The Metrics Trap

Michael Feathers

We all know *fold*

```
foldl1 (+) [1..5]
```

```
foldl1 (+) [1..5]

15
```

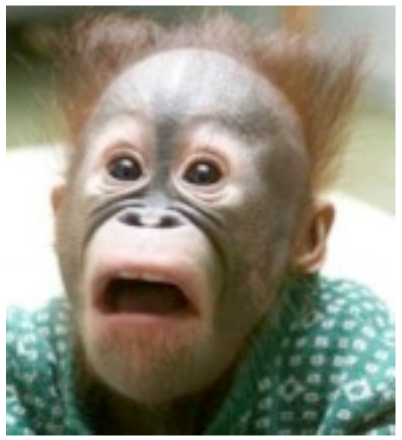
We all know Average

```haskell
data Pair = Pair !Int !Double

average :: Vector Double -> Double
average xs = s / fromIntegral n
  where
    Pair n s         = foldl' k (Pair 0 0) xs
    k (Pair n s) x = Pair (n+1) (s+x)
```

```haskell
data Pair = Pair !Int !Double

average :: Vector Double -> Double
average xs = s / fromIntegral n
  where
    Pair n s         = foldl' k (Pair 0 0) xs
    k (Pair n s) x = Pair (n+1) (s+x)
```



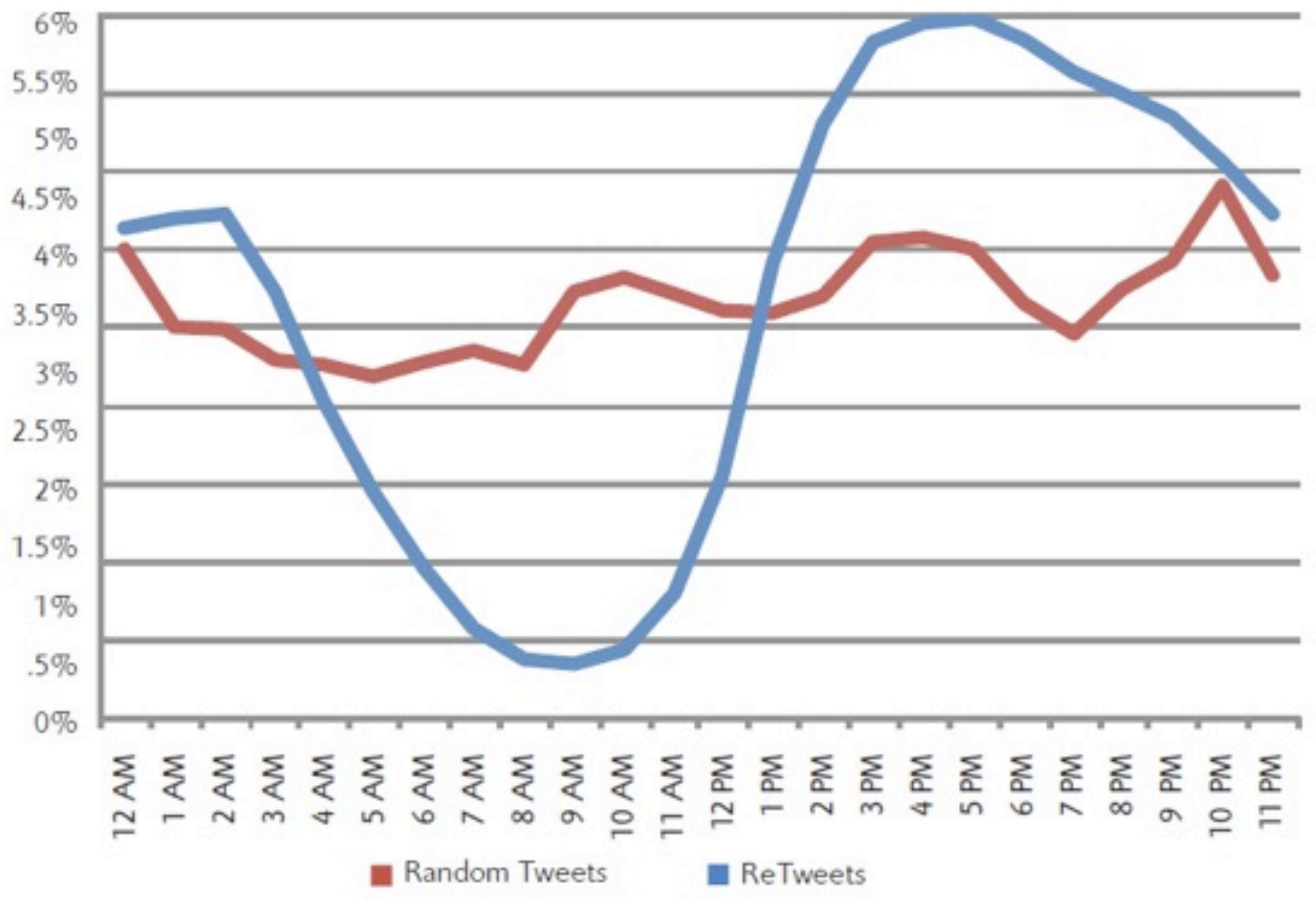
Standard ways of reducing information

```
(average [1, 1, 1, 1, 1, 1])  ==  (average [11, -9, 12, -10])
```

It turns out that we do this all the time

10
50
20
EURO
© BCE ECB EZB EKT EKP 2002

Time of Day (EST)



Quantitative Bias

In every use of units, there's an assumption of uniformity

Quantitative Bias

In every use of units, there's an assumption of uniformity

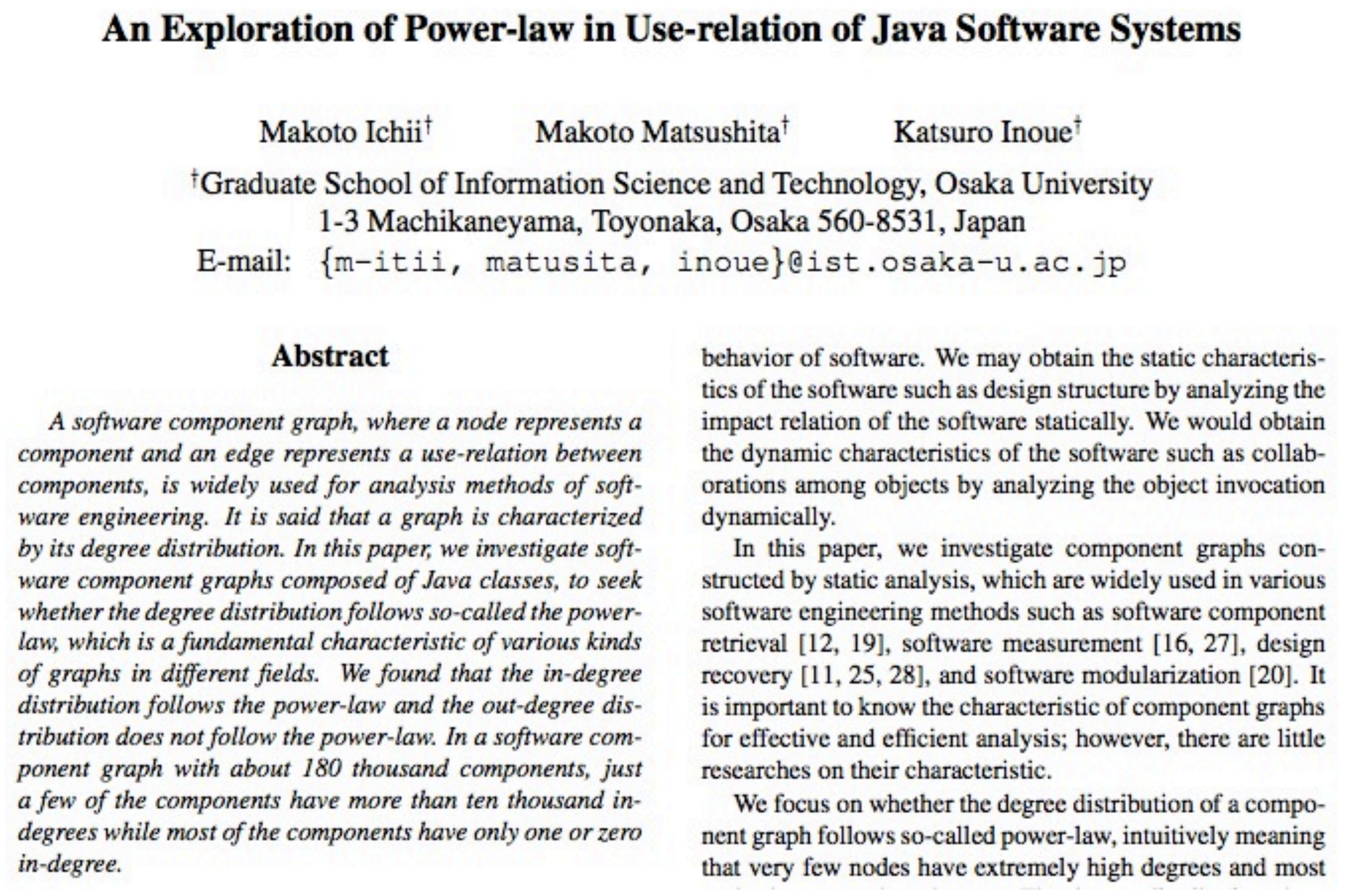
1 *can be exchanged for* 1

“We want to establish key metrics and indicators to measure our progress toward the goal.”

"We want to establish key metrics and indicators to measure our progress toward the goal."

But..

We don’t have (m)any uniform distributions in software



An Exploration of Power-law in Use-relation of Java Software Systems

Makoto Ichii[†] Makoto Matsushita[†] Katsuro Inoue[†]

[†]Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
E-mail: {m-itii, matusita, inoue}@ist.osaka-u.ac.jp

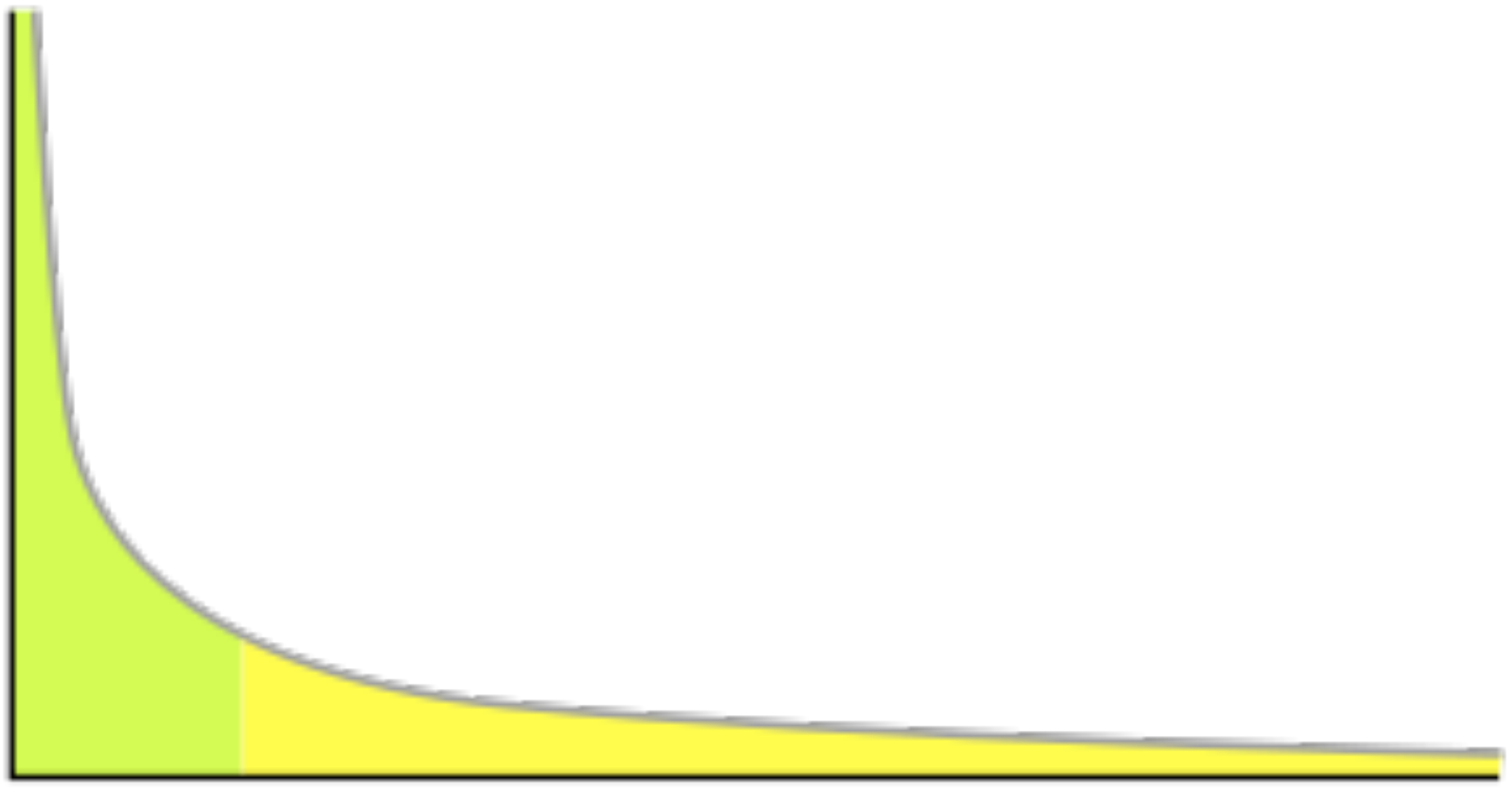
Abstract

A software component graph, where a node represents a component and an edge represents a use-relation between components, is widely used for analysis methods of software engineering. It is said that a graph is characterized by its degree distribution. In this paper, we investigate software component graphs composed of Java classes, to seek whether the degree distribution follows so-called the power-law, which is a fundamental characteristic of various kinds of graphs in different fields. We found that the in-degree distribution follows the power-law and the out-degree distribution does not follow the power-law. In a software component graph with about 180 thousand components, just a few of the components have more than ten thousand in-degrees while most of the components have only one or zero in-degree.

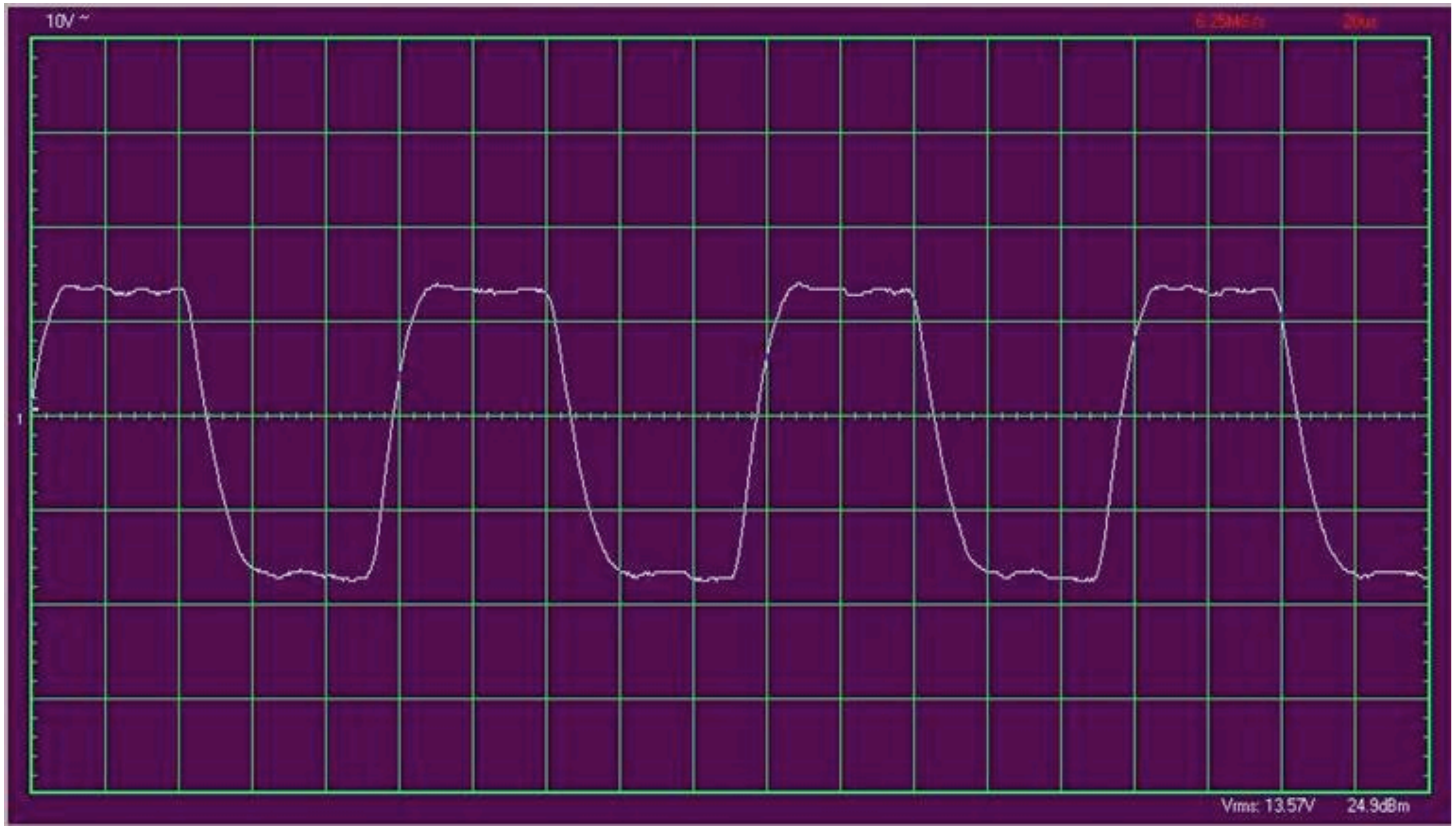
behavior of software. We may obtain the static characteristics of the software such as design structure by analyzing the impact relation of the software statically. We would obtain the dynamic characteristics of the software such as collaborations among objects by analyzing the object invocation dynamically.

In this paper, we investigate component graphs constructed by static analysis, which are widely used in various software engineering methods such as software component retrieval [12, 19], software measurement [16, 27], design recovery [11, 25, 28], and software modularization [20]. It is important to know the characteristic of component graphs for effective and efficient analysis; however, there are little researches on their characteristic.

We focus on whether the degree distribution of a component graph follows so-called power-law, intuitively meaning that very few nodes have extremely high degrees and most



Distortion via Metrics



There is no “right” number for *method size*, *class size*, *amount of complexity*, etc

Why do we persist in our reductionism?

Maybe it is because we can’t think of anything better

"We want to establish key metrics and indicators to measure our progress toward the goal."

“Laws” of Metrics

“Laws” of Metrics

1. Distance Causes Misunderstanding

“Laws” of Metrics

1. Distance Causes Misunderstanding
2. Highlighting Leads to Focus

“Laws” of Metrics

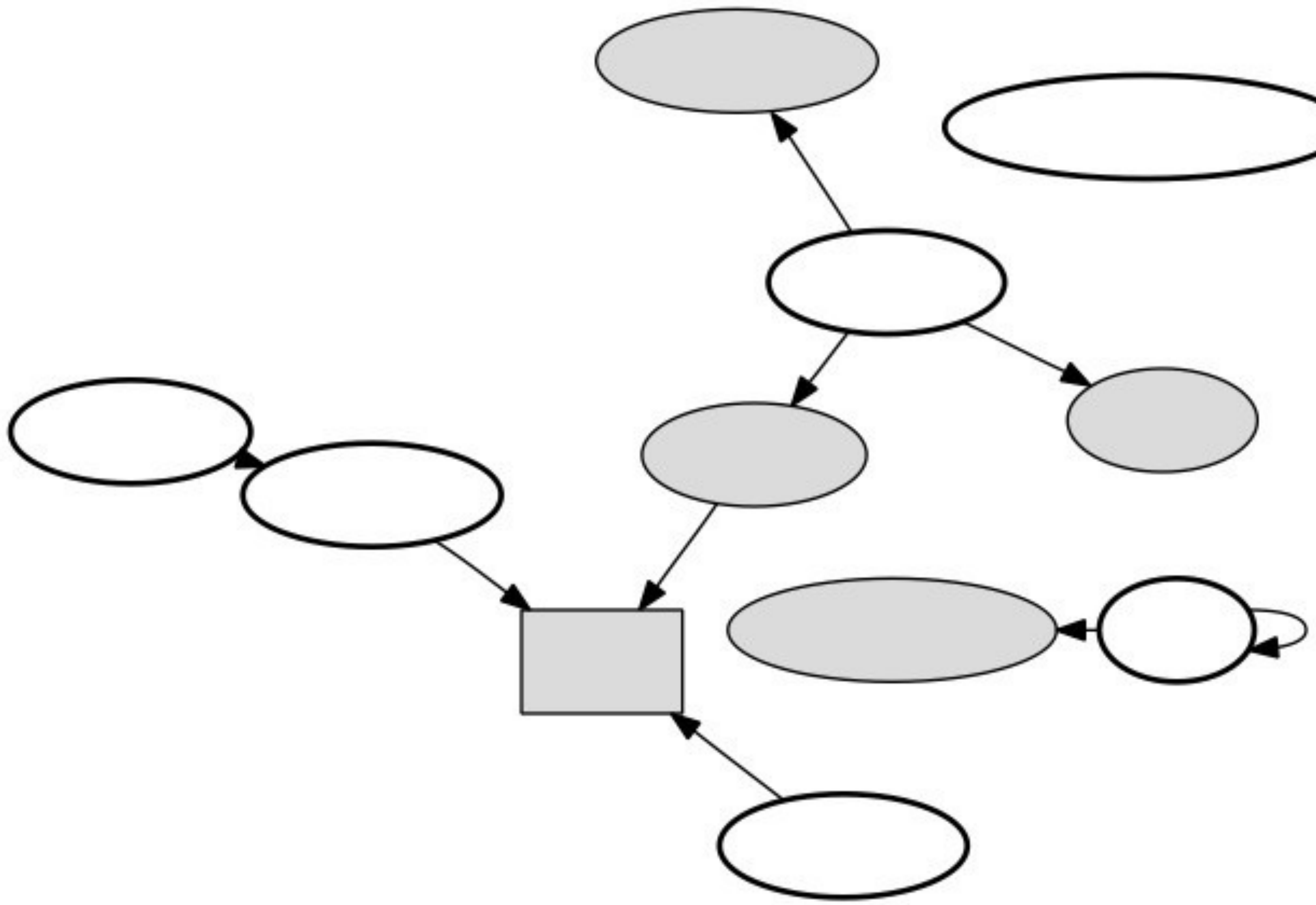
1. Distance Causes Misunderstanding
2. Highlighting Leads to Focus
3. Focus Leads to Action

“Laws” of Metrics

1. Distance Causes Misunderstanding
2. Highlighting Leads to Focus
3. Focus Leads to Action
4. Focus Leads to Side-Effects

Death of Locality

Use Qualitative “Measures” when Possible



Practice

Silent Alarms

Silent Alarms

Don’t have check-in gates. Let people make mistakes. Investigate the mistakes off-line and see why they happened. Then, intervene

Disposable Metrics

Disposable Metrics

Secondary effects are less likely when metrics come and go. Use them to highlight concerns

Practice

Targeted Metrics

Targeted Metric - Feature Trend Cards

Hypothesize a couple of features that you will never add to your code. Task them and estimate them periodically. See the debt trend for areas they touch.

Deluge with Metrics

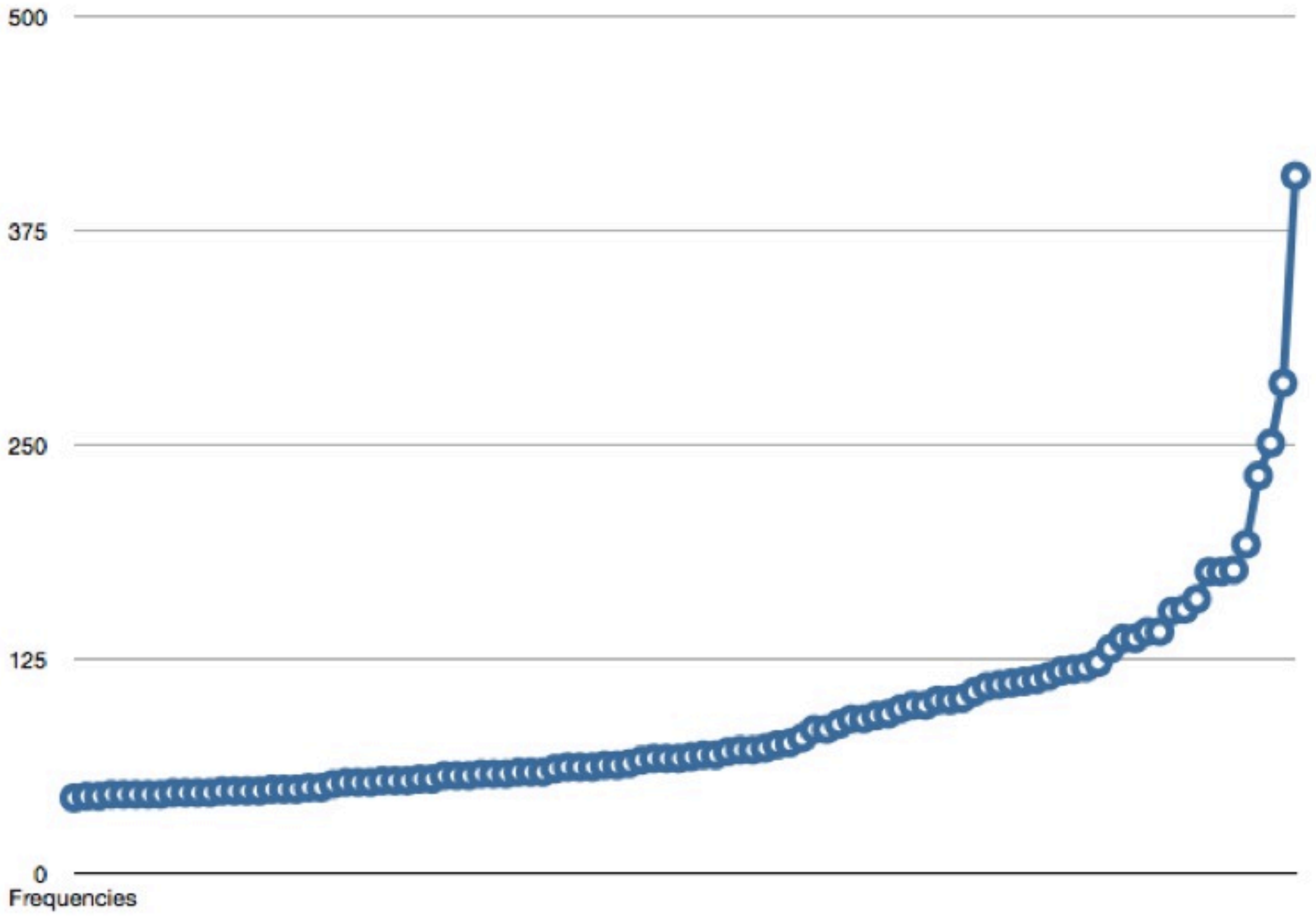
Deluge

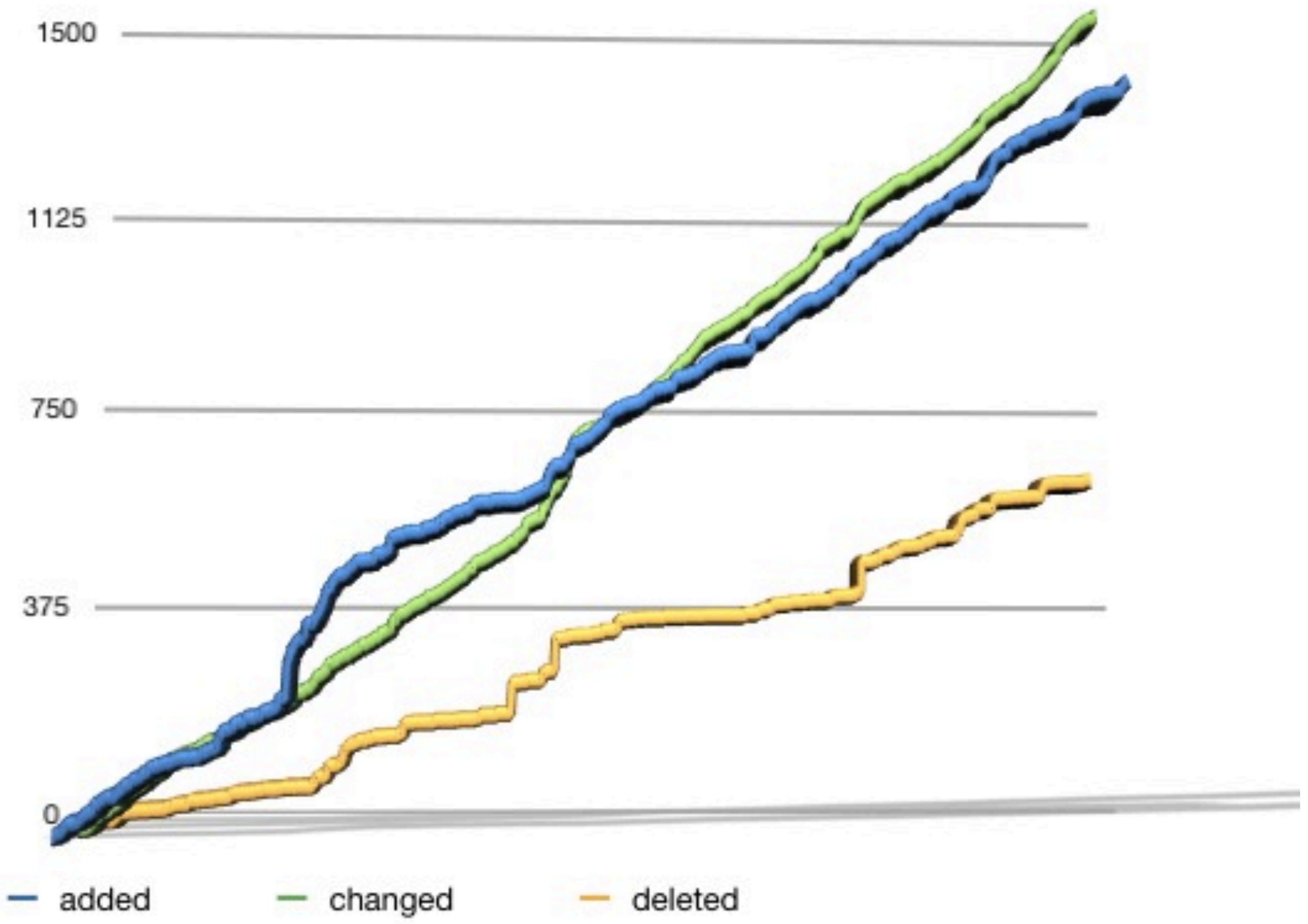
The more metrics you have, the harder it is to take any one of them too seriously. They become "vital signs" and indicators as we have in medicine.

Temporal Correlation of Class Changes

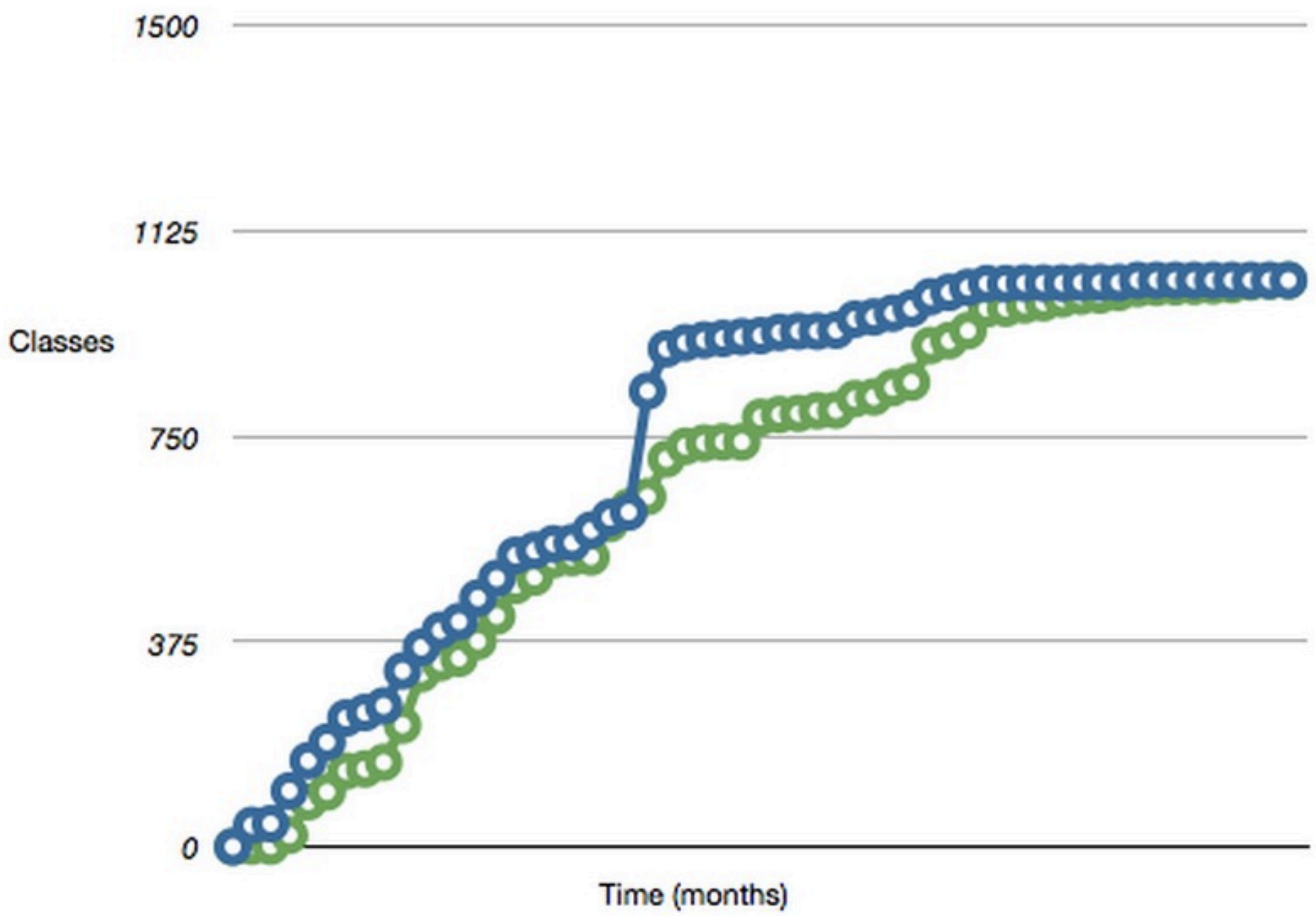
```
events.group_by {|e| [e.day,e.committer]}.values
  .map {|e| e.map(&:class_name).uniq.combination(2).to_a }
  .flatten(1).norm_pairs.freq_by {|e| e }.sort_by {|p| p[1] }
```

When you examine these sorts of frequencies, they typically have that power law-ish shape:

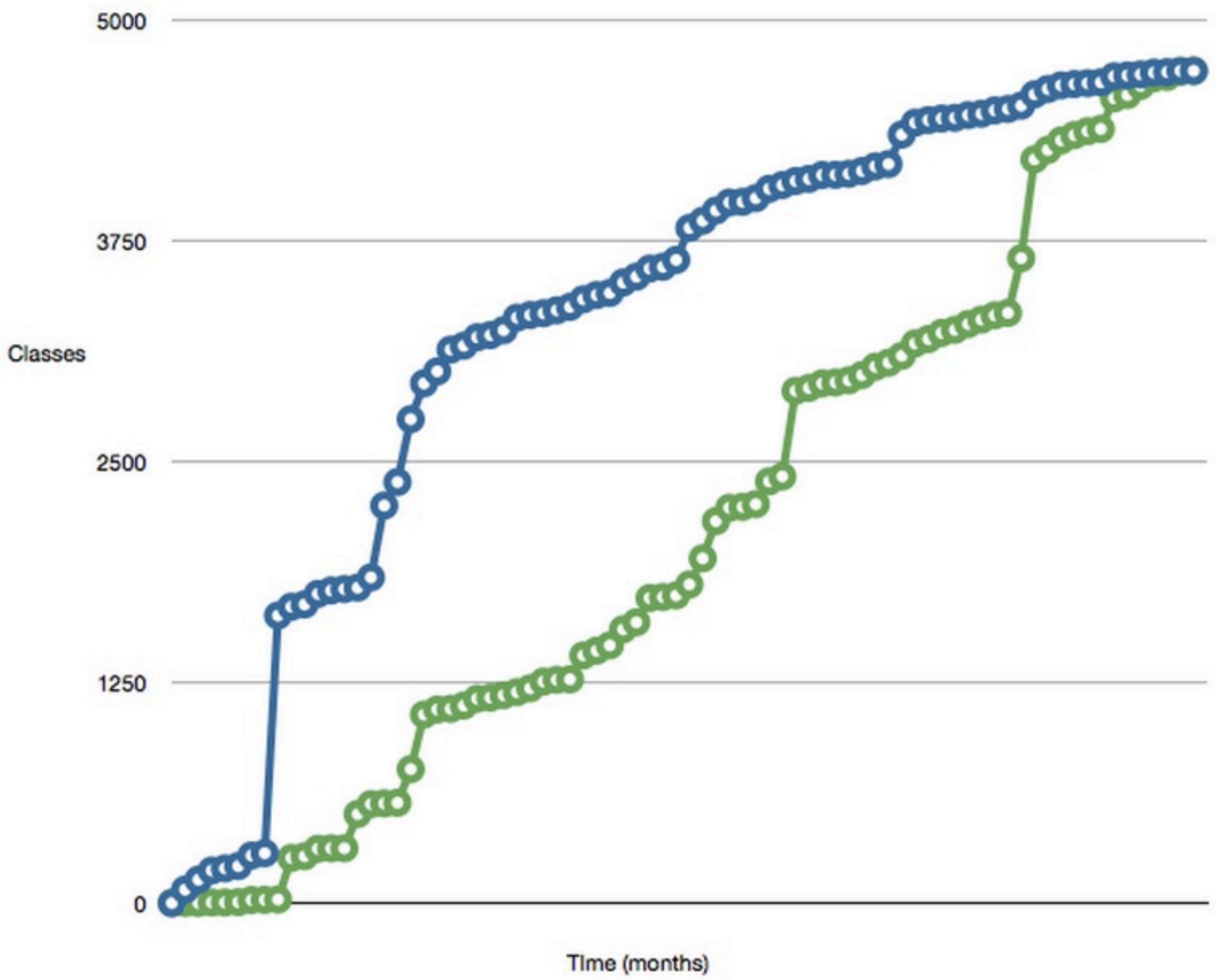



1500
1125
750
375
0
added
changed
deleted

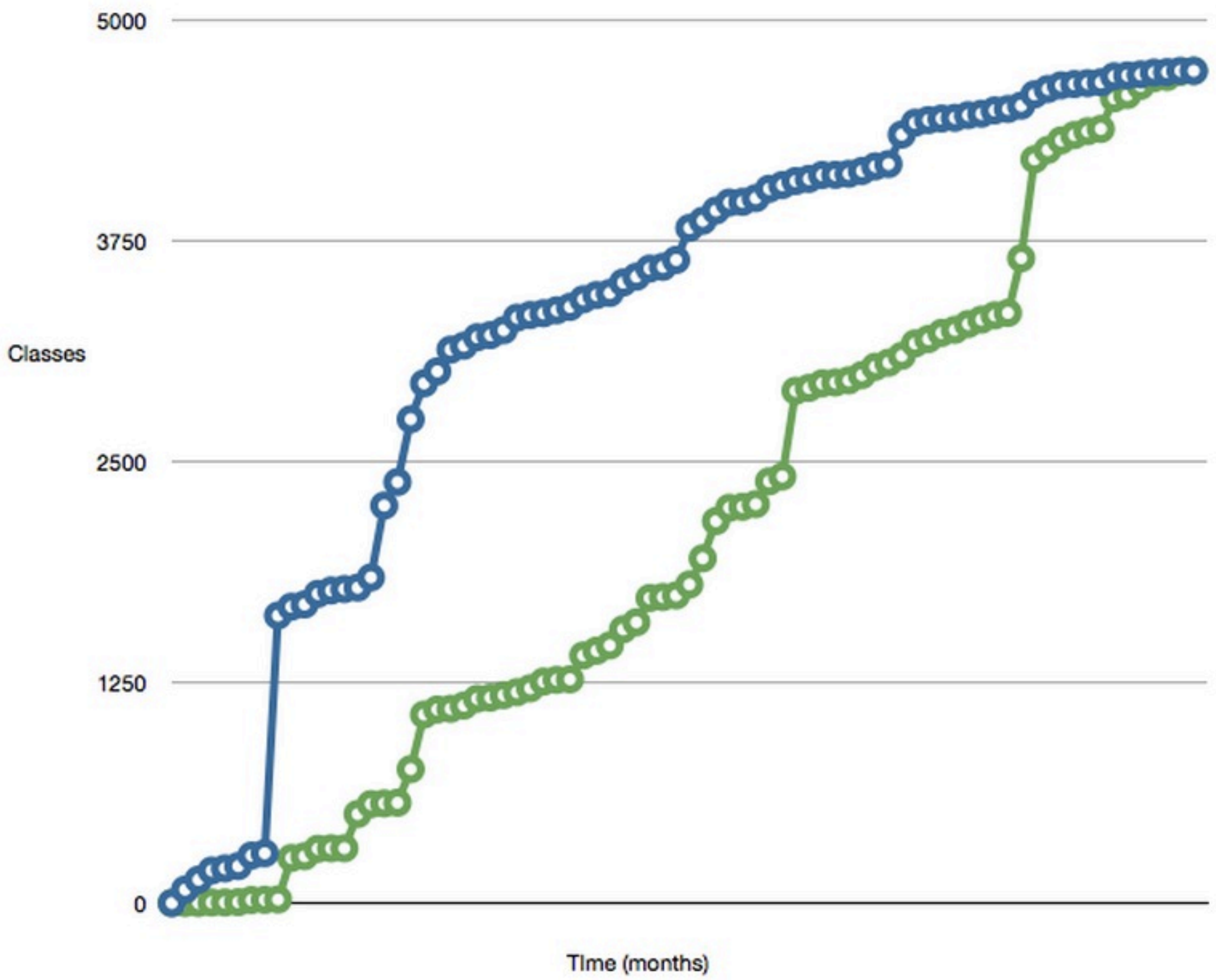
Active Set of Classes



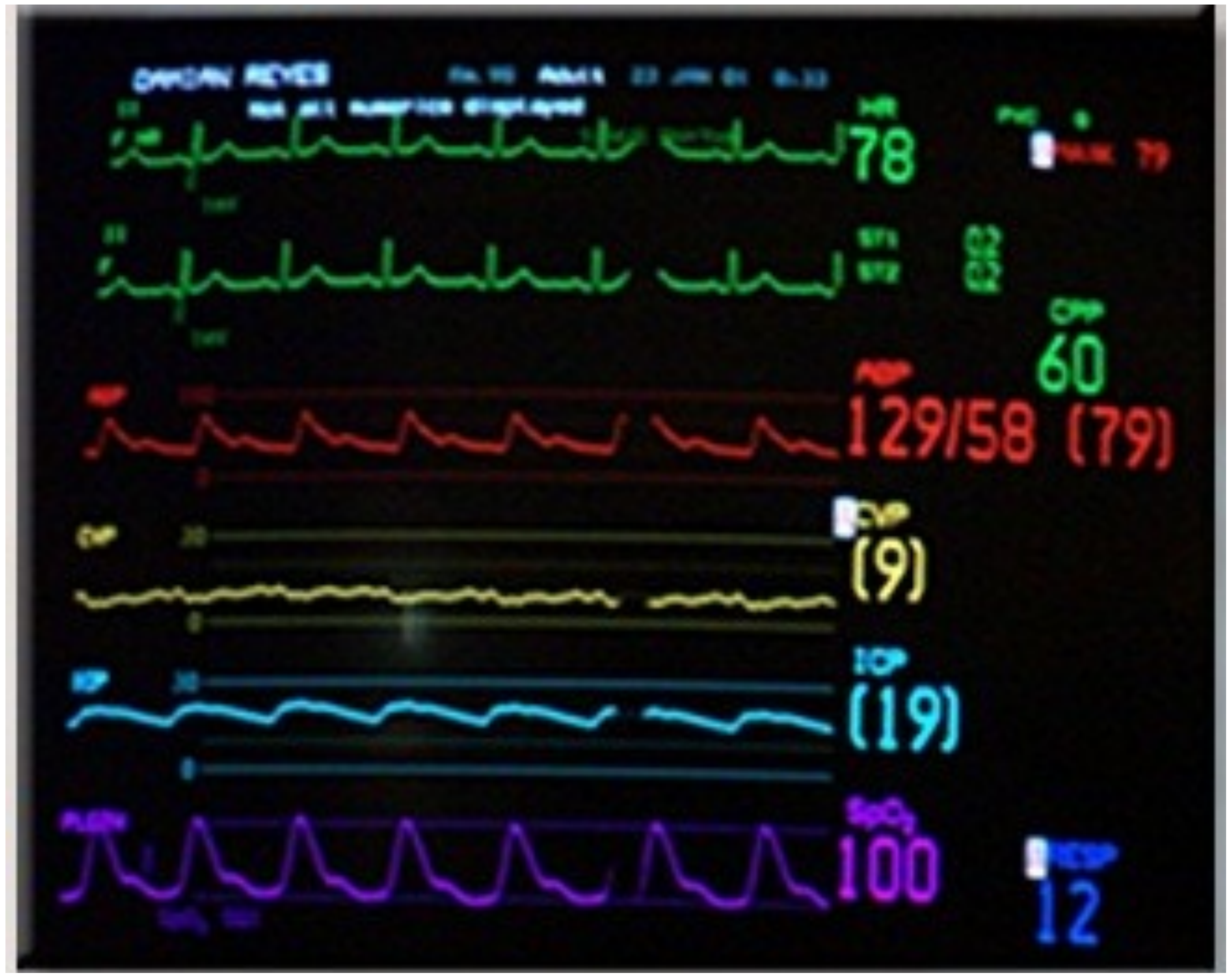
Active Set of Classes



Active Set of Classes



Vital Signs



Reduction is the Enemy