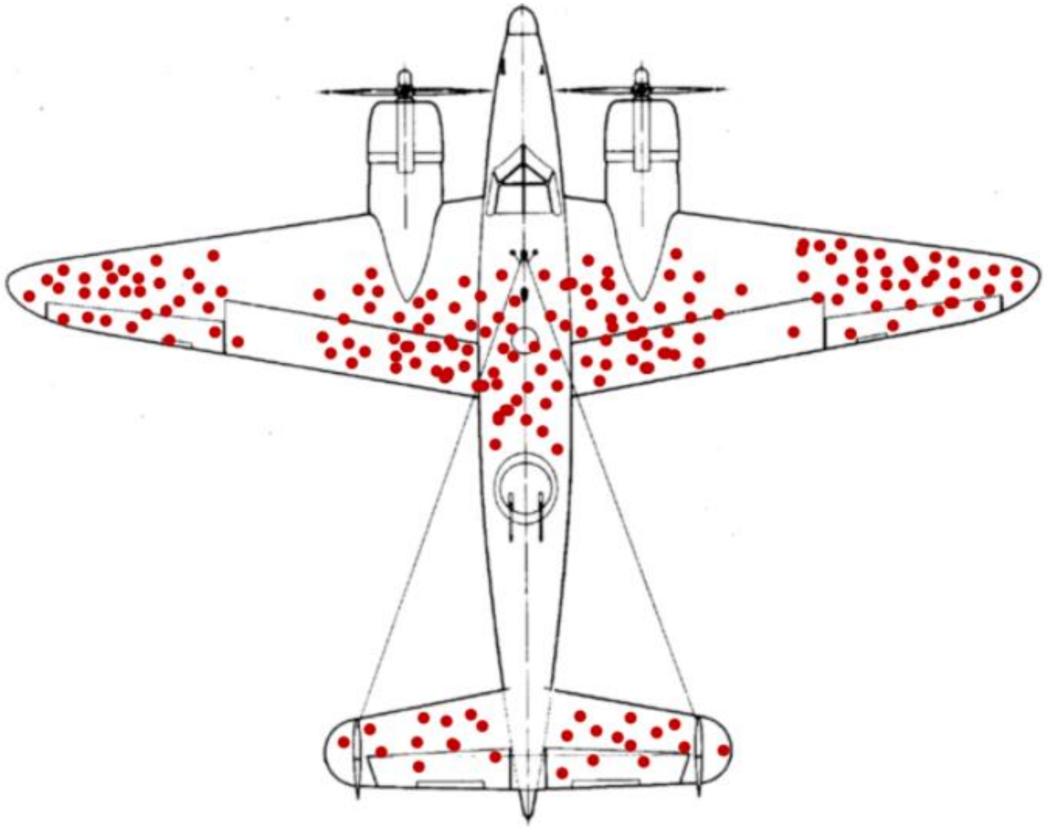


The flaw of averages is a set of systematic errors that occurs when people use single numbers (usually averages) to describe uncertain future quantities.

Plans based on the assumption that average conditions will occur are usually wrong

SURVIVORSHIP BIAS

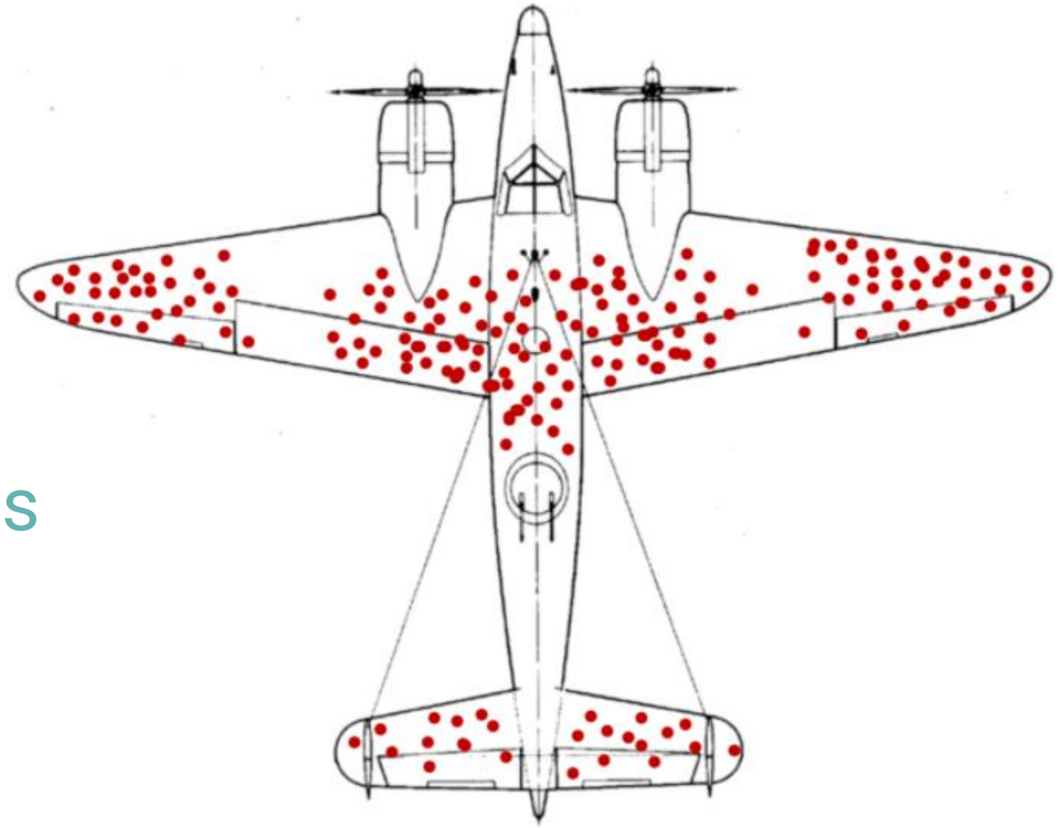
- Imagine you’re in charge of sending airplanes out to fight a war.
- The planes that do come back to base have been hit in the spots indicated by the red dots.
- Where should you add armor to reinforce them?



https://www.mcgill.ca/oss/article/general-science/tips-better-thinking-surviving-only-half-story

SURVIVORSHIP BIAS

- Our first instinct is to say "on the red dots!" This is where the planes were hit: let's make these areas stronger

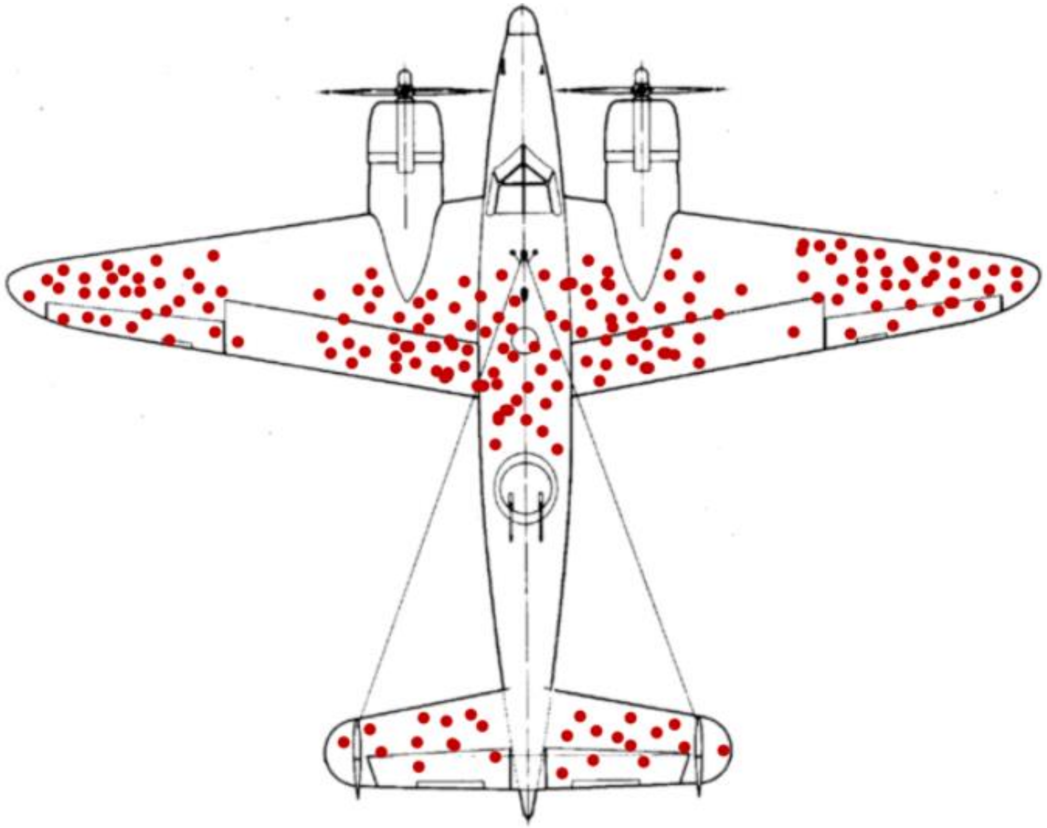


However, we're looking at airplanes that **survived**.

https://www.mcgill.ca/oss/article/general-science/tips-better-thinking-surviving-only-half-story

SURVIVORSHIP BIAS

- In World War II, statistician Abraham Wald recommended that planes be reinforced where **there were no red dots**, assuming that these were the spots that would deal a lethal blow to an airplane.

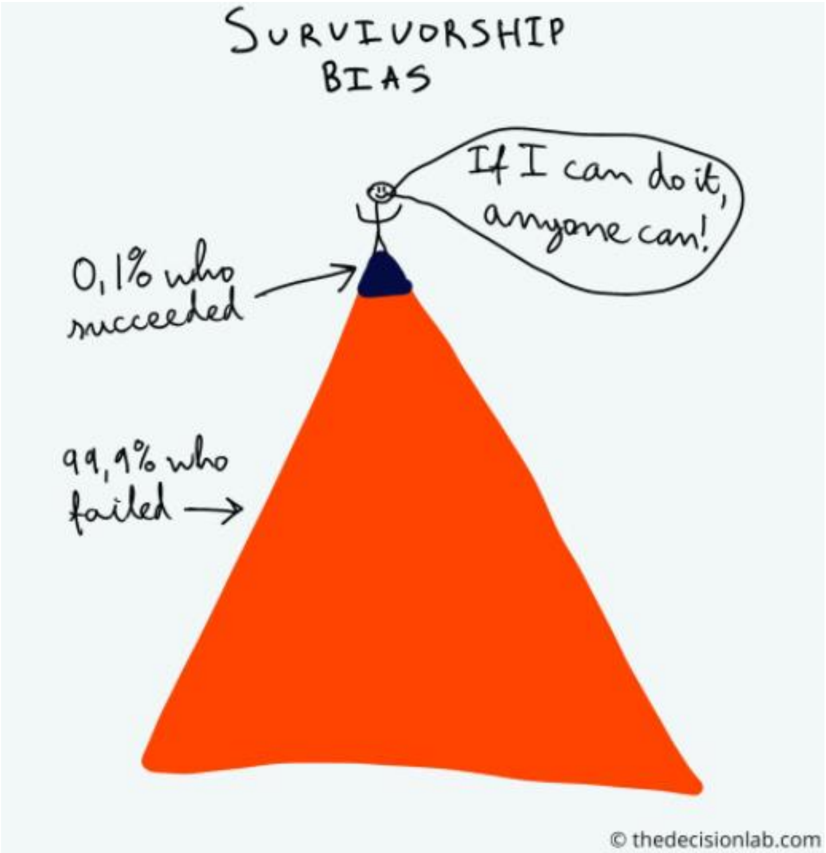


https://www.mcgill.ca/oss/article/general-science/tips-better-thinking-surviving-only-half-story

SURVIVORSHIP BIAS

Survivorship bias is a cognitive shortcut that occurs when a visible successful subgroup is mistaken as an entire group, due to the failure subgroup not being visible.

https://thedecisionlab.com/biases/survivorship-bias



EXAMPLE

- Dev team measures page load time, and optimize cache & image compressions accordingly
- Yet, the slowness is due to a subtle bug on the server code, which reduces server response time and is much harder to measure

A slow web application

THE STREETLIGHT EFFECT

EXAMPLE

- Dev team computes the average page load time, and finds it satisfactory

THE FLAW OF AVERAGES

A **slow** web application

EXAMPLE

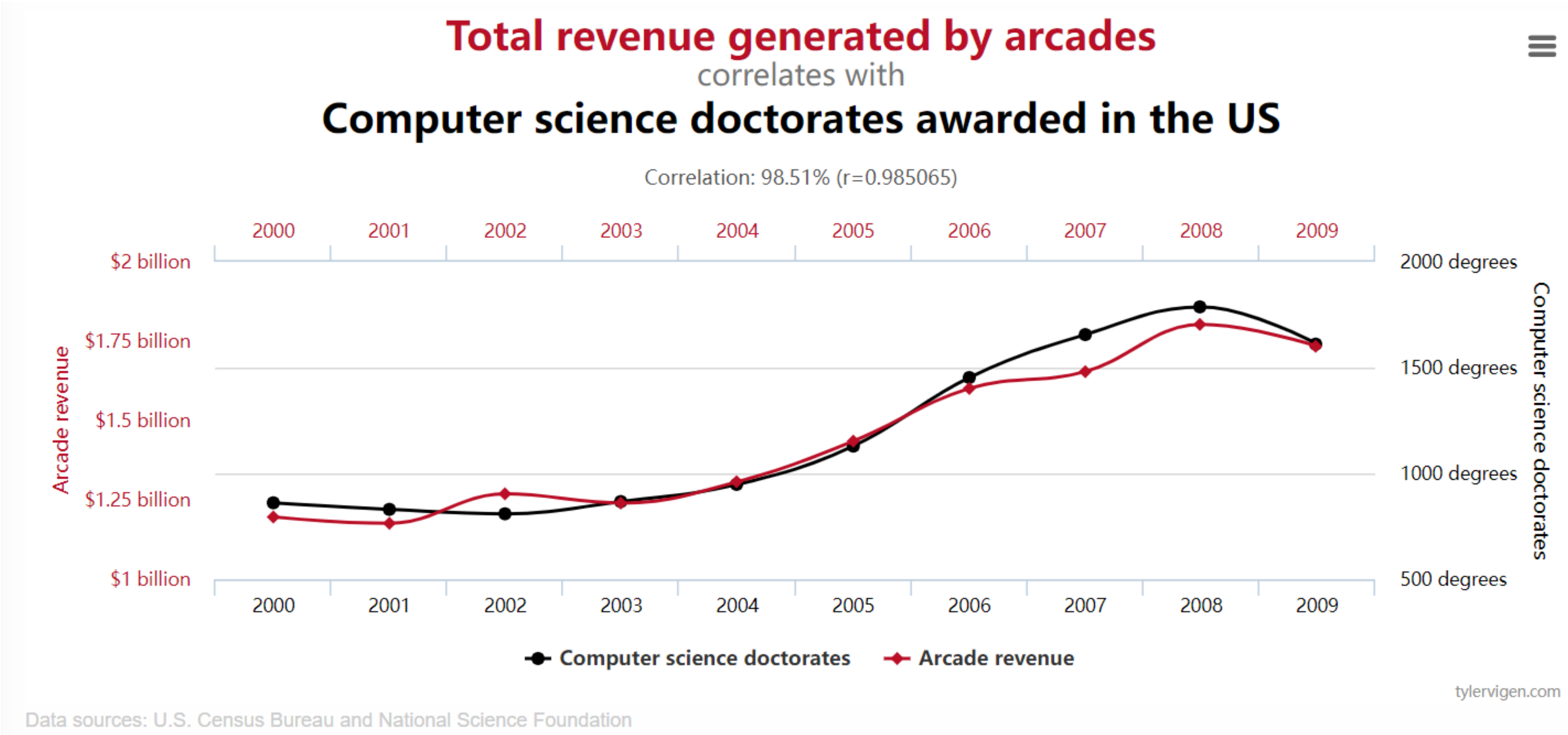
- Dev team collects and analyzes only executed user queries in order to optimize

A slow web application

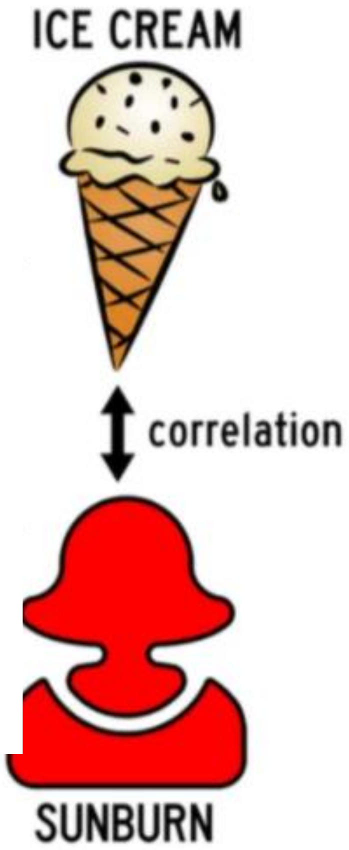
SURVIVORSHIP BIAS

What about the failed user queries that are never executed?

CORRELATION DOES NOT IMPLY CAUSATION



CORRELATION DOES NOT IMPLY CAUSATION



https://towardsdatascience.com/correlation-is-not-causation-ae05d03c1f53

CORRELATION DOES NOT IMPLY CAUSATION

Increased # of commits

Changes of requirements

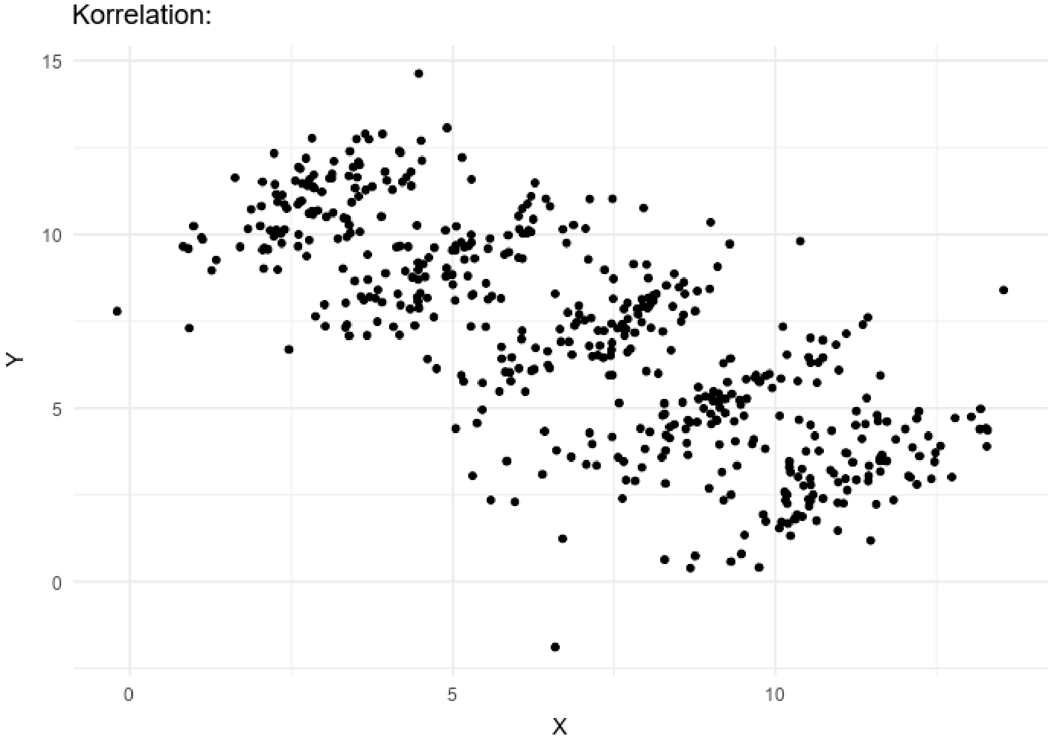
correlation

Increased # of bugs

DEV team concludes that by reducing the number of commits, they can also reduce the number of bugs in the codebase.

SIMPSON'S PARADOX

Simpson's paradox is a phenomenon in probability and statistics in which a trend appears in several groups of data but disappears or reverses when the groups are combined.



https://en.wikipedia.org/wiki/Simpson%27s_paradox

SIMPSON'S PARADOX: EXAMPLE

- By analyzing the programming experience and # of bugs produced for each individual developer, the company concludes that less experienced developers are better at writing high-quality code (produce less bugs)

- By analyzing the programming experience and # of bugs produced for each team, the company concludes that teams with more experienced developers are better at writing high-quality code (produce less bugs)

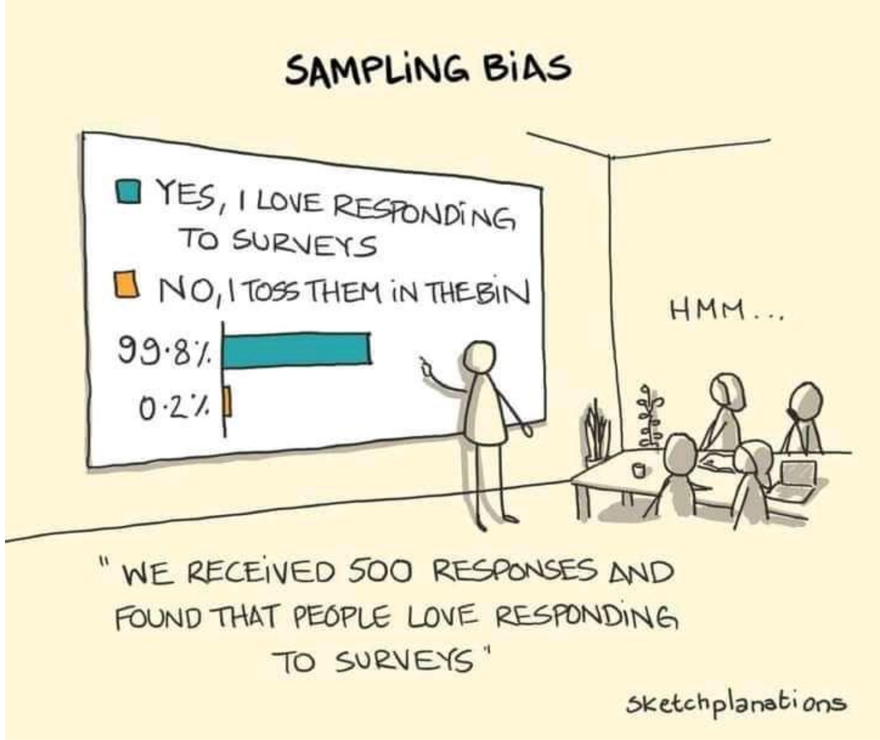
MEASUREMENTS VALIDITY

- **Construct validity**: are we measuring what we intended to measure?

To measure "user loyalty", we ask users for their "satisfaction".
Is "user satisfaction" the proper measure for "user loyalty"?

MEASUREMENTS VALIDITY

- **Internal validity** focuses on the accuracy of the conclusions drawn based on a cause and effect relationship (is the Cause-Effect okay?)



MEASUREMENTS VALIDITY

- **Internal validity** focuses on the accuracy of the conclusions drawn based on a cause and effect relationship (is the Cause-Effect okay?)

Dev team tests the performance of 2 algorithms. Factors that could threaten the internal validity:

- **Hardware**
- **Execution environment**
- **Configurations**
- **Test input size**
- **Etc.**

MEASUREMENTS VALIDITY

- **External validity**: Can the conclusions be generalizable?

Our findings are drawn by studying Java code. Can we say the same thing for Python?

MEASUREMENTS RELIABILITY

- Extent to which a measurement yields similar results when applied multiple times

- Goal is to reduce uncertainty and increase consistency

MEASUREMENTS RELIABILITY

- Example: measure program performance
  - Time, memory usage
  - Should measure many times

- Law of large numbers (大数定律)
  - As a sample size grows, its mean gets closer to the average of the whole population.
  - Taking multiple measurements to reduce error

WARNINGS

- Most software metrics are controversial
  - Usually only plausibility argument, rarely rigorously validated
  - Cyclomatic complexity was repeatedly refuted and is still used
  - Code size still dominates many metrics
- Metrics can be gamed: you get what you want
- Metrics and measurements are important for decision making.
- Pick or design suitable metrics and use them carefully!

NEXT

- Software evolution
- Software maintenance