# Software: the Ground Rules

## Truths, Facts, Axioms

Itay Maman

~~Solid~~

Nothing new

You can’t control what you can’t measure.

– Tom DeMarco, 1982

I can see why measuring productivity is so seductive. If we could do it we could assess software much more easily and objectively than we can now. But false measures only make things worse. This is somewhere I think we have to admit to our ignorance.

– Martin Fowler, 2003

# Truth 1: Software is buggy

No systematic way to detect bugs

# Truth 2: Recursiveness (I)

A module is built from similar modules

# Implications

Difficult to estimate size (time)

A building spans a fixed-depth hierarchy

Building → Floor → Apartment → Room

A class spans a tree of unbounded depth

# Truth 3: Recursiveness (II)

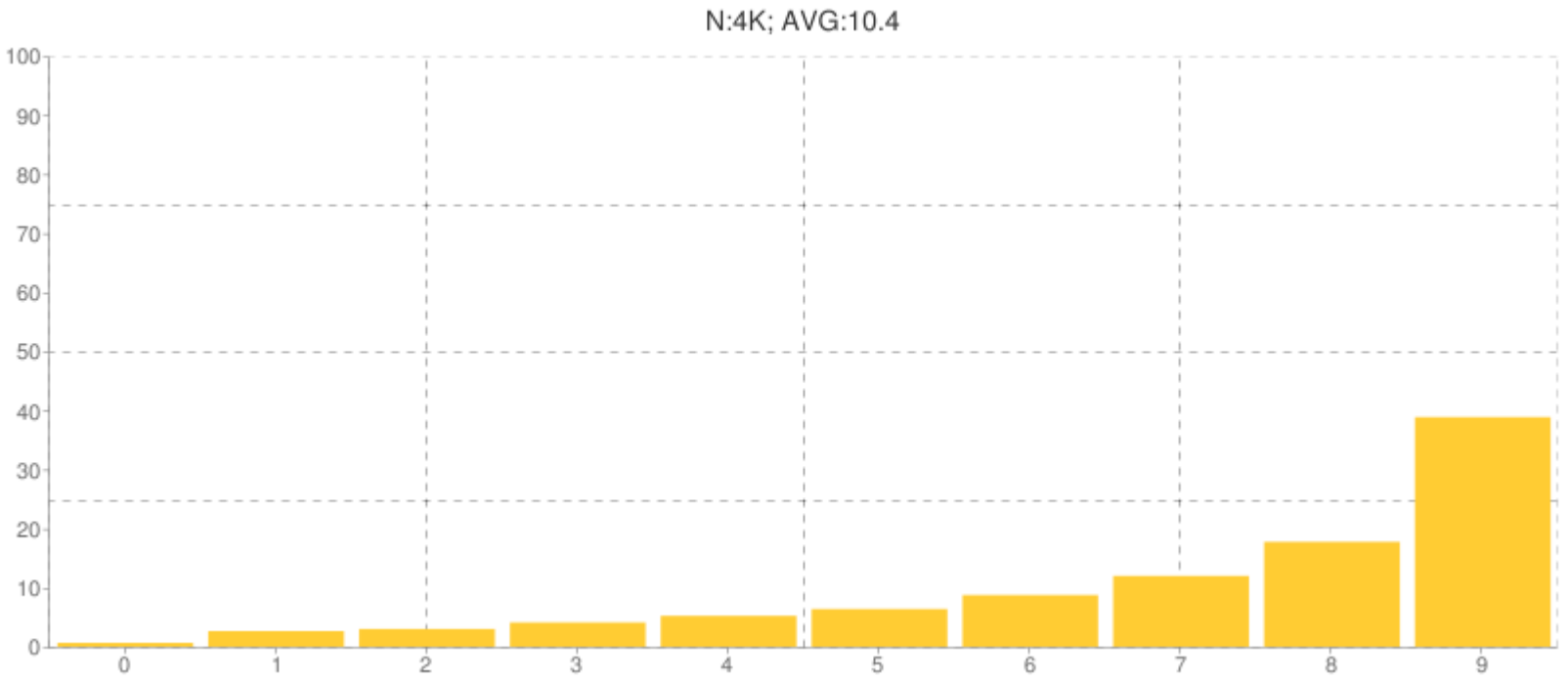
Every programming language
is powerful enough to express itself
(or every other language)

# Fact 1: Long Tail

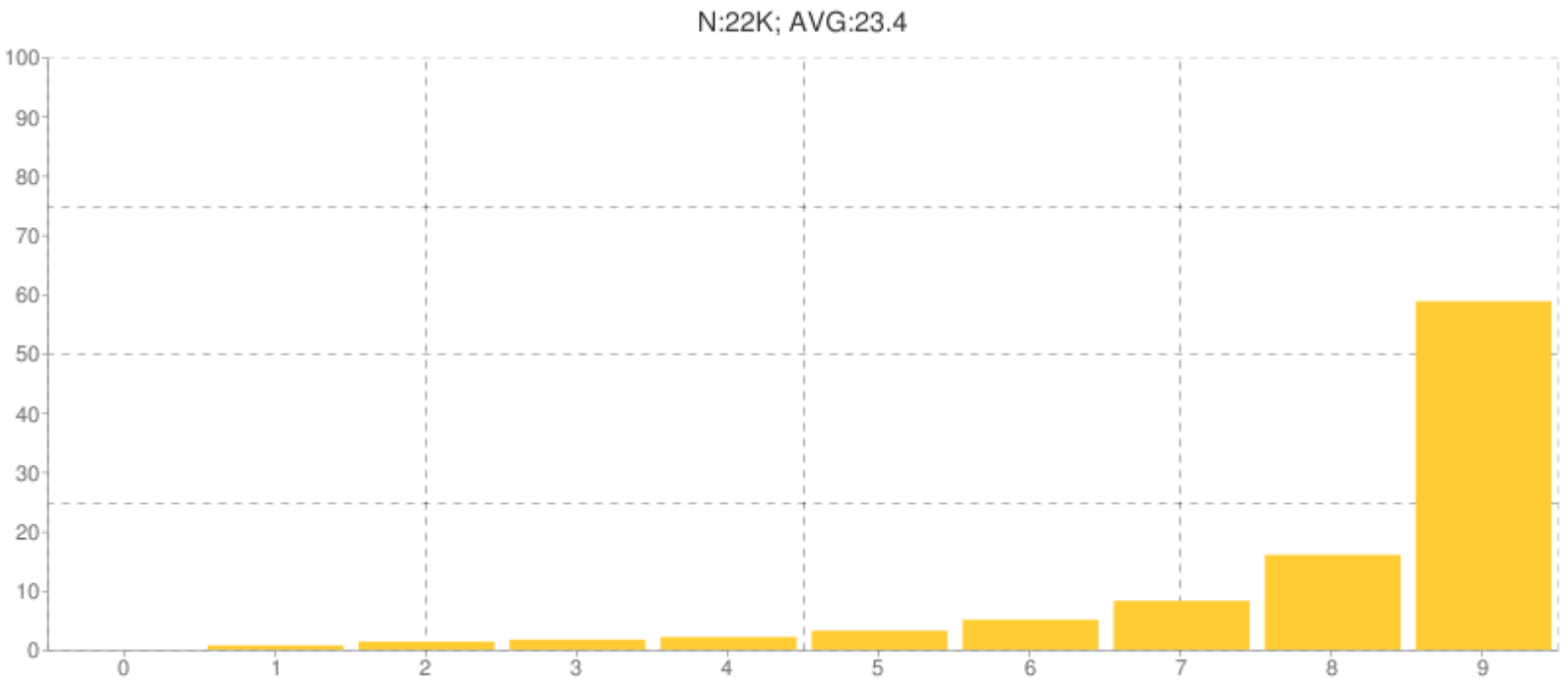
How many methods are ugly?

How ugly are they?

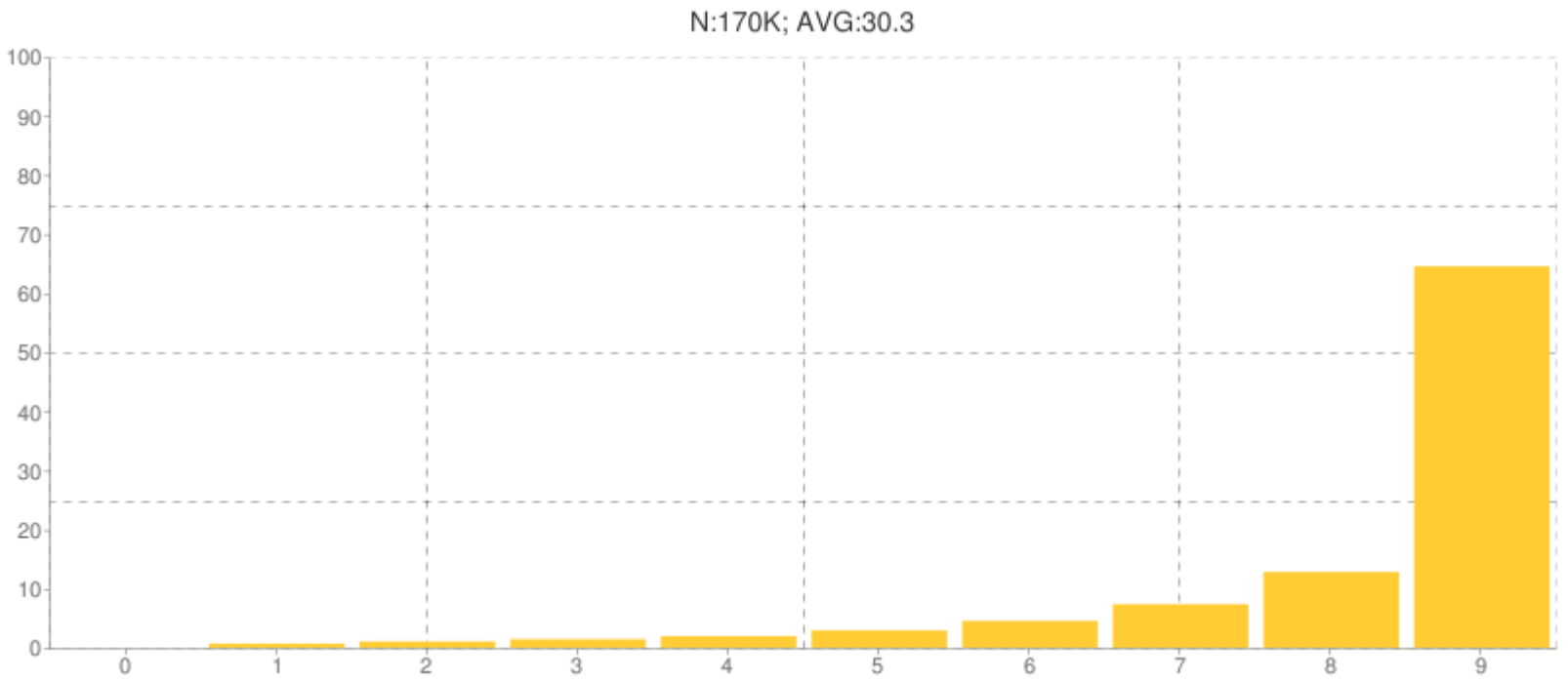
# JUnit



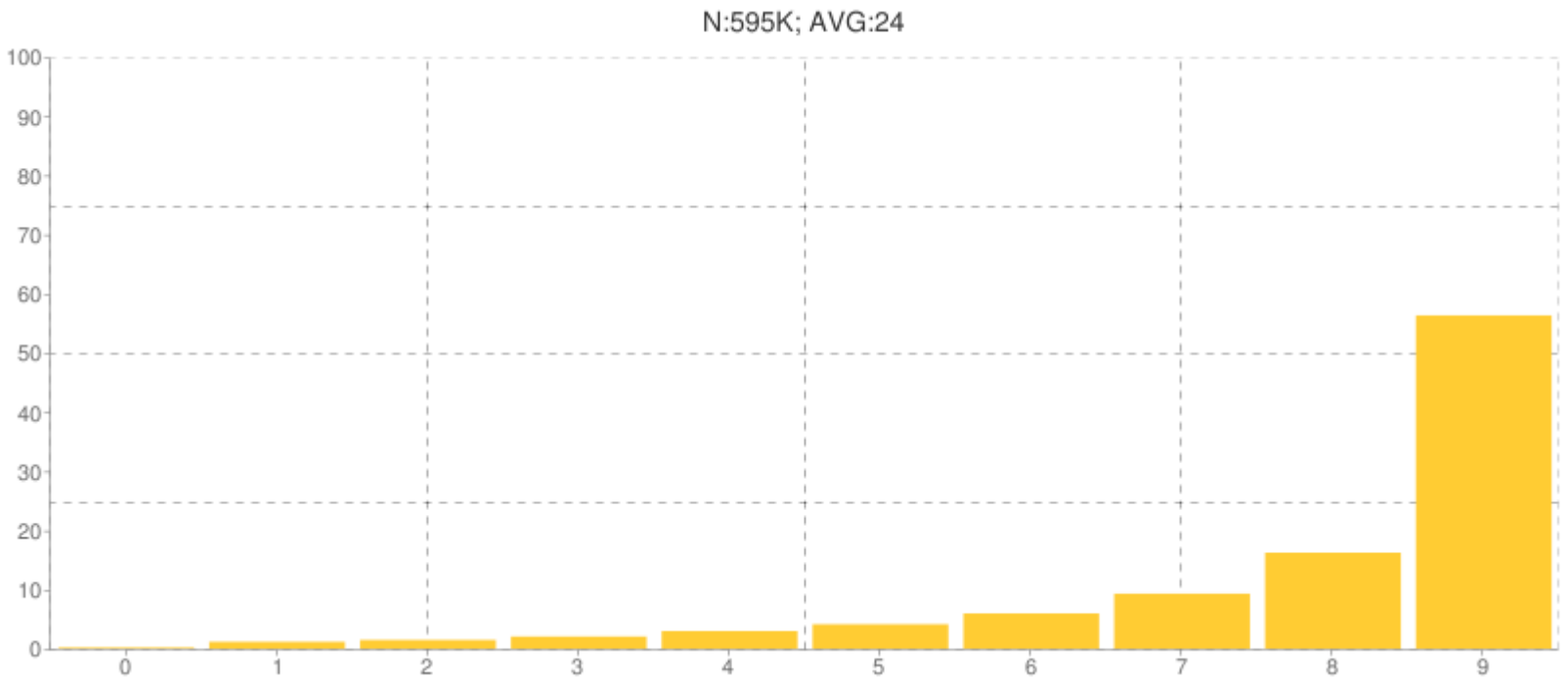
# Ant



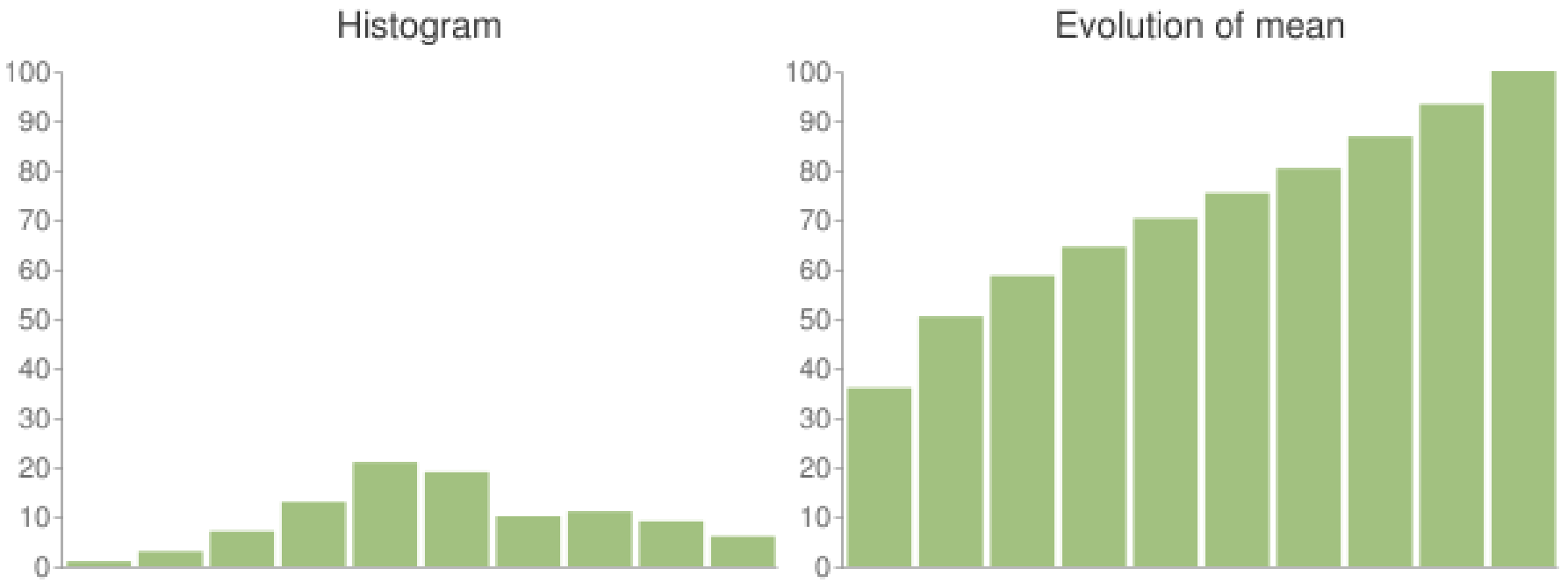
# JRE



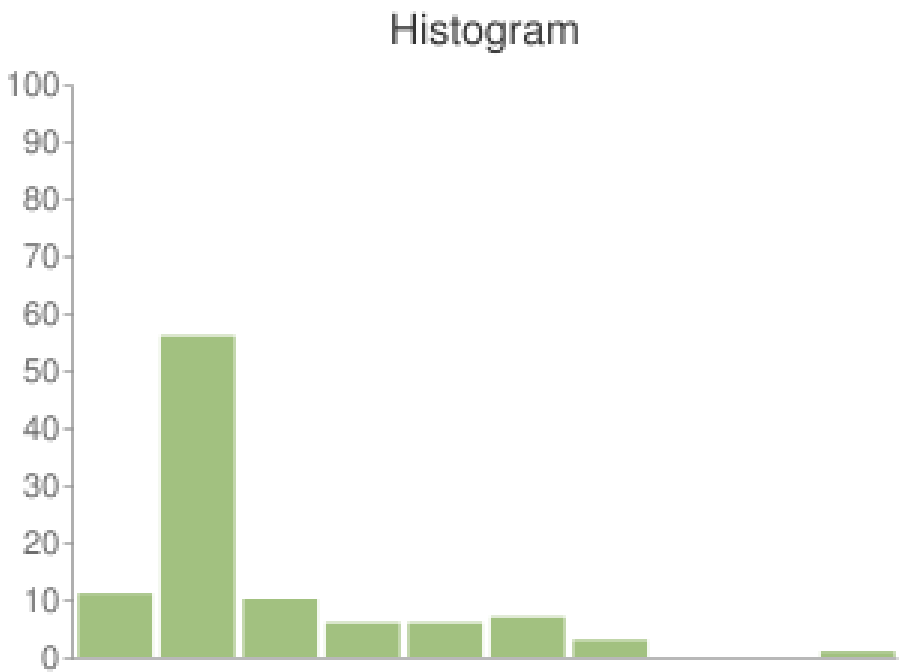
# Eclipse

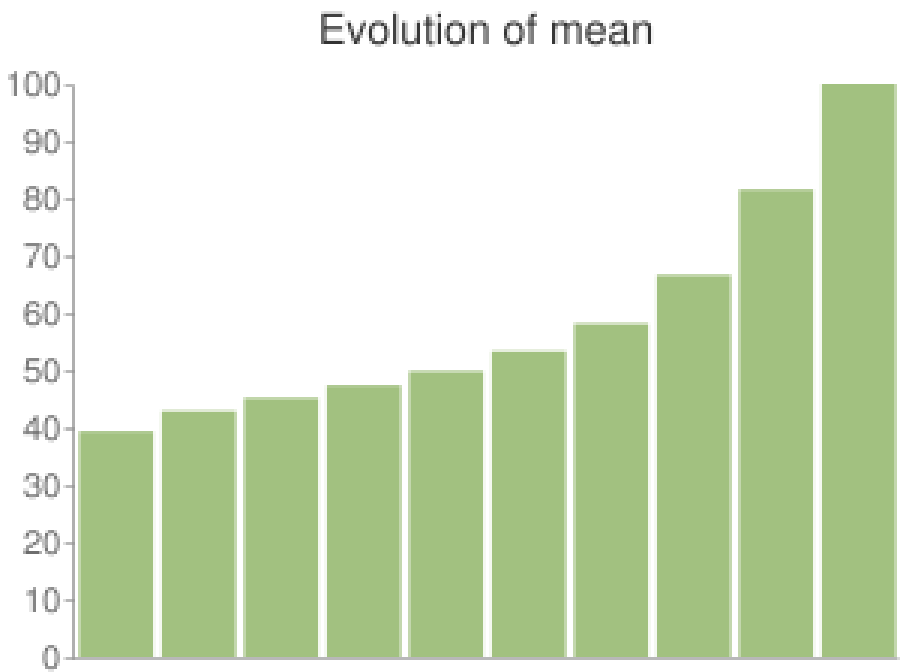


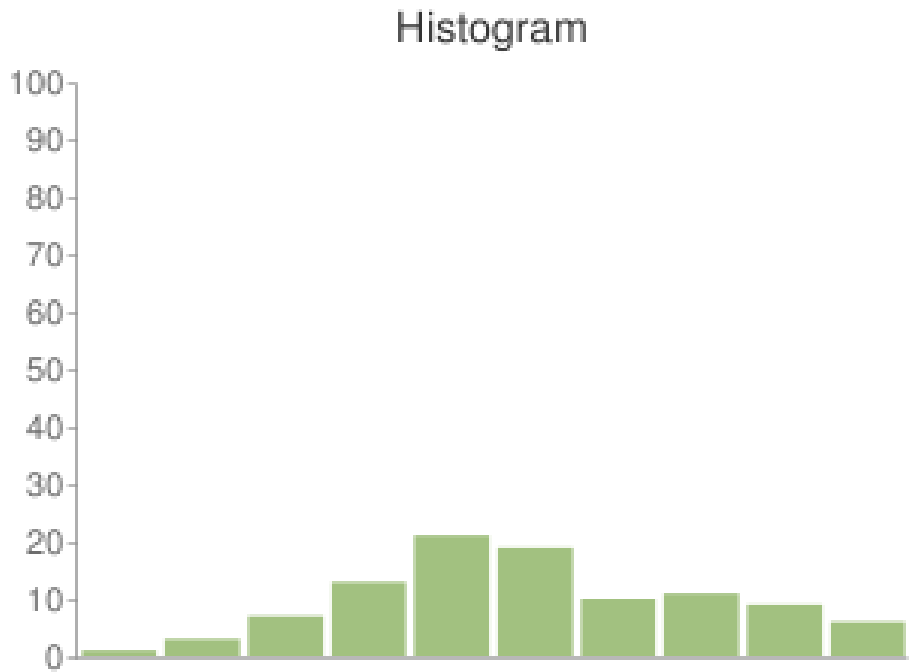
# Normal Distribution

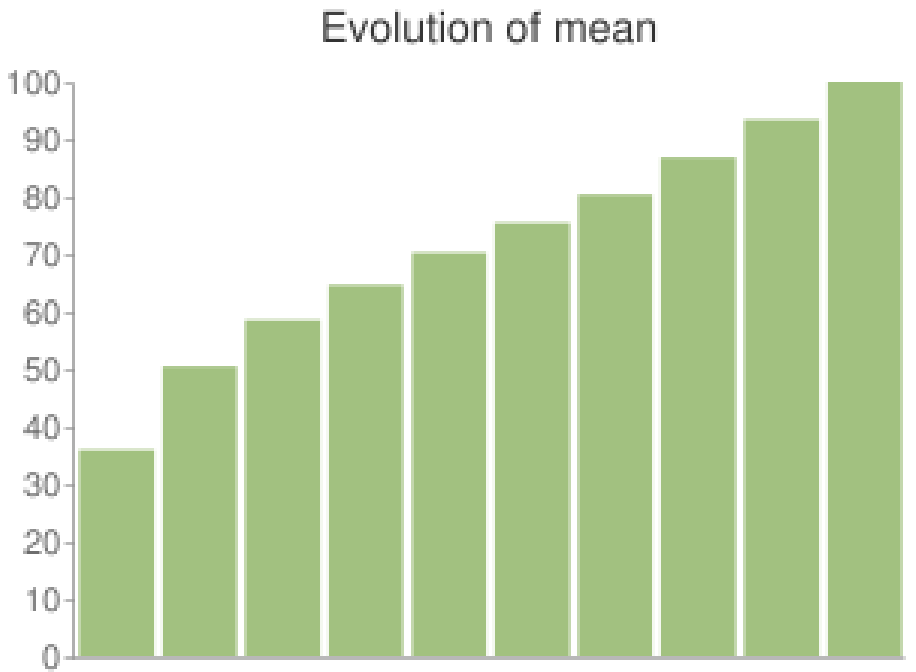


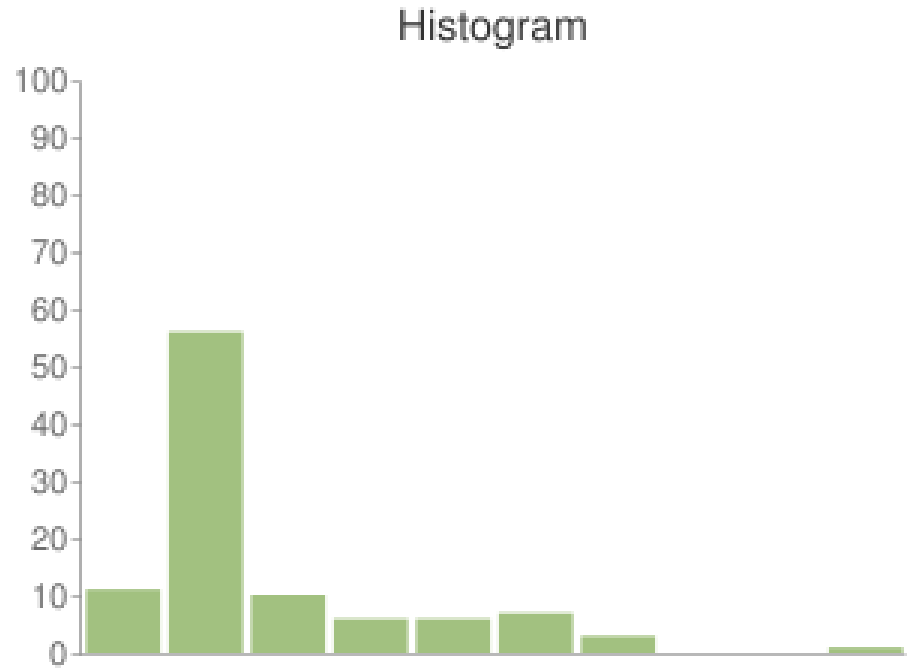
# Power-Law Distribution

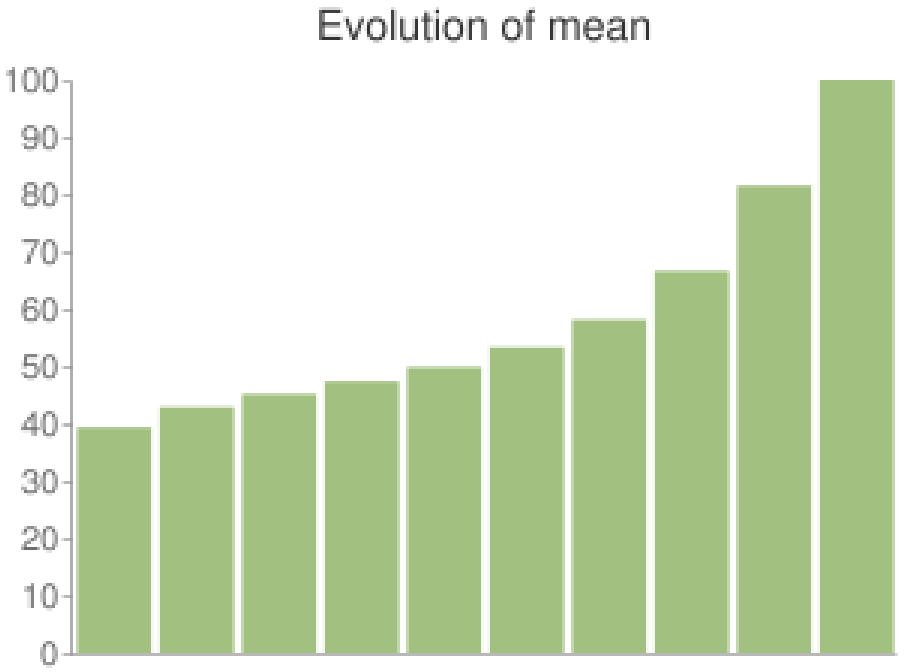



Histogram
100
90
80
70
60
50
40
30
20
10
0


Evolution of mean
100
90
80
70
60
50
40
30
20
10
0


Histogram
100
90
80
70
60
50
40
30
20
10
0


Evolution of mean
100
90
80
70
60
50
40
30
20
10
0

# Implications

Costs: Expect the worse

Cheap & expensive do not cancel out

# Fact 2: Entropy

Code gets messy over time

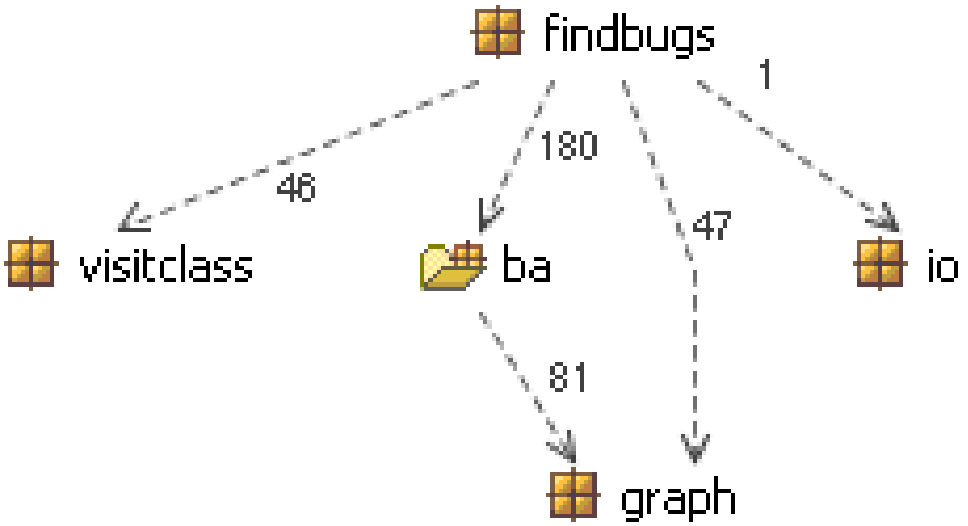
Restoring order requires explicit effort

# Thermodynamics

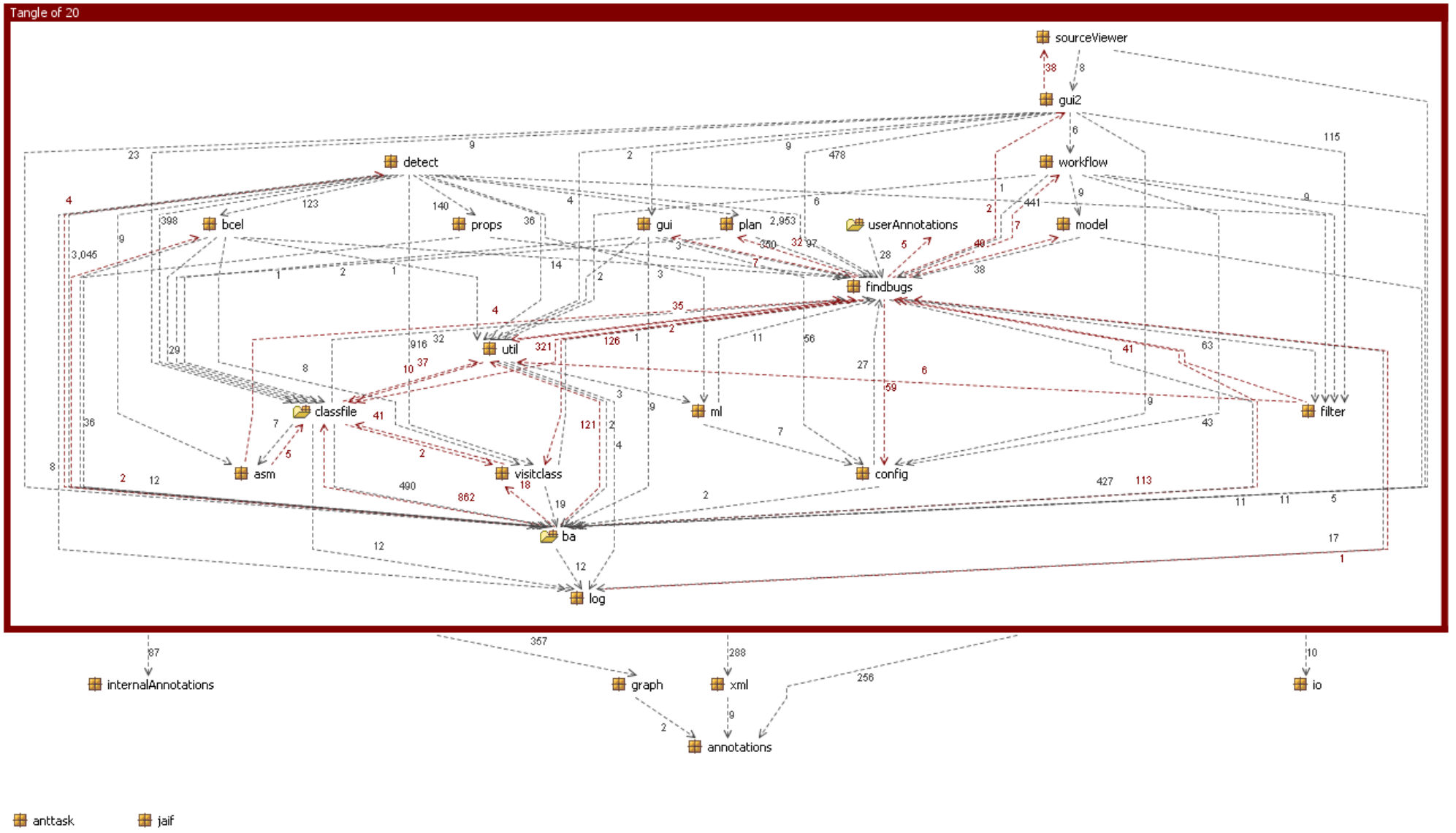
Chaos occurs naturally

External energy is needed to restore order

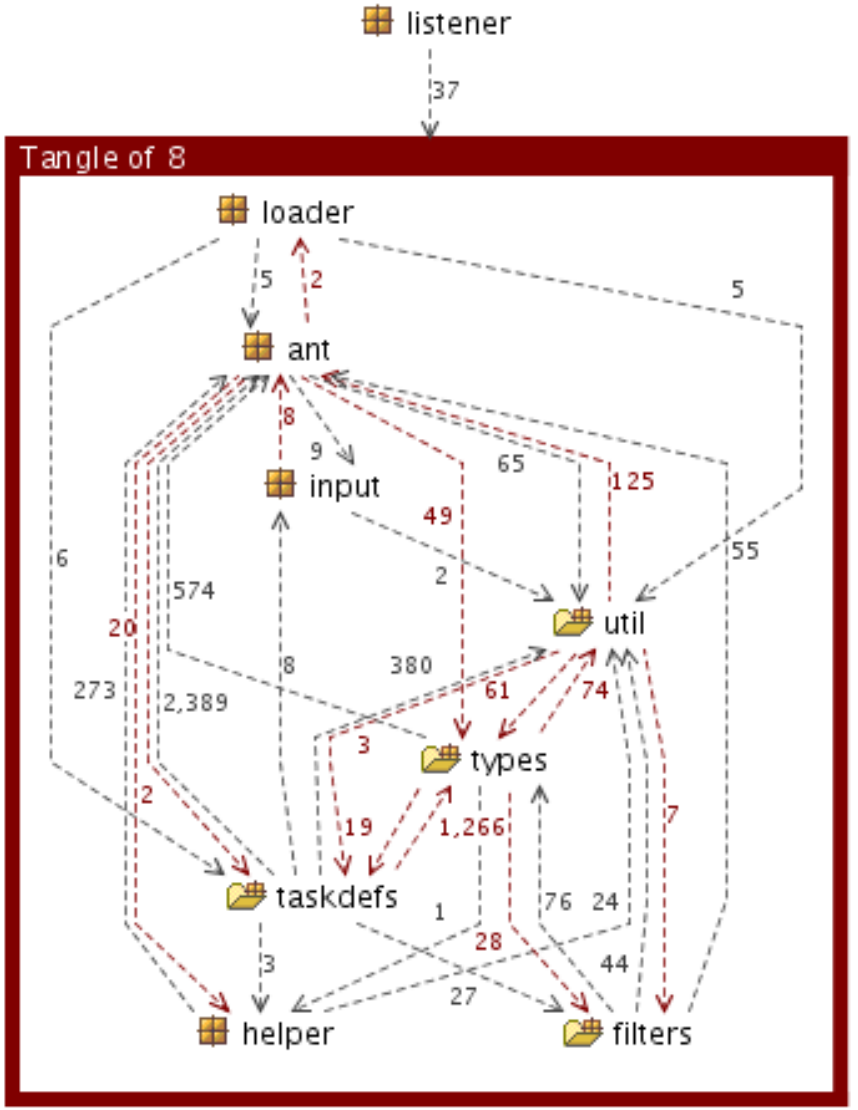
# FindBugs 0.72 (March 2004)



# FindBugs 1.35 (September 2008)



# Ant 1.65


listener
37
Tangle of 8
loader
5
2
5
ant
8
9
65
125
input
49
6
574
2
55
util
20
8
380
273
2,389
61
74
3
types
2
19
1,266
7
taskdefs
1
76
24
28
44
3
27
helper
filters

AVOID ACCIDENTS

# We have gone

# 1

# days since last

‘drop database `production`’

KEEP YOUR COWORKERS SAFE

# Axiom 1: Discontinuity

One small step for a programmer,

One giant failure for an entire company

A few hours ago I was upgrading our continuous integration setup when a configuration error caused it to run against our production environment rather than our testing environment.

– Chris Wanstrath

Today we made a change to the persistent copy of a configuration value that was [mistakenly] interpreted as invalid [by an automated system for verifying configuration values].

This meant that every single client saw the invalid value and attempted to fix it. Because the fix involves making a query to a cluster of databases, that cluster was quickly overwhelmed by hundreds of thousands of queries a second.

... To make matters worse, every time a client got an error attempting to query one of the databases it interpreted it as an invalid value, and deleted the corresponding cache key.

This meant that even after the original problem had been fixed, the stream of queries continued. As long as the databases failed to service some of the requests, they were causing even more requests to themselves. We had entered a feedback loop that didn't allow the databases to recover.

– Robert Johnson

# Explanations

Binary World

Most approximations will fail

# Implications

Hard to predict effect/scope of a change

Retest everything

# Axiom 2: Super-linearity

Size Matters

# Axiom 2: Super-linearity

Smaller is better

# Axiom 2: Super-linearity

Cost of a programming task
is super-linear in its size

# Axiom 2: Super-linearity

Few large tasks more expensive

than

numerous small tasks

# Manifestation

Source control conflict resolution

1 day vs. 1 month

# Manifestation

Debugging

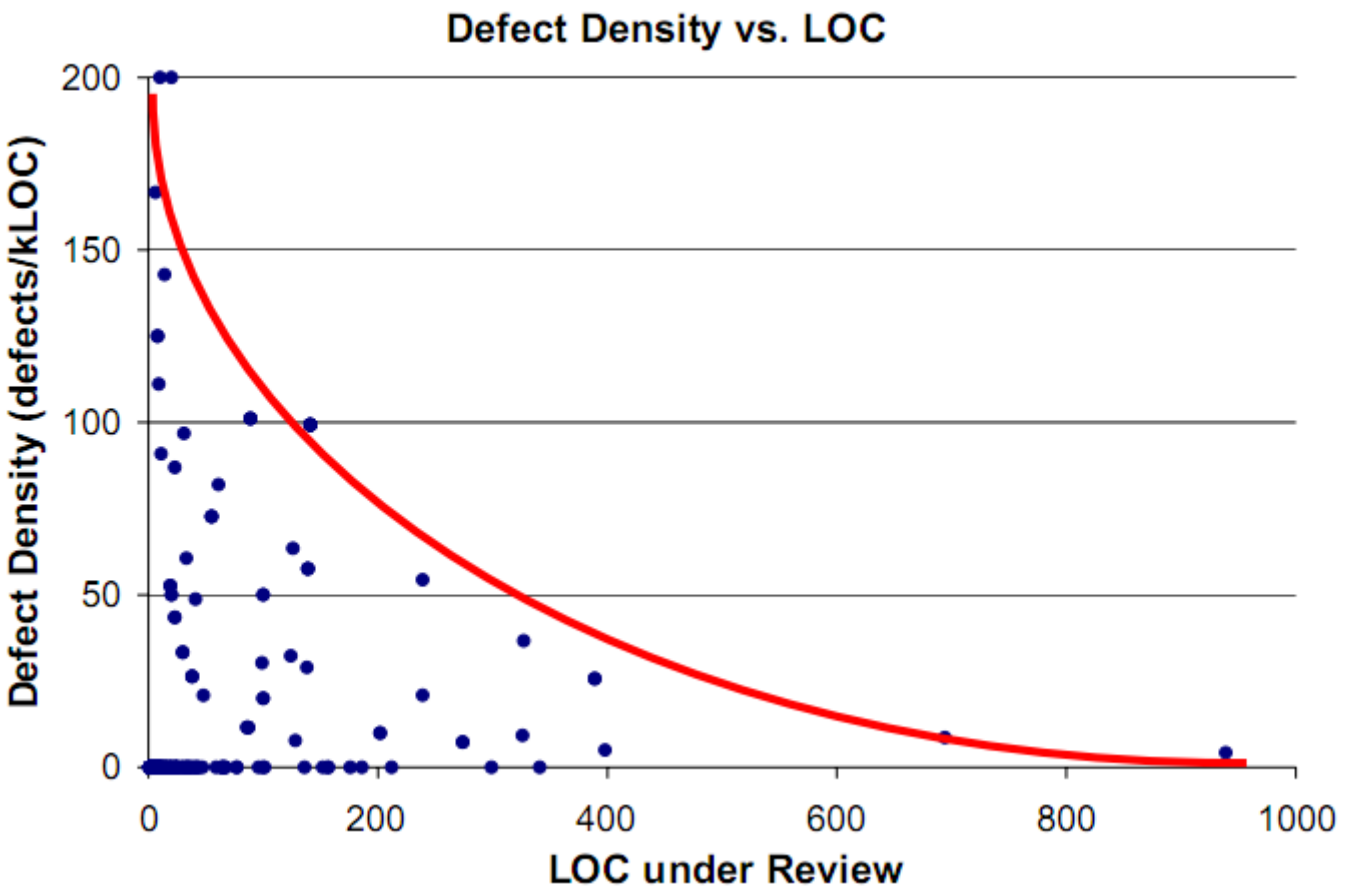
Build cycle: 10 sec. vs. 5 min.

# Manifestation

Debugging

Reproduction steps: Automatic vs. GUI

# Manifestation

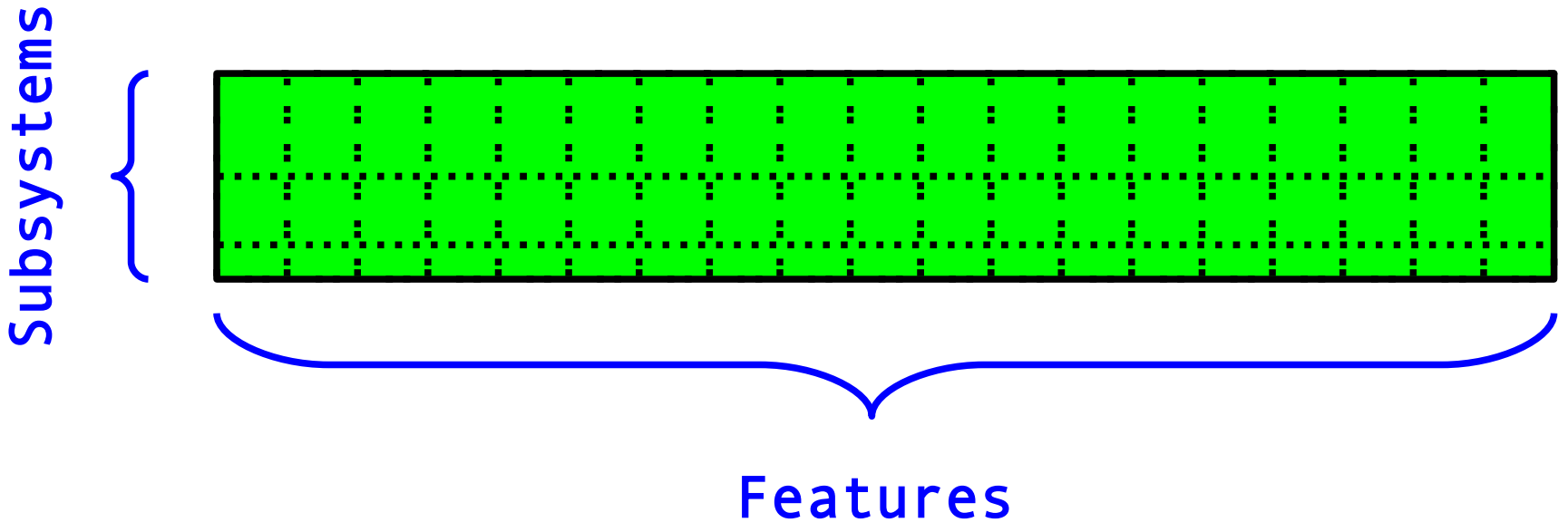


# Explanation

7 +/- 2

S/W construction == A tree of decisions

# Exploitation

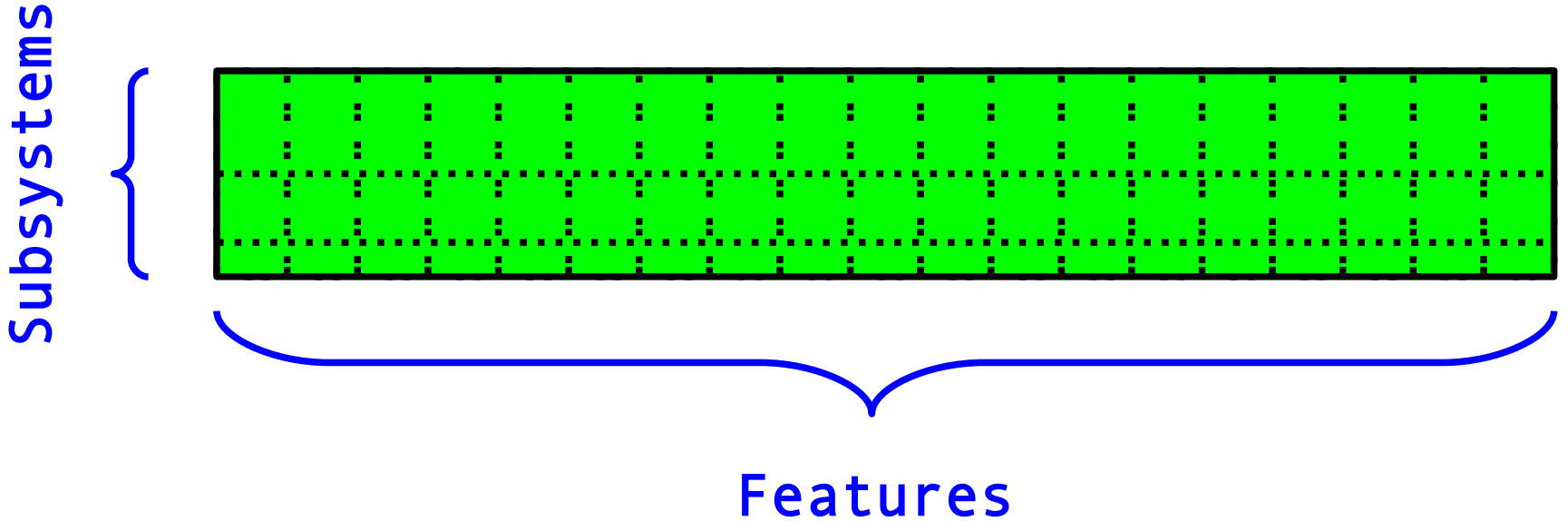


# Benefits

Higher throughput

#Features >> #Subsystems

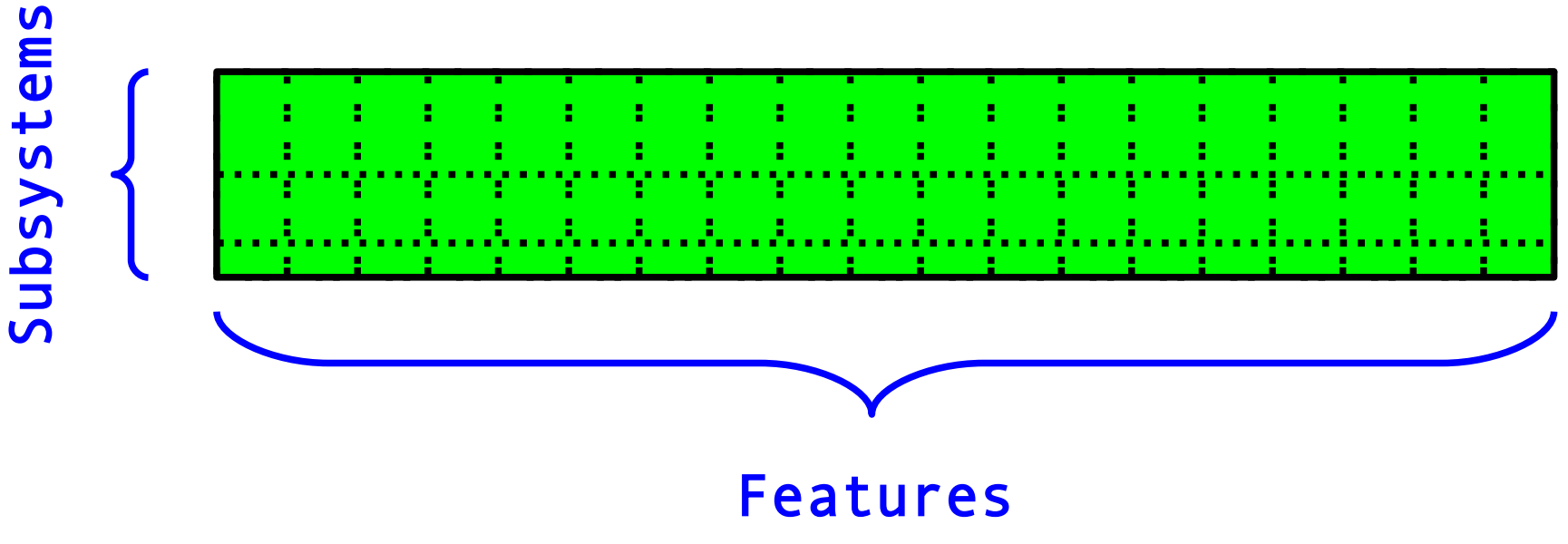
size(feature) << size(subsystem)



# Benefits

Less starvation

Reduce the damage of delays

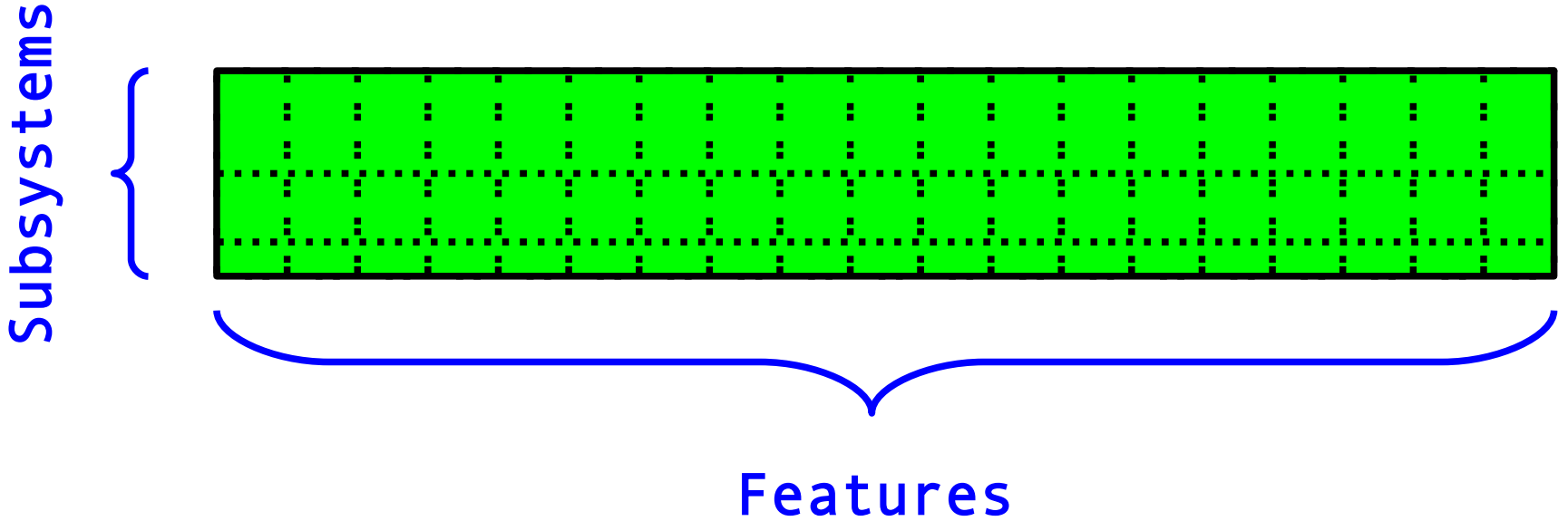


# Challenge

Efficient decomposition

into

Small, independent, tasks



# Exploitation

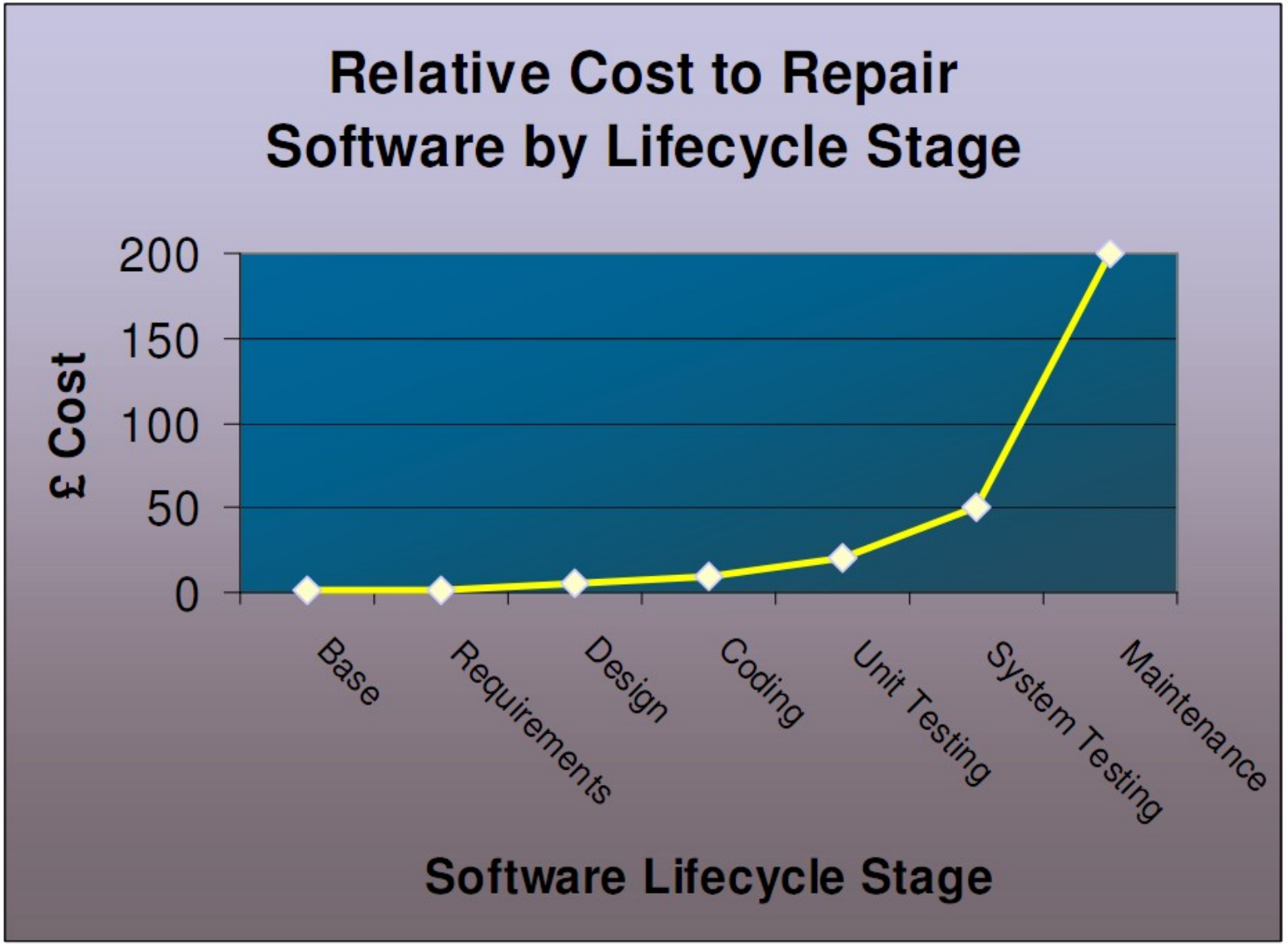
Plugin-architectures

TDD

Continuous Deployment

Short loop to client

# Old data, New Conclusions



# Analogies

Appealing

# Analogies

Risky

# Analogies?

Car Manufacturing

Car Design

Buildings

Novels

Cities

# Imagine...

A factory that interacts with its own products,

Can produce itself,

Goes wild when changed,

With unpredictable maintenance times

In my reflective mood, I'm wondering, was its advice ["You can't control what you can't measure"] correct at the time, is it still relevant, and do I still believe that metrics are a must for any successful software development effort?

In my reflective mood, I'm wondering, was its advice ["You can't control what you can't measure"] correct at the time, is it still relevant, and do I still believe that metrics are a must for any successful software development effort?

My answers are no, no, and no.

In my reflective mood, I'm wondering, was its advice ["You can't control what you can't measure"] correct at the time, is it still relevant, and do I still believe that metrics are a must for any successful software development effort?

My answers are no, no, and no.

– Tom DeMarco, 2009

# Thank you

http://javadots.blogspot.com