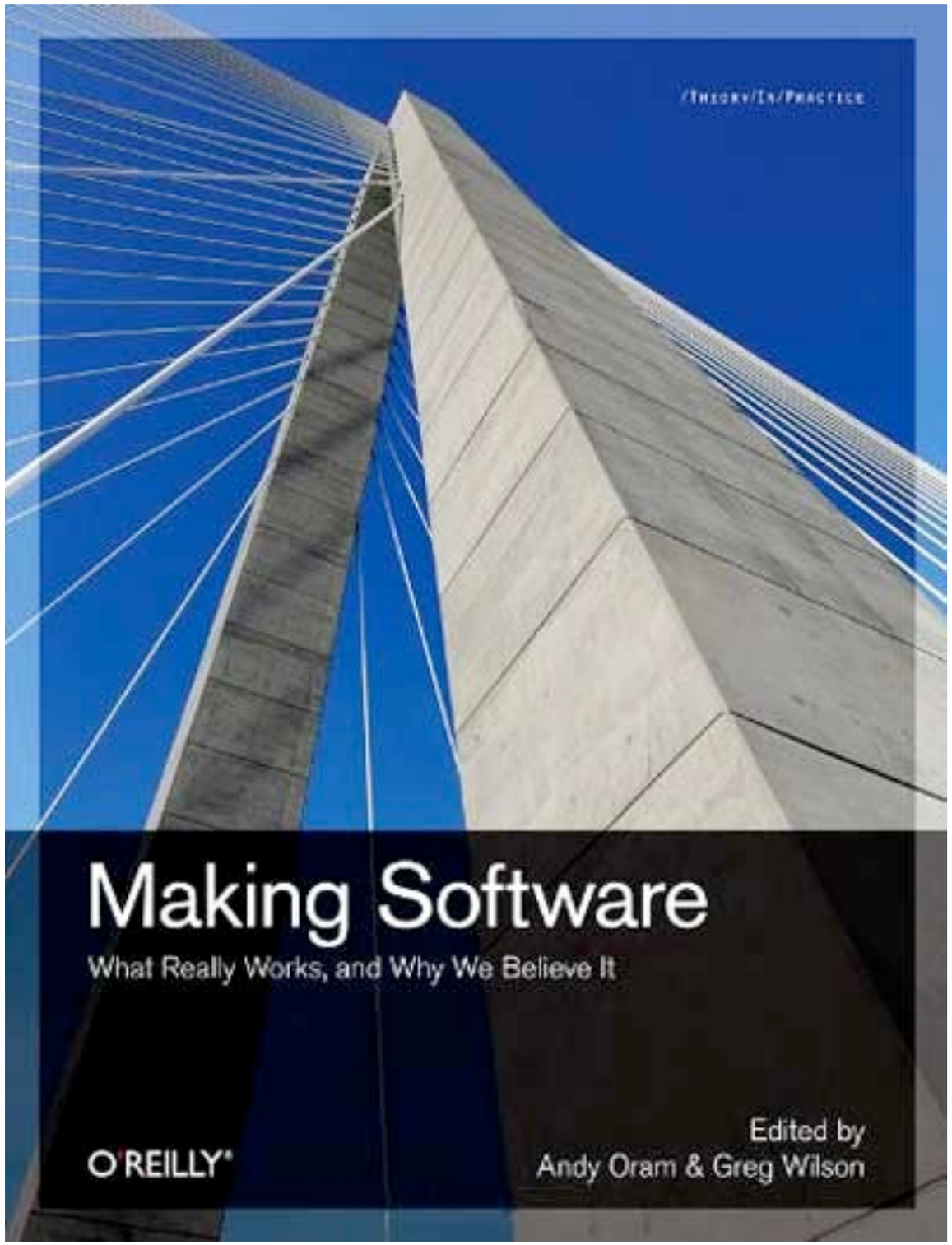
Software Naturalism

Embracing the Real Behind the Ideal

Michael Feathers
Groupon

How Much Do We Know?


/Theory/In/Practice
Making Software
What Really Works, and Why We Believe It
O'REILLY
Edited by
Andy Oram & Greg Wilson

Fault Prediction System - Elaine J. Weyuker, Thomas J. Ostrand

Models for Project Managers, to help them decide where to apply effort.

Research used 6 large projects built or contracted by AT&T

- 300-500 KSLOC
- 2 yrs to 10 yrs
- 4-50 langs per system
- 3 month release cycle

Validated the hypothesis that distribution of faults across files is Pareto:

> 80% faults are in < 20% of files

Table 9-2. Percentage of faults in top 20% of files for previously studied systems

System	Final release files	Final release KLOC	% faults in top 20% files
Inventory	1950	538	83%
Provisioning	2308	438	83%
Voice Response	1888	329	75%
Maintenance Support A	668	442	81%

Inputs to Prediction Model (per file):

- LOC
- New file (y/n)?
- number of changes in release N-1
- number of changes in release N-2
- number of faults in release N-1
- programming language

Experimented with (per file):

- Cumulative # of developers
- Recent # of developers (Release N-1)
- Number of new developers

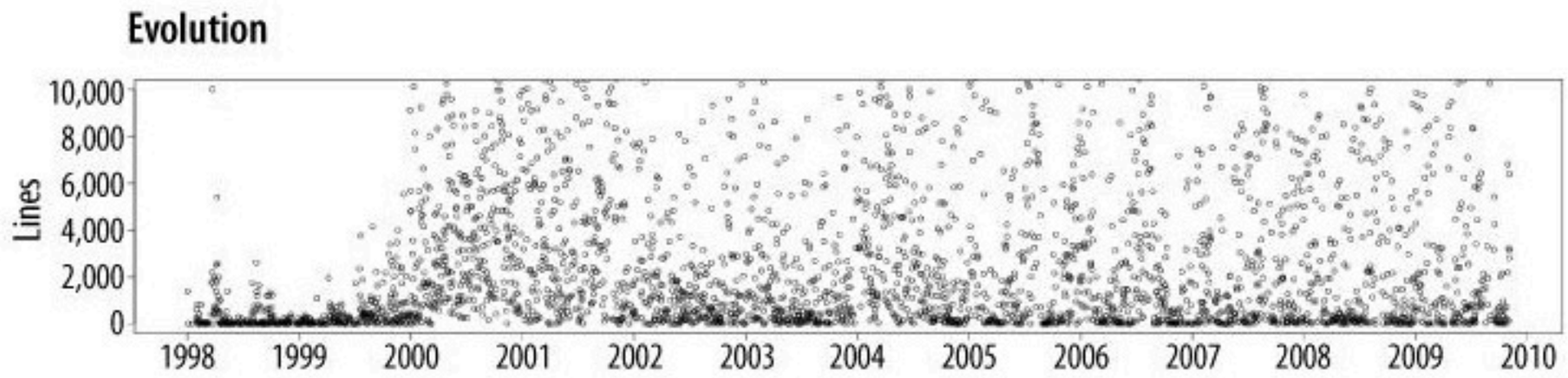
Experimented with (per file):

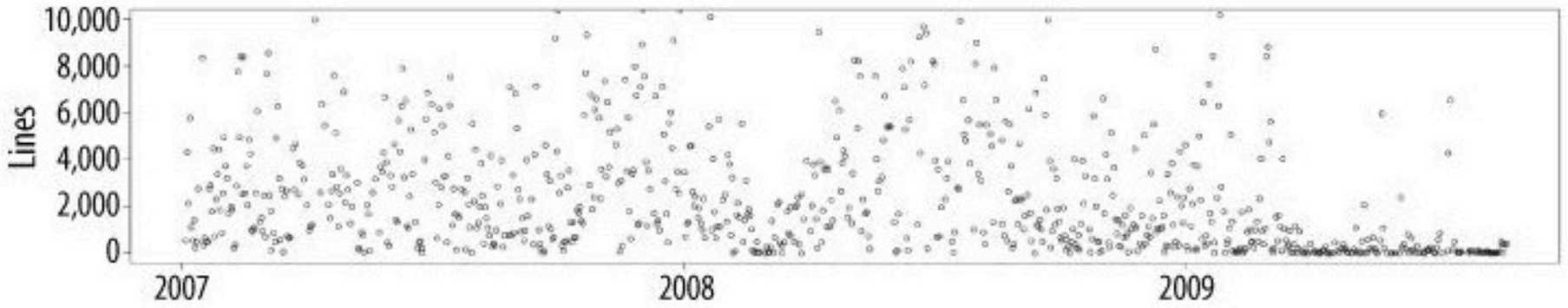
- Cumulative # of developers
- Recent # of developers (Release N-1)
- Number of new developers

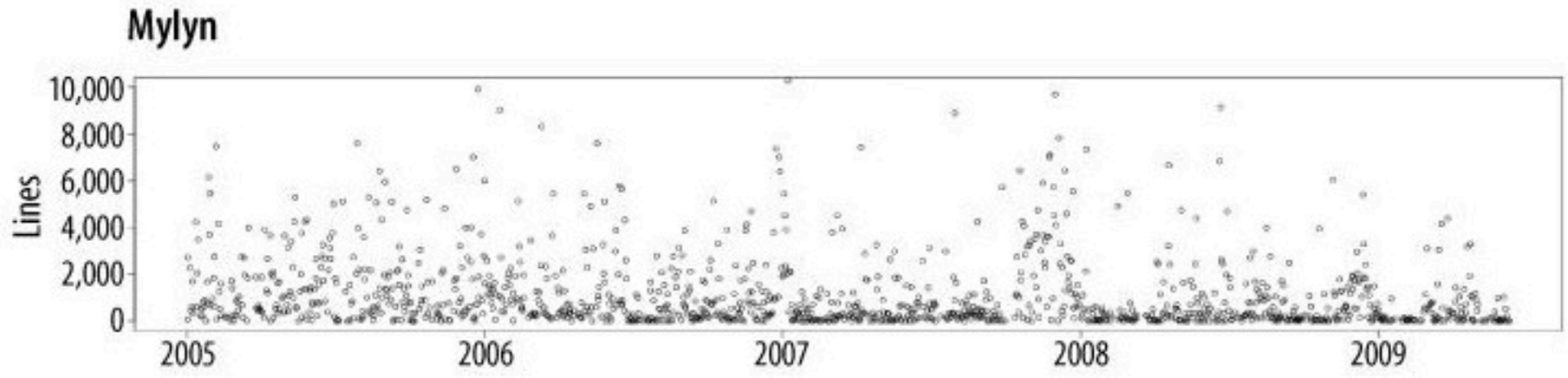
Dev-based Inputs did not improve the model much

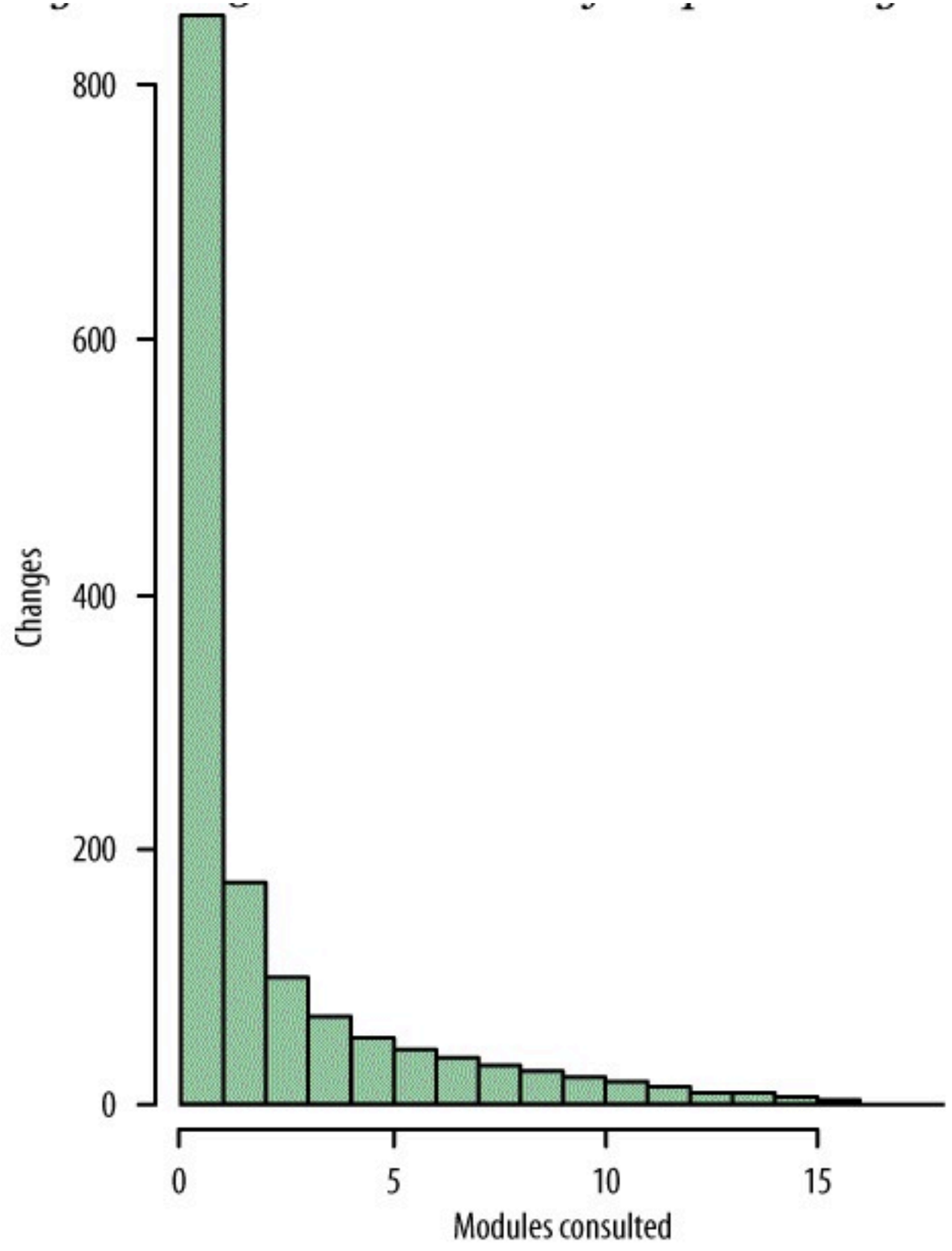
How Effective Is Modularization? - Neil Thomas and Gail Murphy

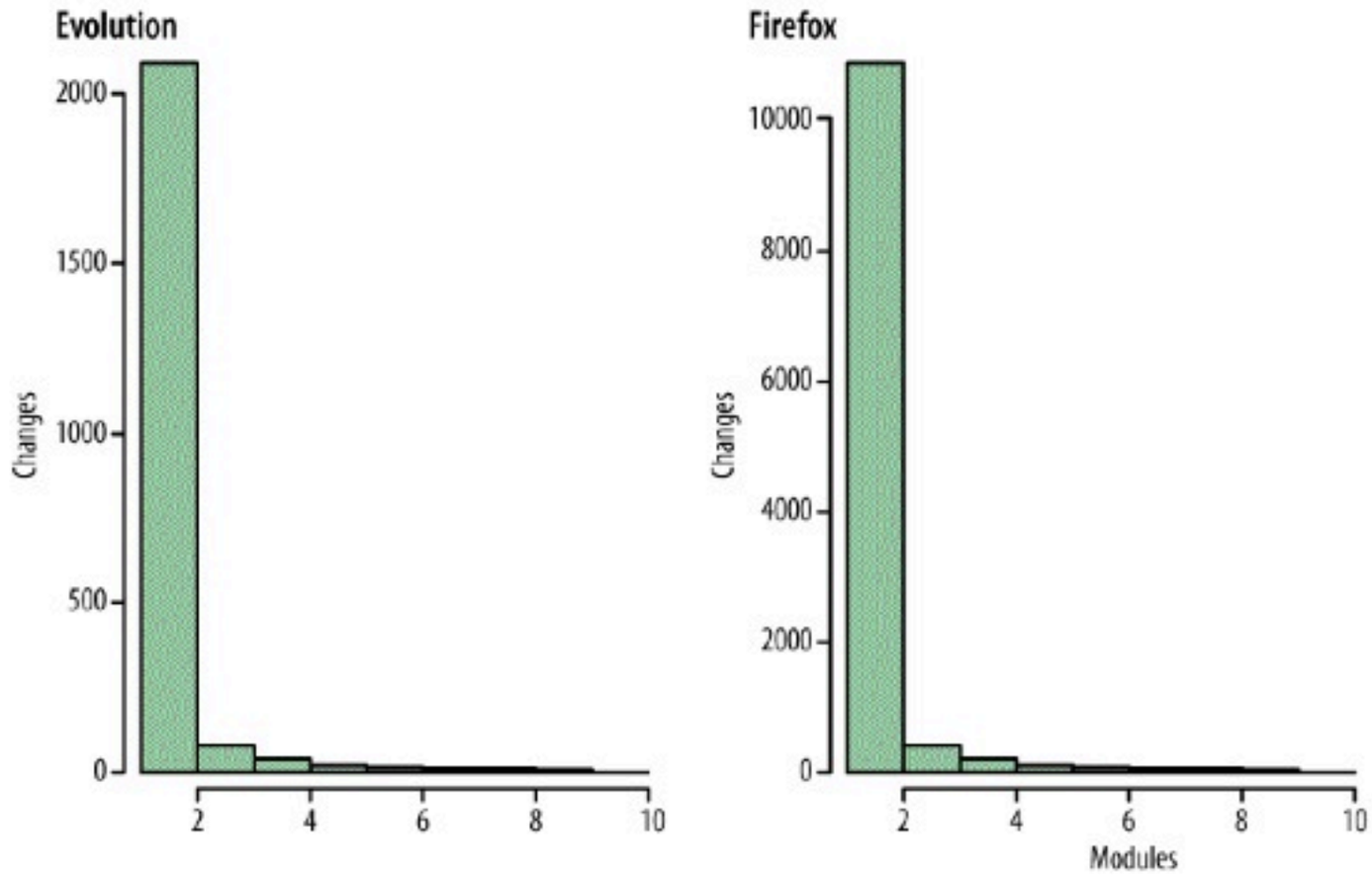
- Are most changes made to the code of a system during a single bug fix or enhancement constrained to a single module?
- When a software developer makes a change to the code of a system, must the developer consult code in other modules?
- Do the patterns in the actual changes and modules consulted suggest a different modular breakdown for a system?

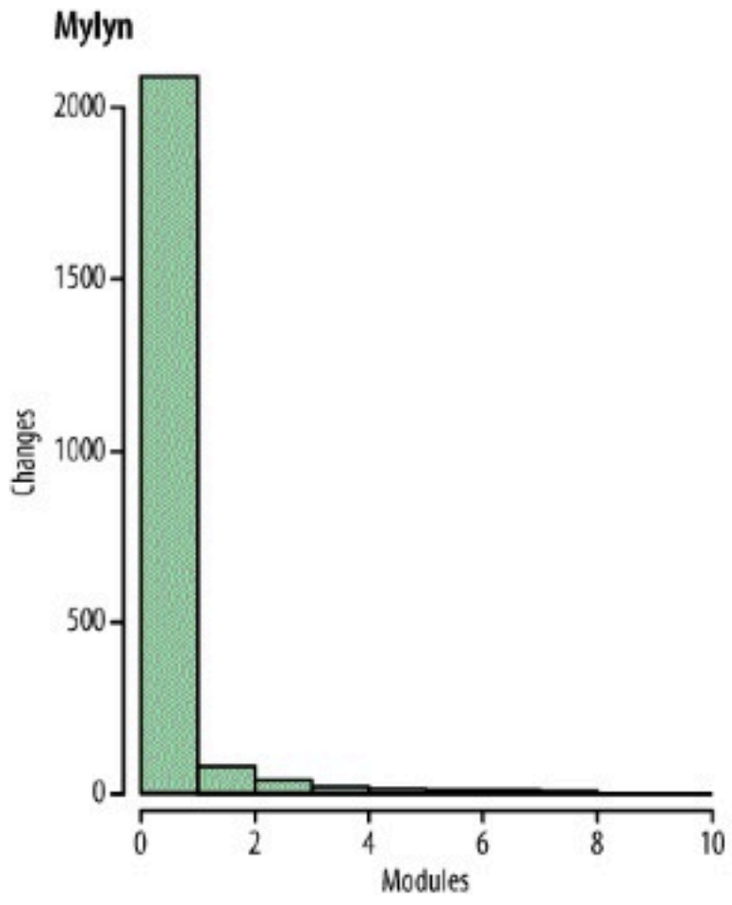

Evolution
Lines
10,000
8,000
6,000
4,000
2,000
0
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010


Firefox
Lines
10,000
8,000
6,000
4,000
2,000
0
2007
2008
2009


Mylyn
Lines
10,000
8,000
6,000
4,000
2,000
0
2005
2006
2007
2008
2009


800
600
400
200
0
Changes
0
5
10
15
Modules consulted


Evolution
Changes
2000
1500
1000
500
0
2
4
6
8
10
Firefox
Changes
10000
8000
6000
4000
2000
0
2
4
6
8
10
Modules


Mylyn
Changes
2000
1500
1000
500
0
0
2
4
6
8
10
Modules

Change Locality

We begin our exploration with a simple question: are most changes to the code constrained to a single module? Figure 21-5 shows a histogram of the number of modules modified per change for each system, and Table 21-4 presents some summary statistics.

Table 21-4. Number of modules affected by each change

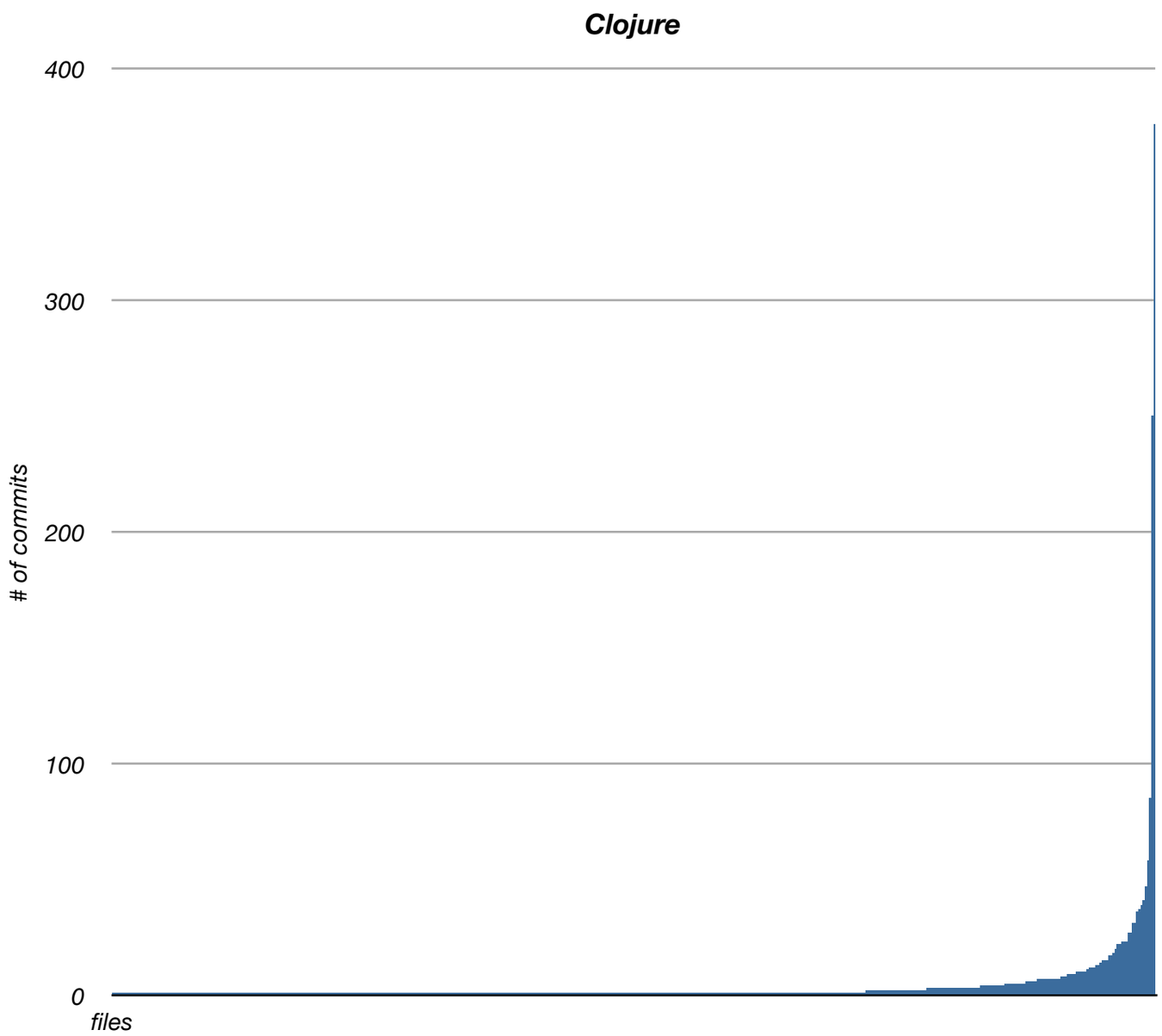
Project	% changes affecting only one module	Mean modules affected
Evolution	86.6%	1.243
Firefox	73.7%	1.577
Mylyn	69.7%	1.634

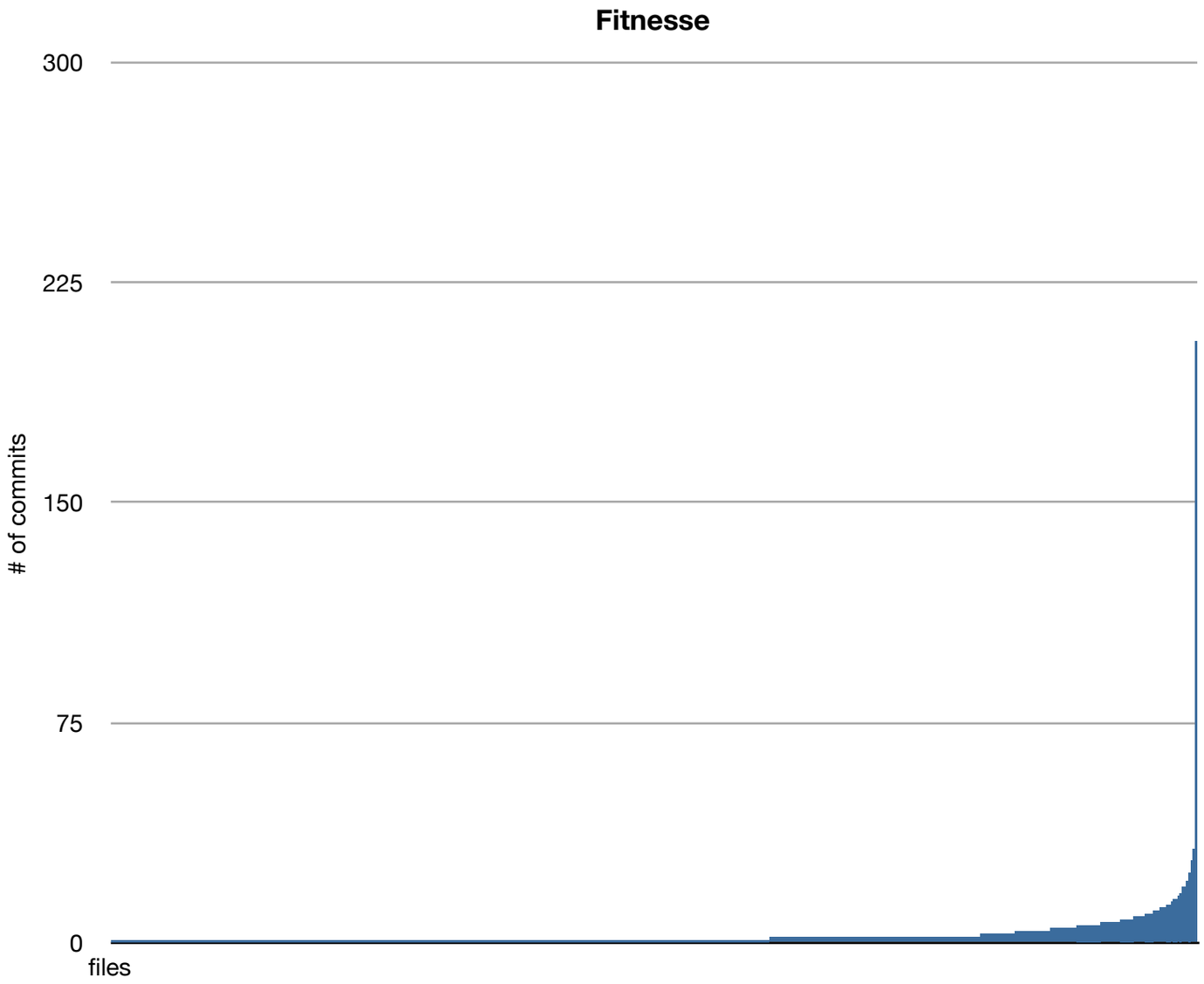
The Open/Closed Principle

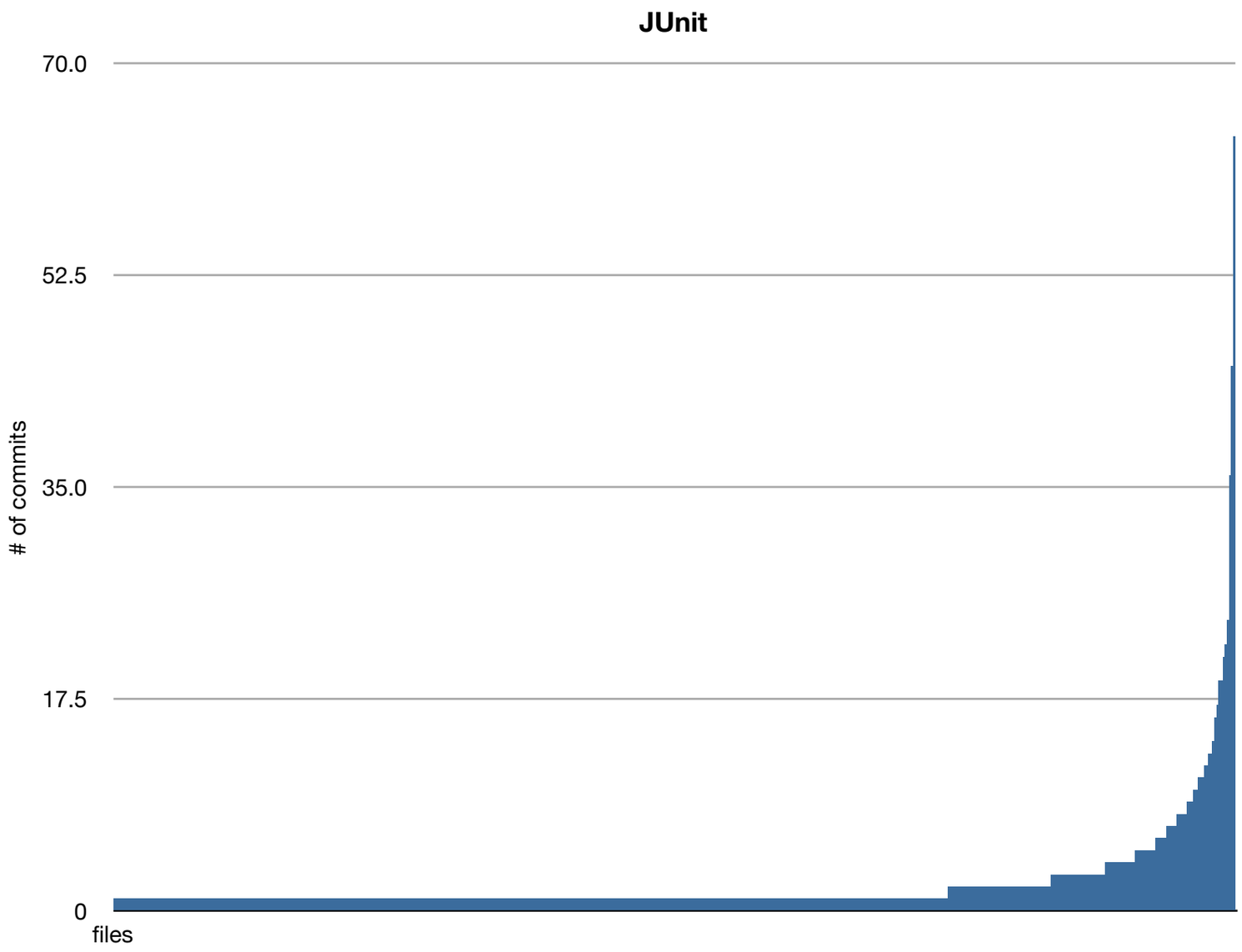
"*software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification*"

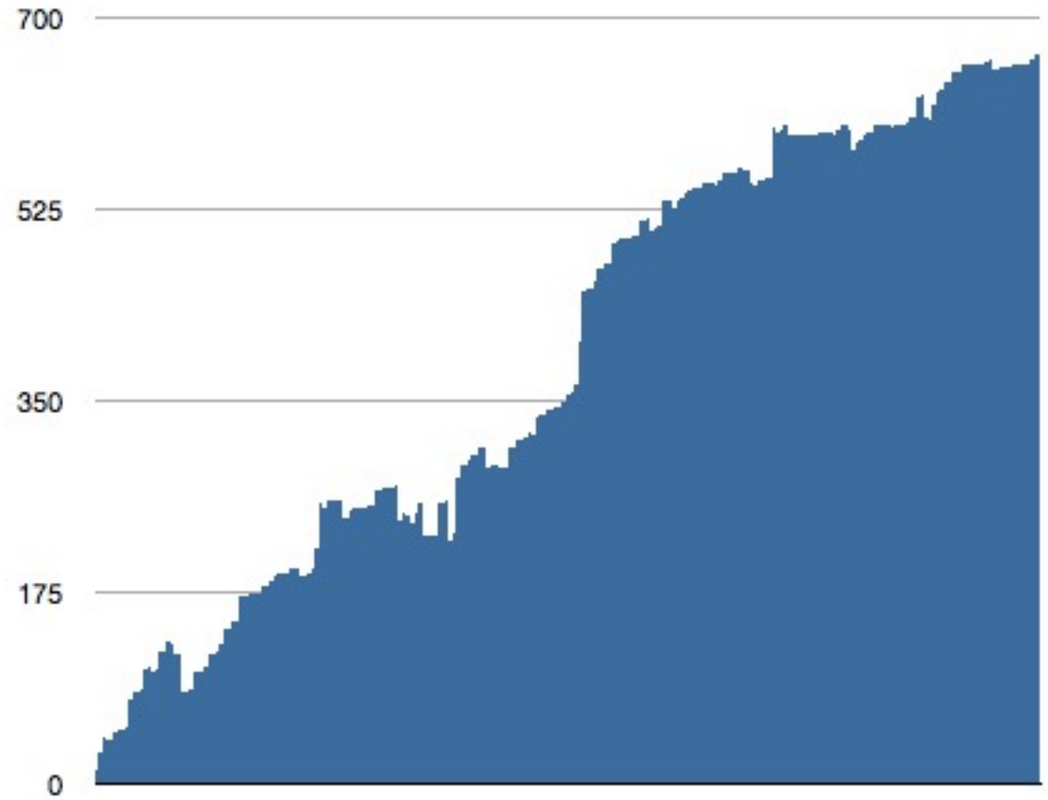
- Bertrand Meyer

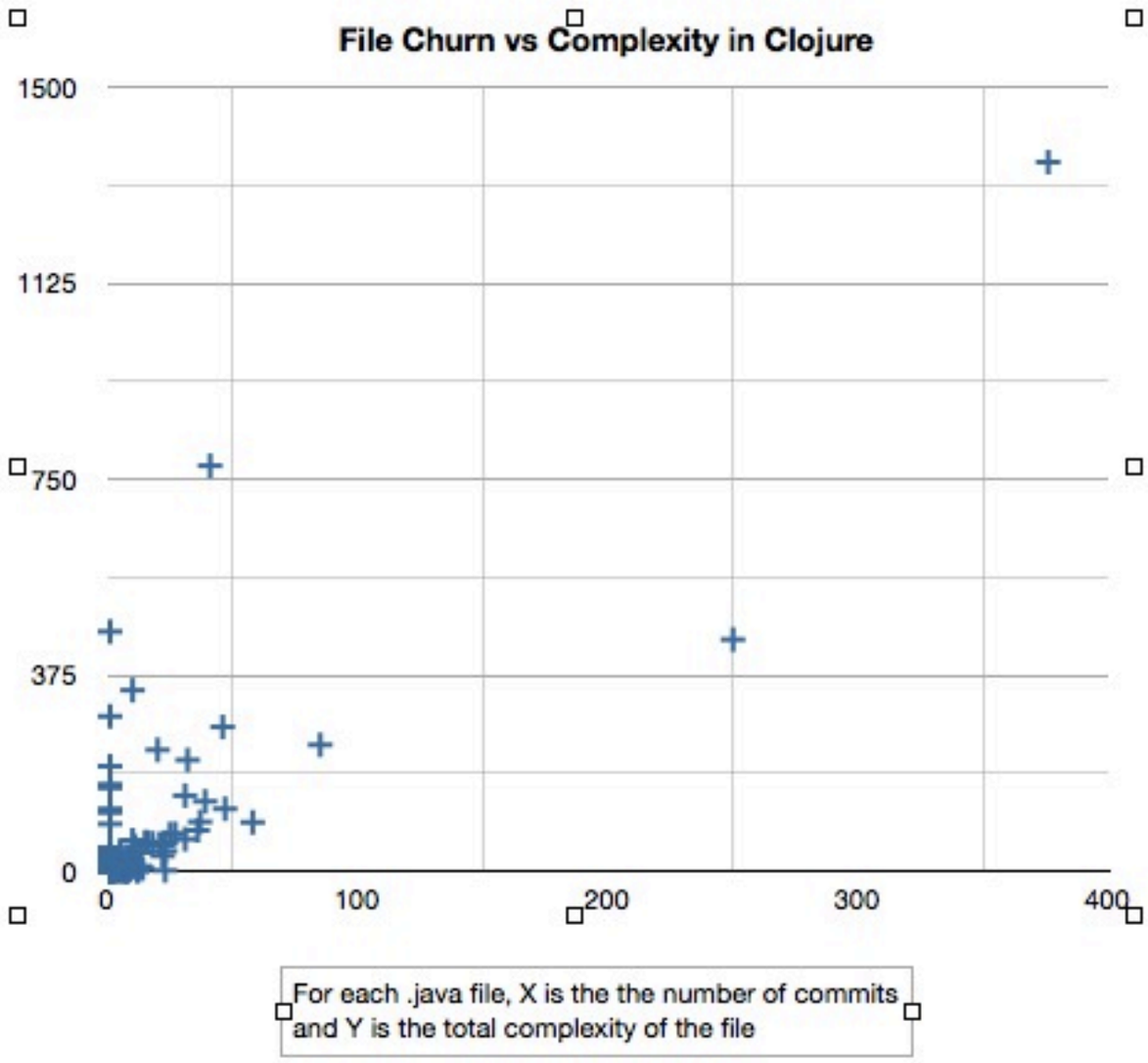

OPEN CLOSED PRINCIPLE
Open Chest Surgery Is Not Needed When Putting On A Coat

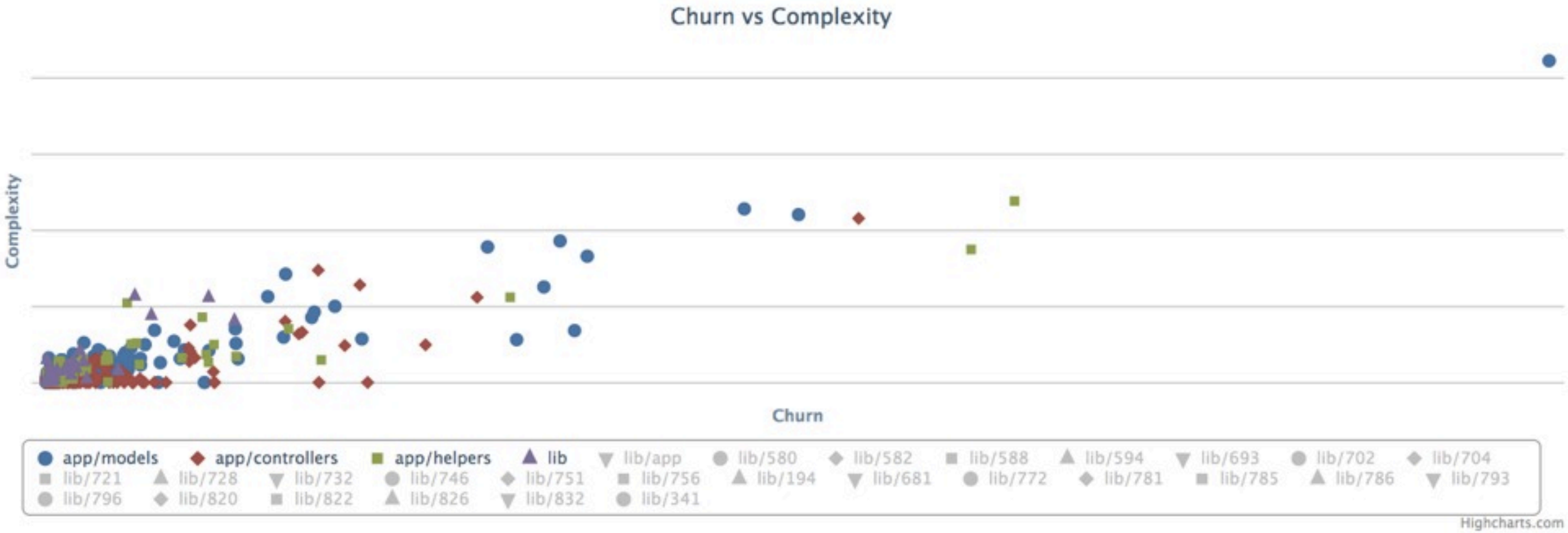

Clojure
400
300
200
100
0
# of commits
files

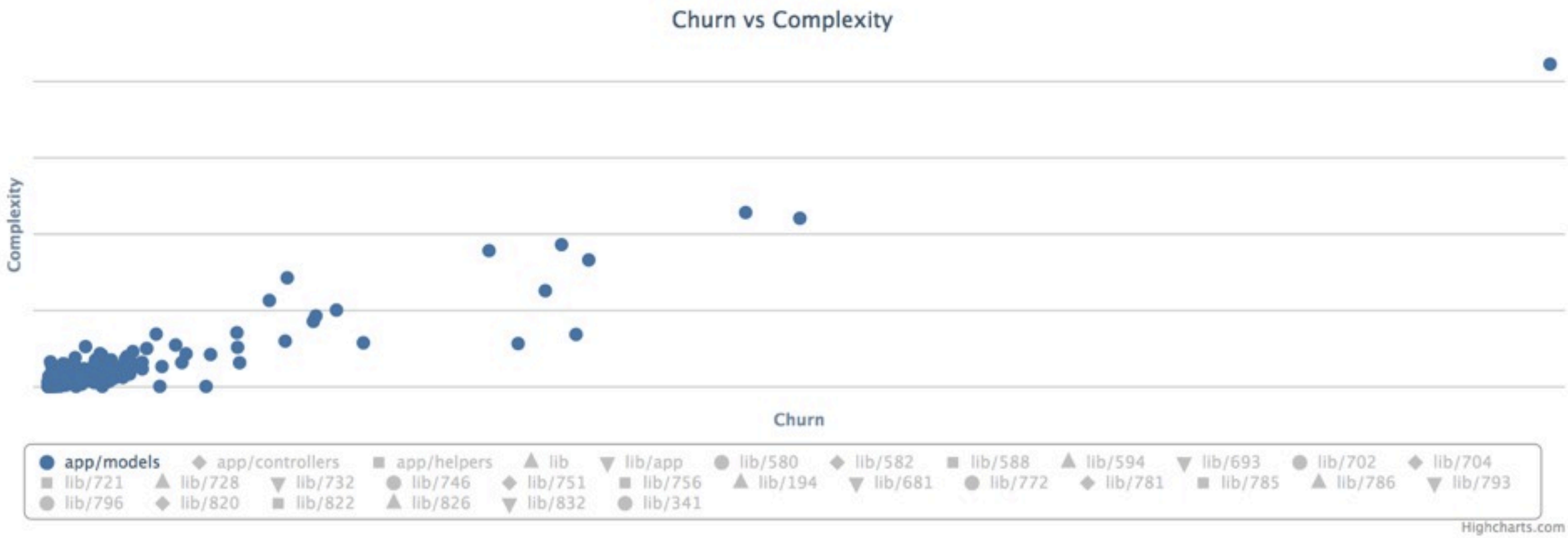

Fitnesse
300
225
150
75
0
# of commits
files

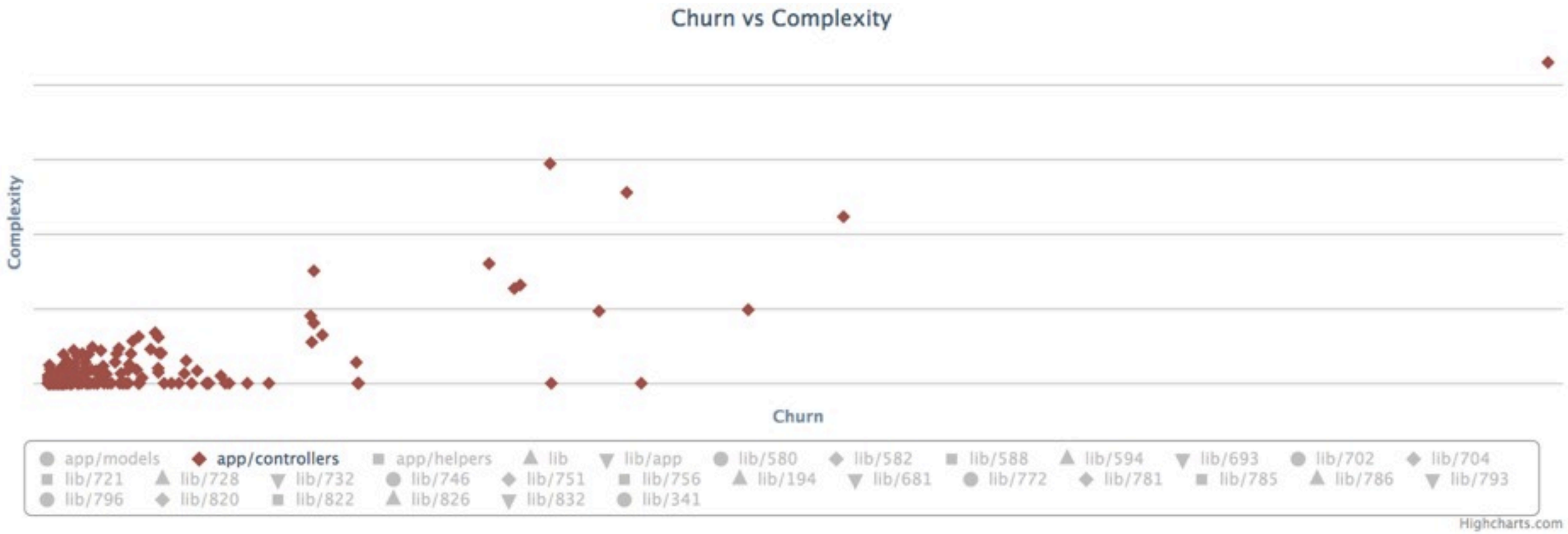

JUnit
70.0
52.5
35.0
17.5
0
# of commits
files


700
525
350
175
0

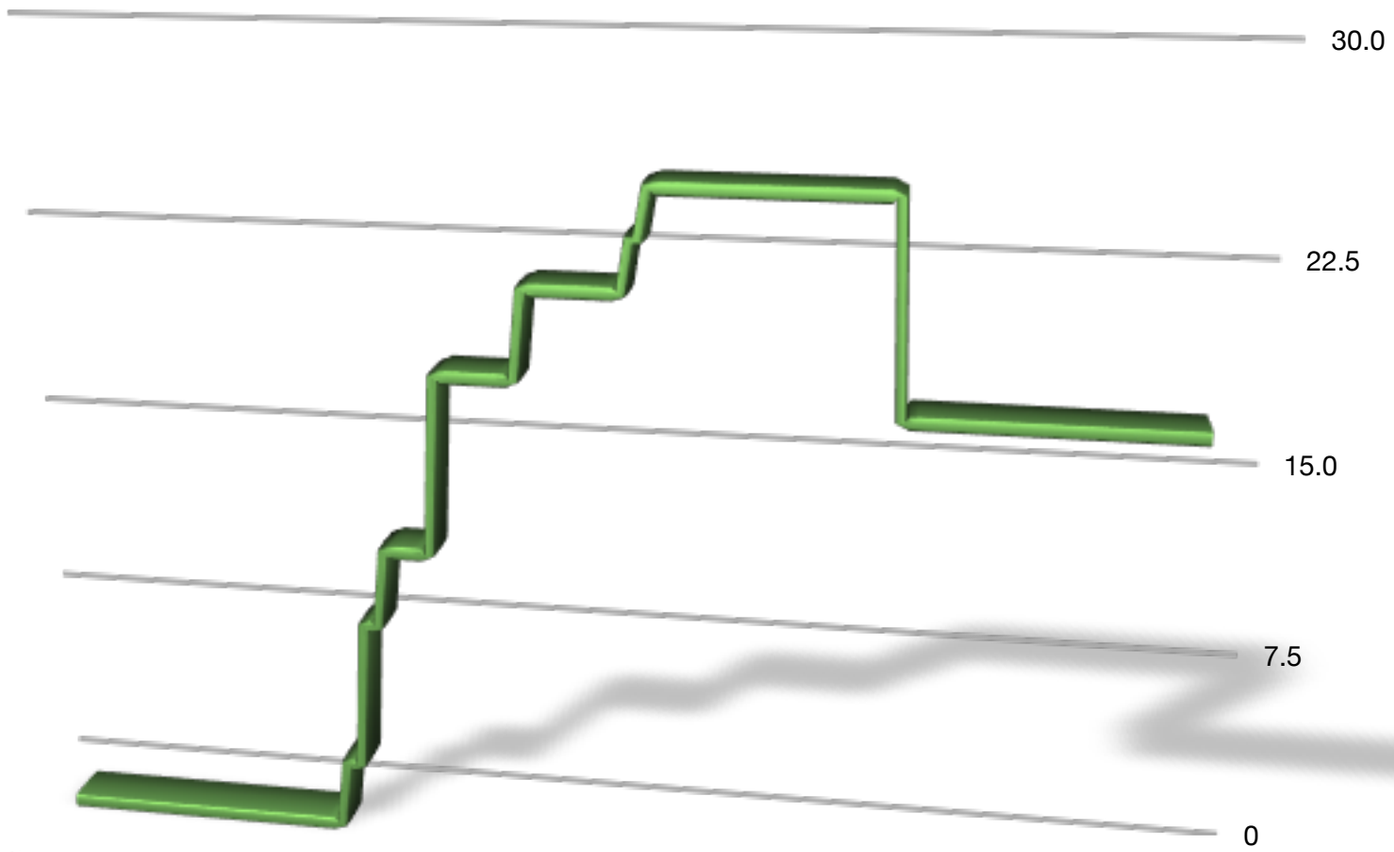

File Churn vs Complexity in Clojure
1500
1125
750
375
0
0
100
200
300
400
For each .java file, X is the the number of commits and Y is the total complexity of the file


Churn vs Complexity
Complexity
Churn
app/models
app/controllers
app/helpers
lib
lib/app
lib/580
lib/582
lib/588
lib/594
lib/693
lib/702
lib/704
lib/721
lib/728
lib/732
lib/746
lib/751
lib/756
lib/194
lib/681
lib/772
lib/781
lib/785
lib/786
lib/793
lib/796
lib/820
lib/822
lib/826
lib/832
lib/341
Highcharts.com

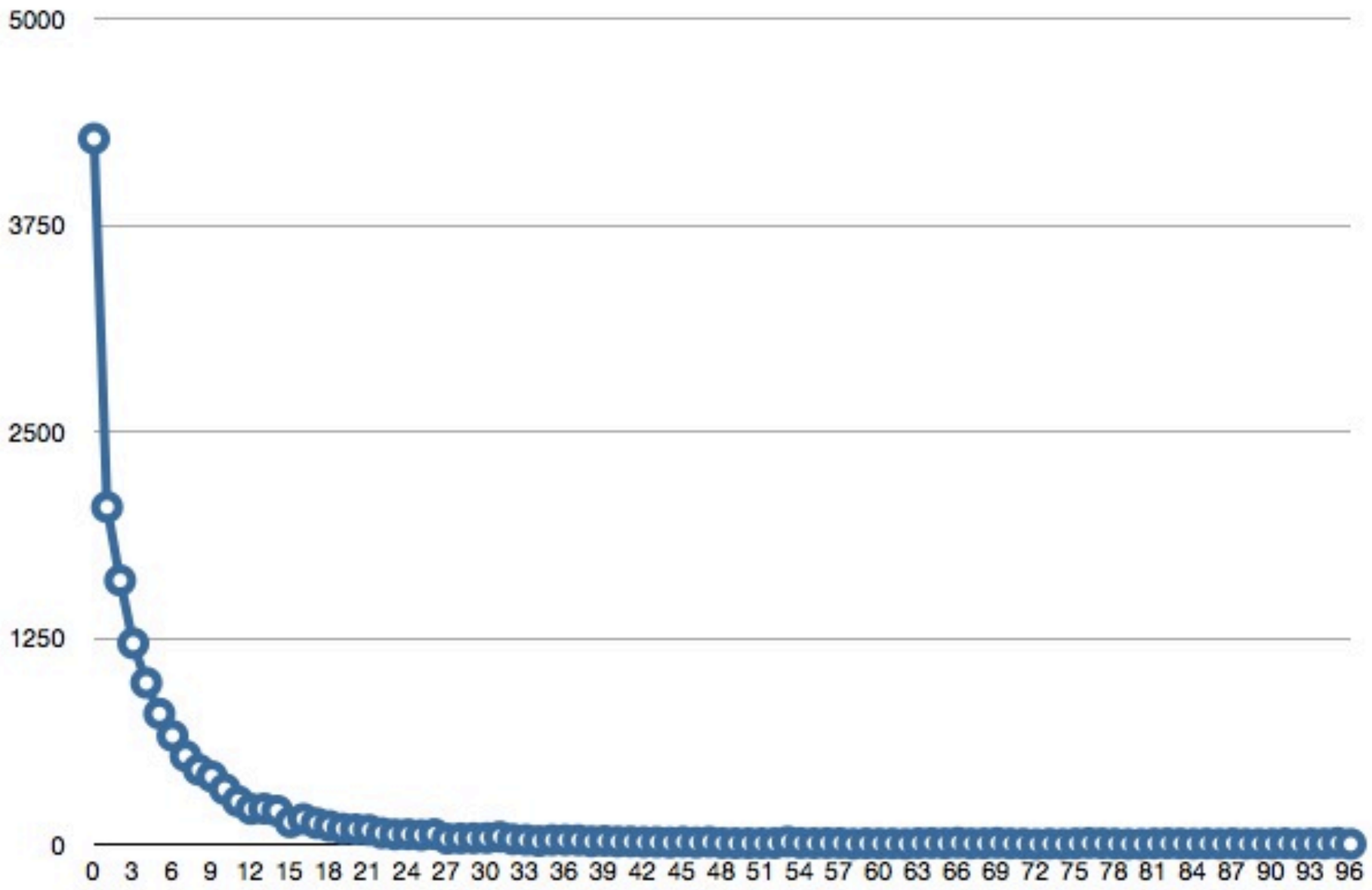

Churn vs Complexity
Complexity
Churn
app/models
app/controllers
app/helpers
lib
lib/app
lib/580
lib/582
lib/588
lib/594
lib/693
lib/702
lib/704
lib/721
lib/728
lib/732
lib/746
lib/751
lib/756
lib/194
lib/681
lib/772
lib/781
lib/785
lib/786
lib/793
lib/796
lib/820
lib/822
lib/826
lib/832
lib/341
Highcharts.com


Churn vs Complexity
Complexity
Churn
app/models
app/controllers
app/helpers
lib
lib/app
lib/580
lib/582
lib/588
lib/594
lib/693
lib/702
lib/704
lib/721
lib/728
lib/732
lib/746
lib/751
lib/756
lib/194
lib/681
lib/772
lib/781
lib/785
lib/786
lib/793
lib/796
lib/820
lib/822
lib/826
lib/832
lib/341
Highcharts.com

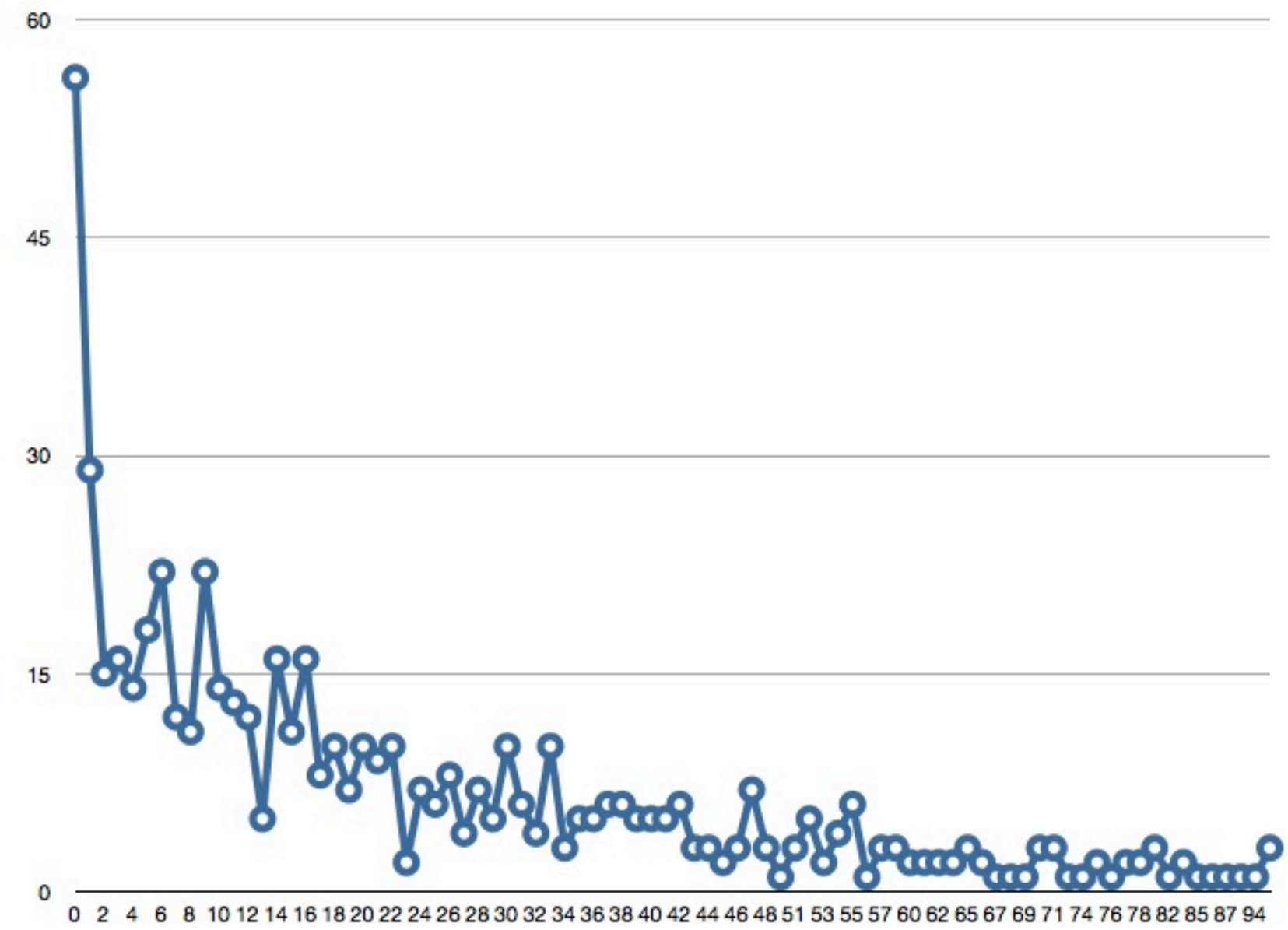
A Method Lifeline



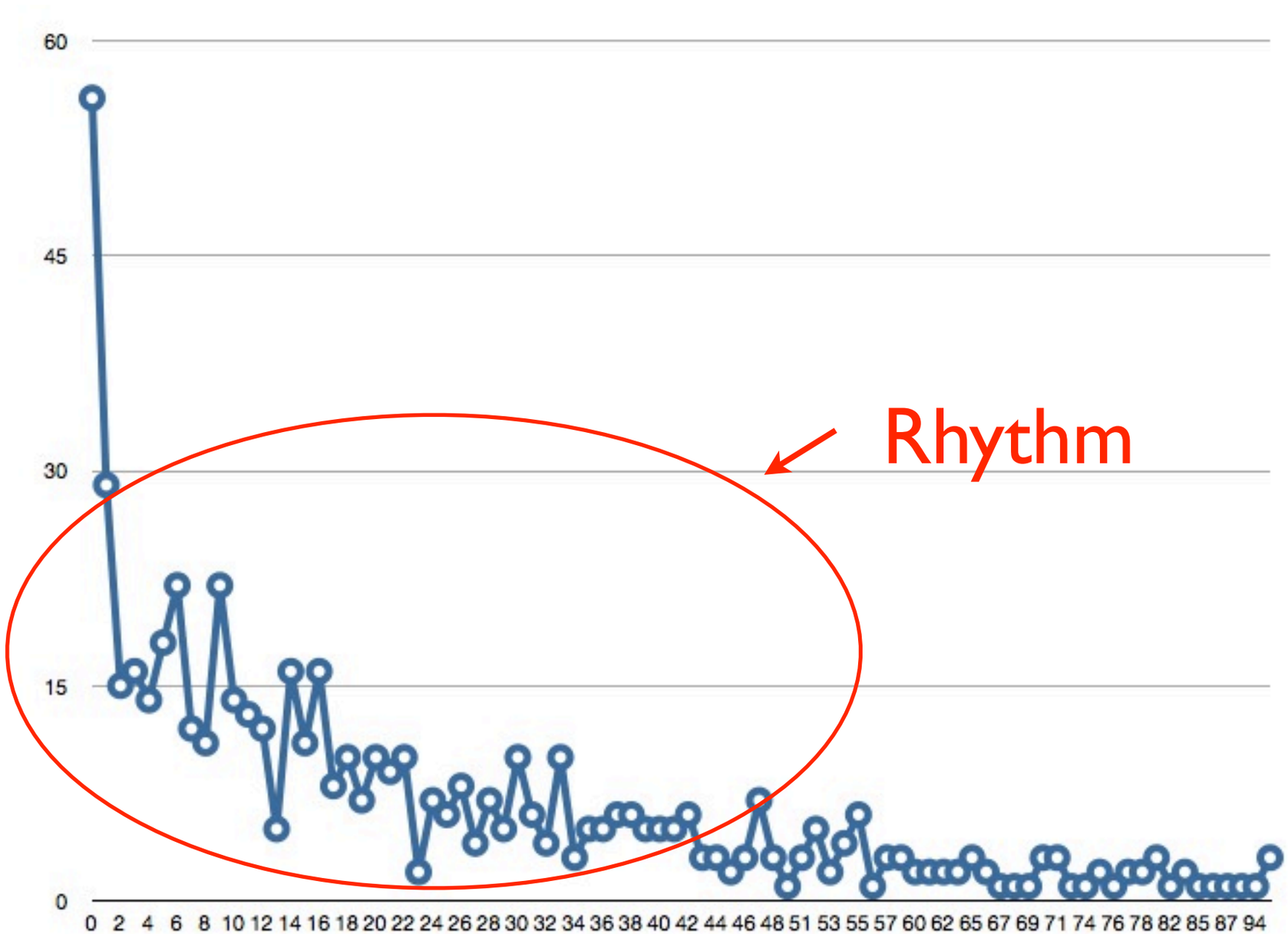
Frequency of Inter-commit Intervals



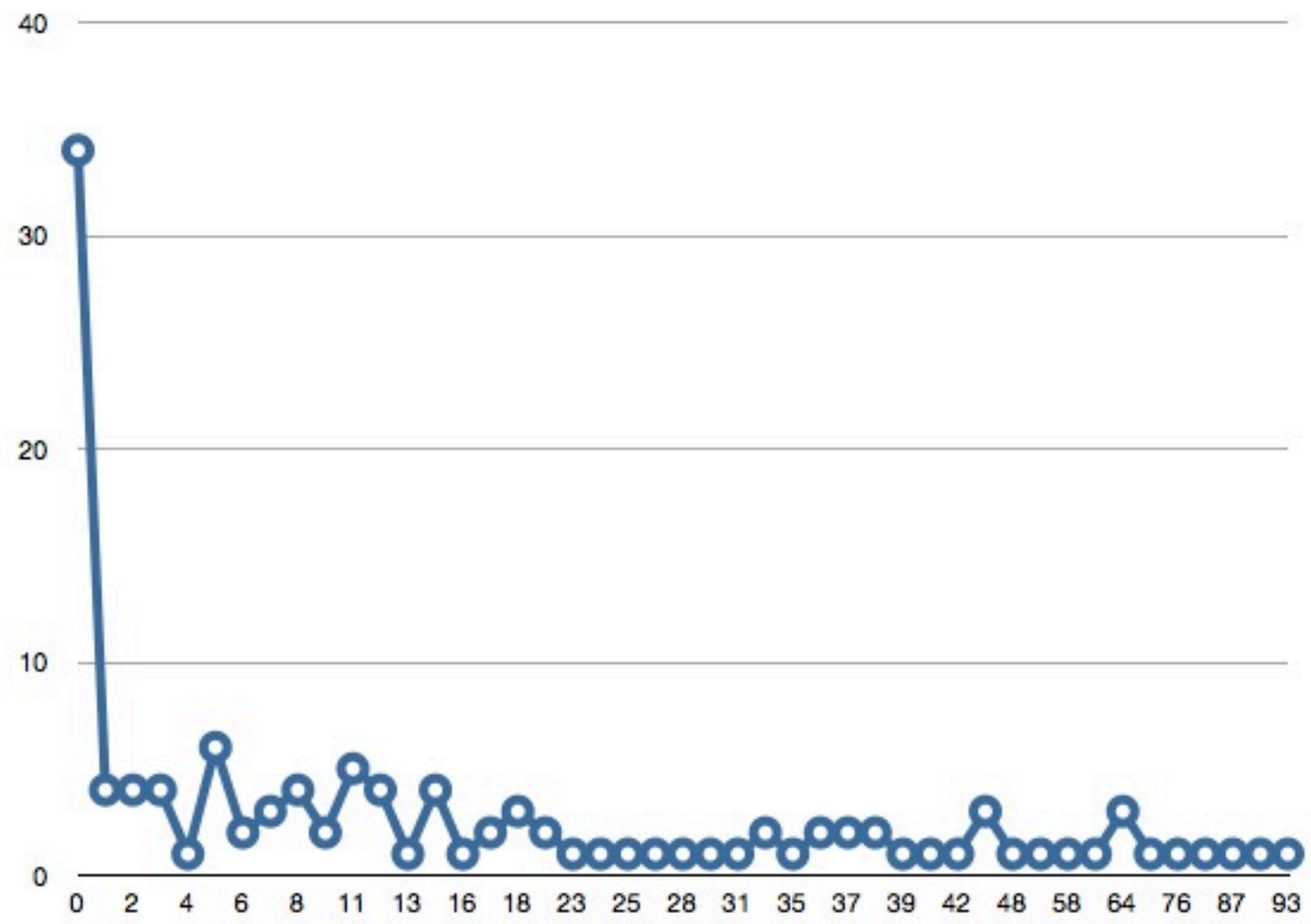
Frequency of Inter-commit Intervals



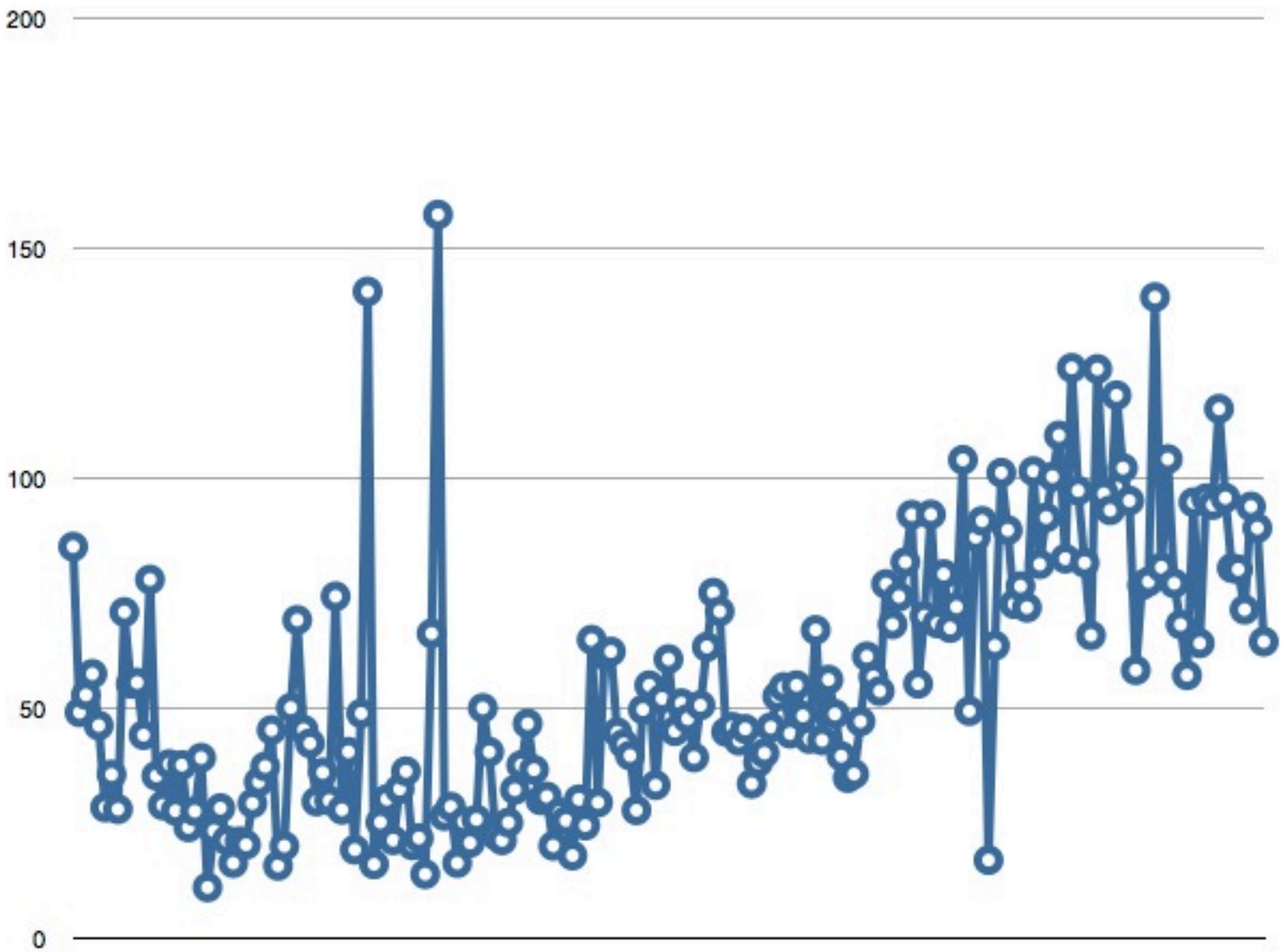
Frequency of Inter-commit Intervals



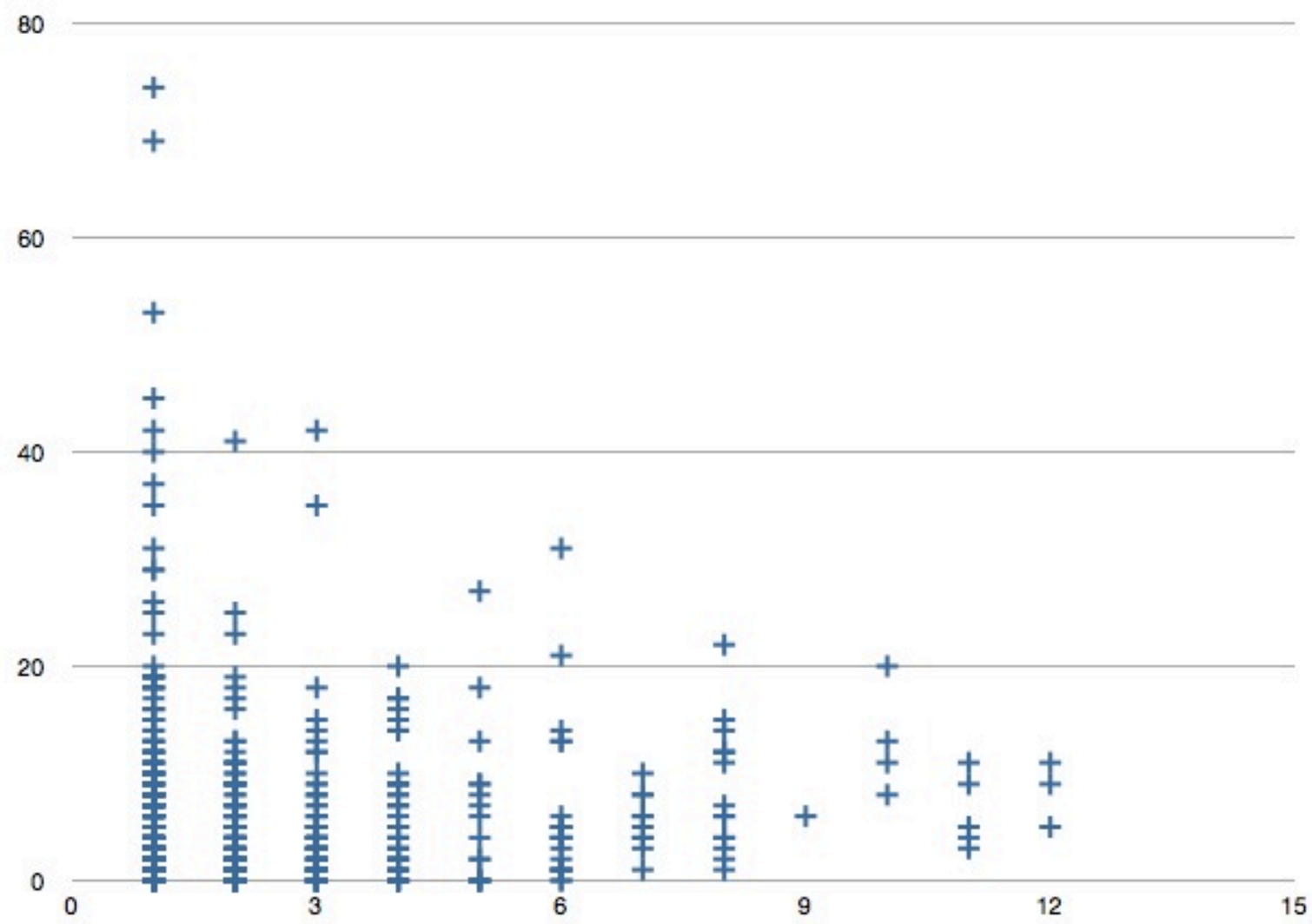
Frequency of Inter-commit Intervals



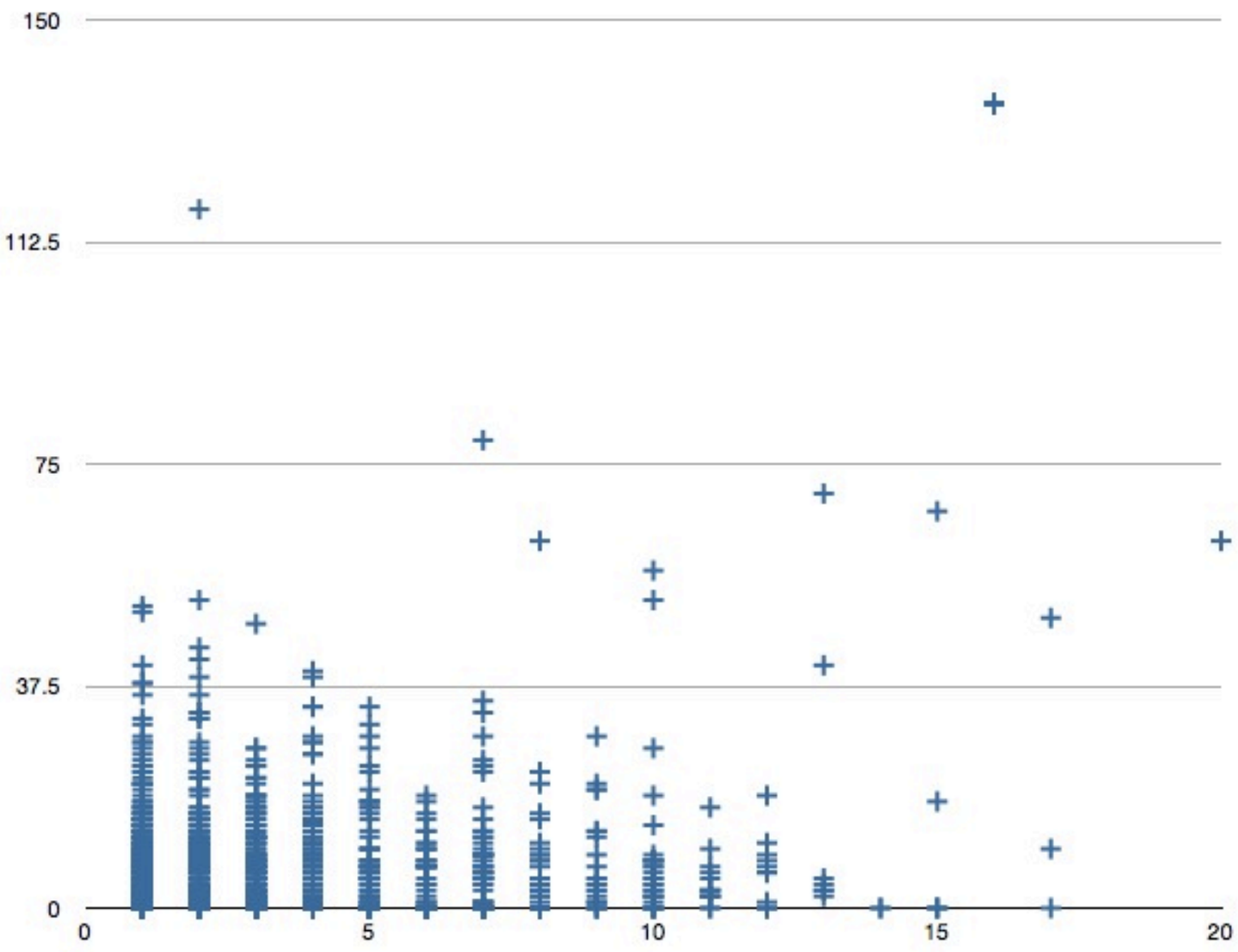
Average Lines of Code Per Commit By Week



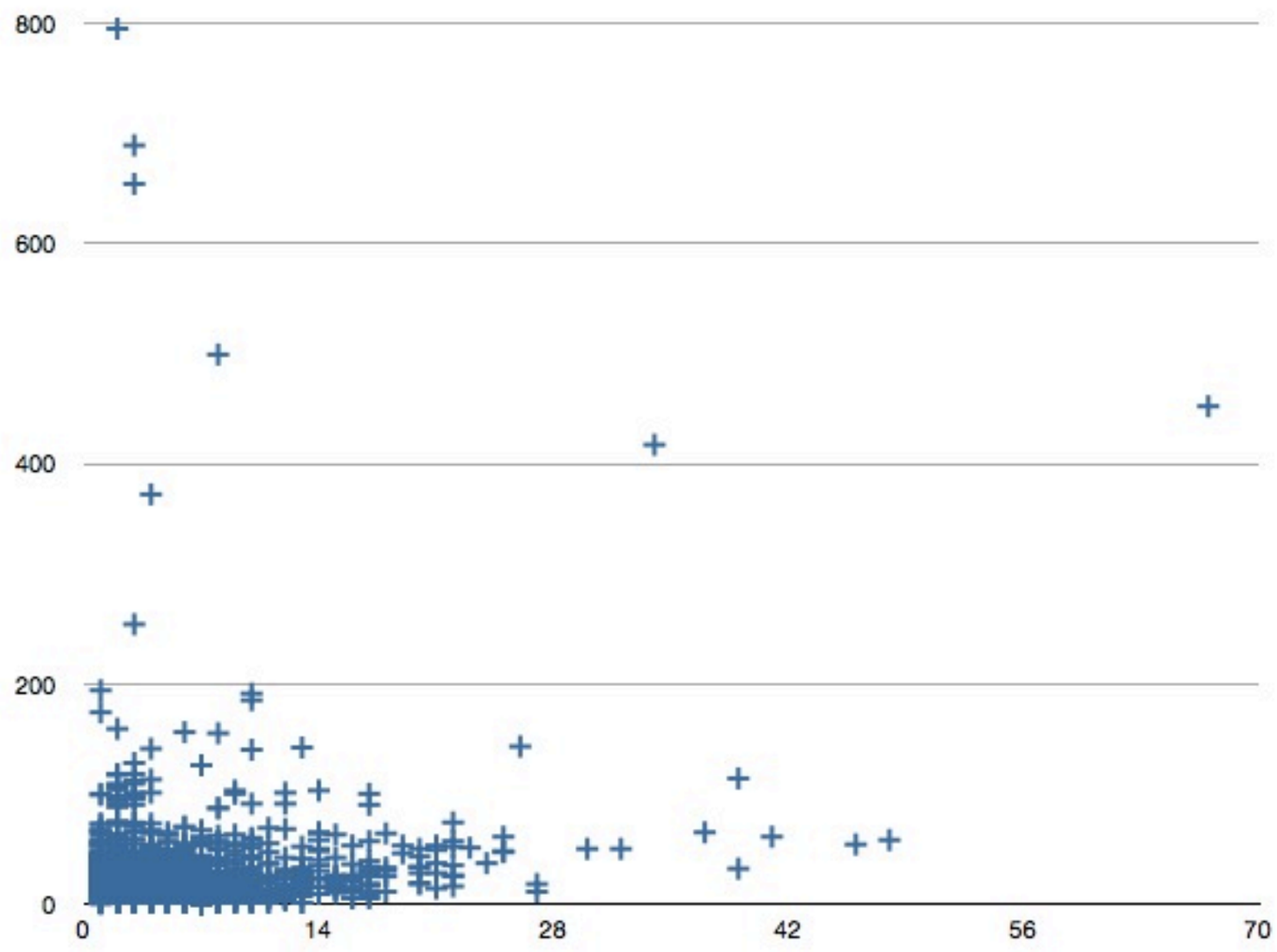
Complexity Tolerance (Developer A)



Complexity Tolerance (Developer B)



Ownership Effect (all methods)



Methods Ascending (5)

```
def methods_ascending_last_n events, n
  methods_by(method_events(events)) do |es|
    es.count >= n && \
      es.map(&:method_length).last(n).each_cons(2).all? {|l,r| l < r }
  end.keys
end
```

Trending Methods

```
def trending_methods events
  method_events(events).select {|e| e.status == :changed } \
                       .group_by {|e| month_from_date(e.date) } \
                       .to_a \
                       .last[1] \
                       .freq_by(&:method_name)
                       .sort_by {|_,count| -count } \
                       .take(10)
end
```

Classes By Closure Date

```ruby
def classes_by_closure events
  class_names = method_events(events).map(&:class_name).uniq
  classes = Hash[class_names.zip([Time.now] * class_names.length)]
  method_events(events).each {|e| classes[e.class_name] = e.date }
  classes.to_a.sort_by {|_,date| date }
end
```

Temporal Correlation of Class Changes

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

Behavioral Economics

If you want parole, have your case heard right after lunch

By Kate Shaw | Last updated 4 days ago

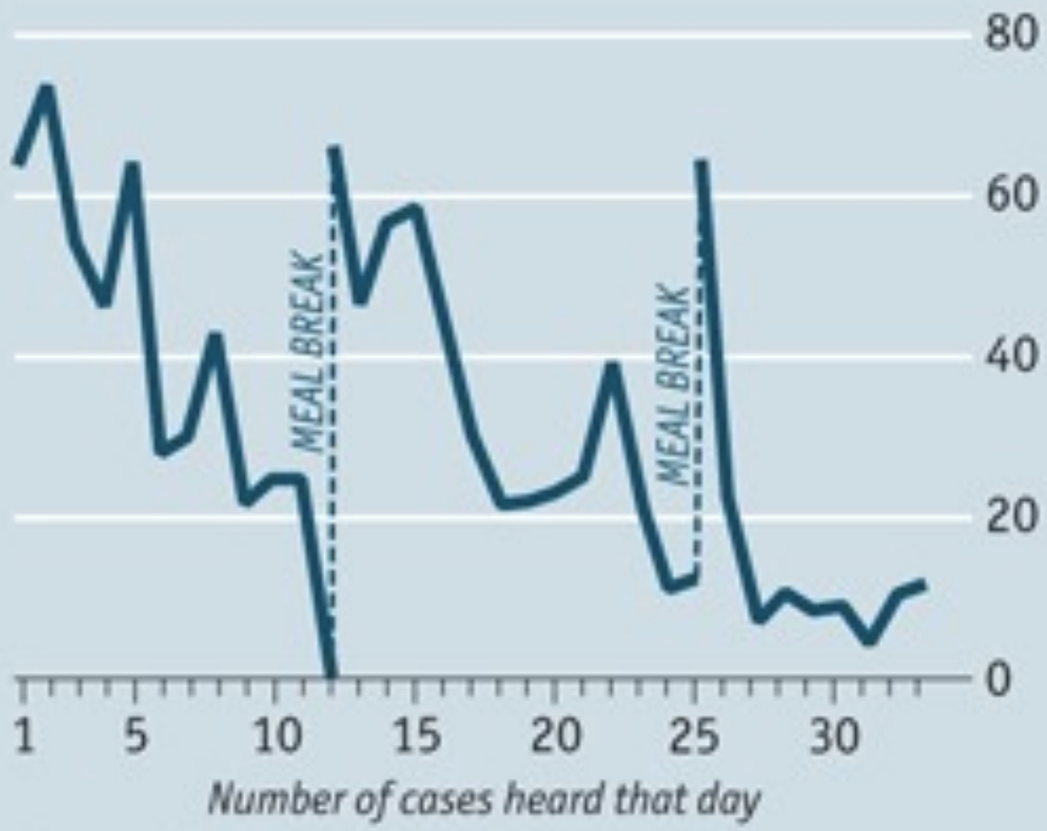
Between the courtroom antics of lawyers, witnesses, and jurors, reason doesn't always prevail in our legal system. But judges are trained to be impartial, consistent, and rational, and make deliberate decisions based on the case in front of them, right? Actually no, according to a new study in *PNAS*, which shows that judges are subject to the same whims and lapses in judgment as the rest of us.

The authors examined over 1,000 parole decisions made by eight judges in Israel over a 10-month period. In each parole request, a prisoner appeared in front of a judge, and the judge could either accept or deny the request. The judges heard between 14 and 35 of these cases per day, separated into three distinct sessions. The first session ran from the beginning of the day until a mid-morning snack break, the second lasted from the snack break until a late lunch, and the third lasted from lunch until the end of the day.

Overall, judges were much more likely to accept prisoners' requests for parole at the beginning of the day than the at end. Moreover, a prisoner's chances of receiving parole more than doubled if his case was heard at the beginning of one of the three sessions, rather than later on in the session. More specifically, it was the number of rulings that a judge made, rather than the time elapsed in a session, that significantly affected later decisions. Every single judge in the sample followed this pattern.

Judgment day

Favourable rulings by parole boards, %



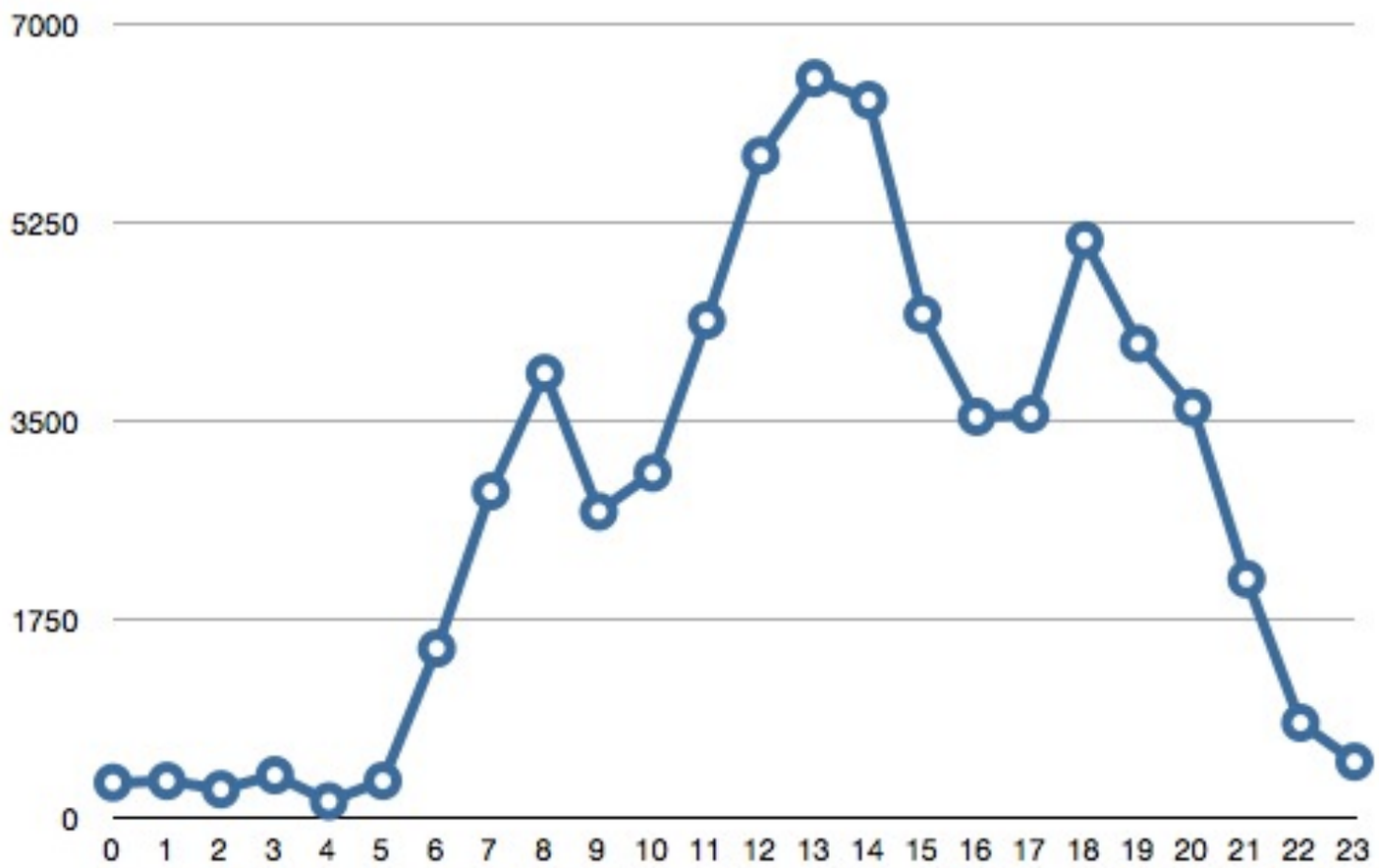
Source: *PNAS*

Is it easier to add code to an existing method or to add a new method?

Is it easier to add a method to an existing class or to add it a new class?

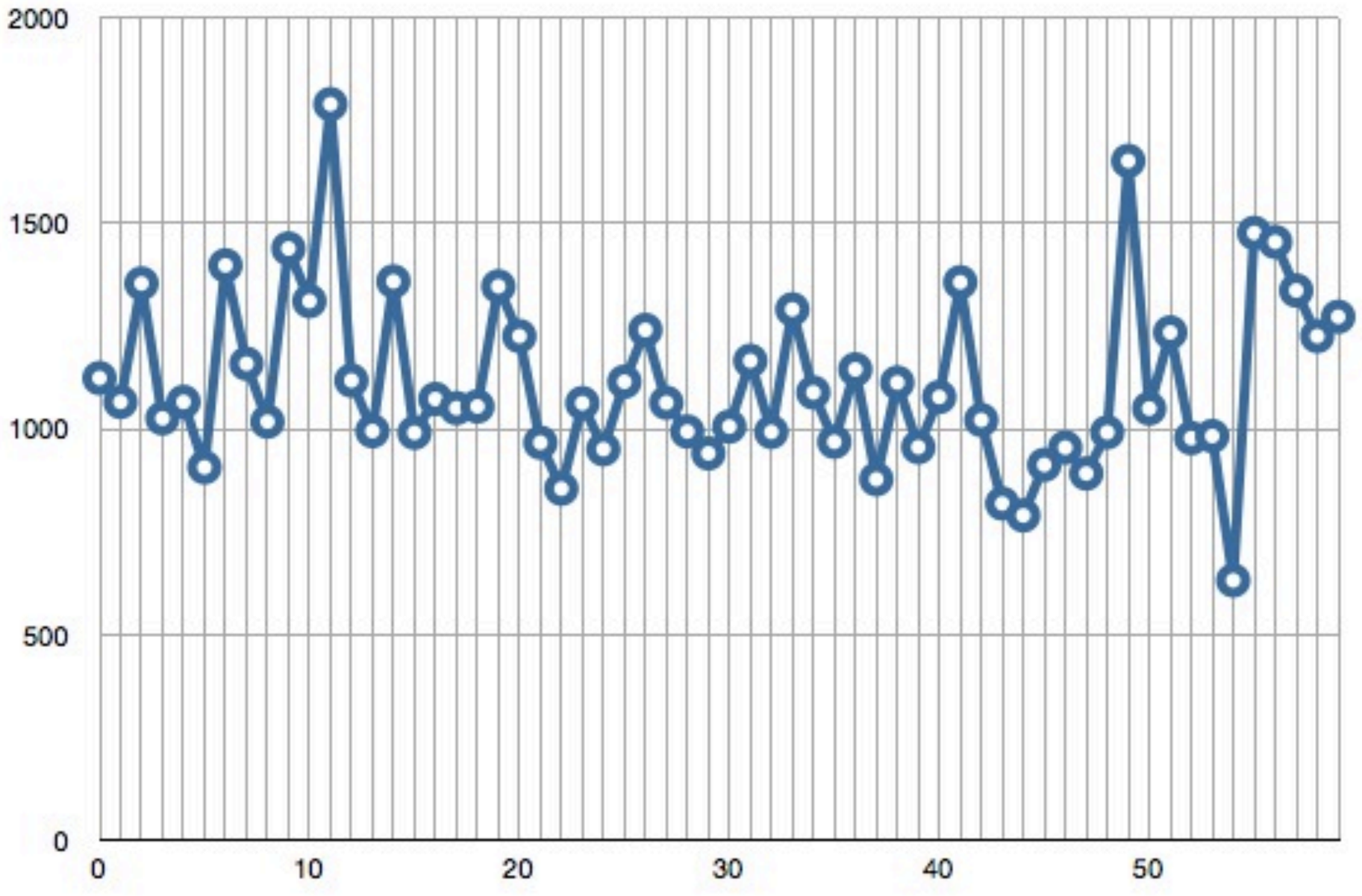
We should not be surprised by what we see.

Commits By Hour



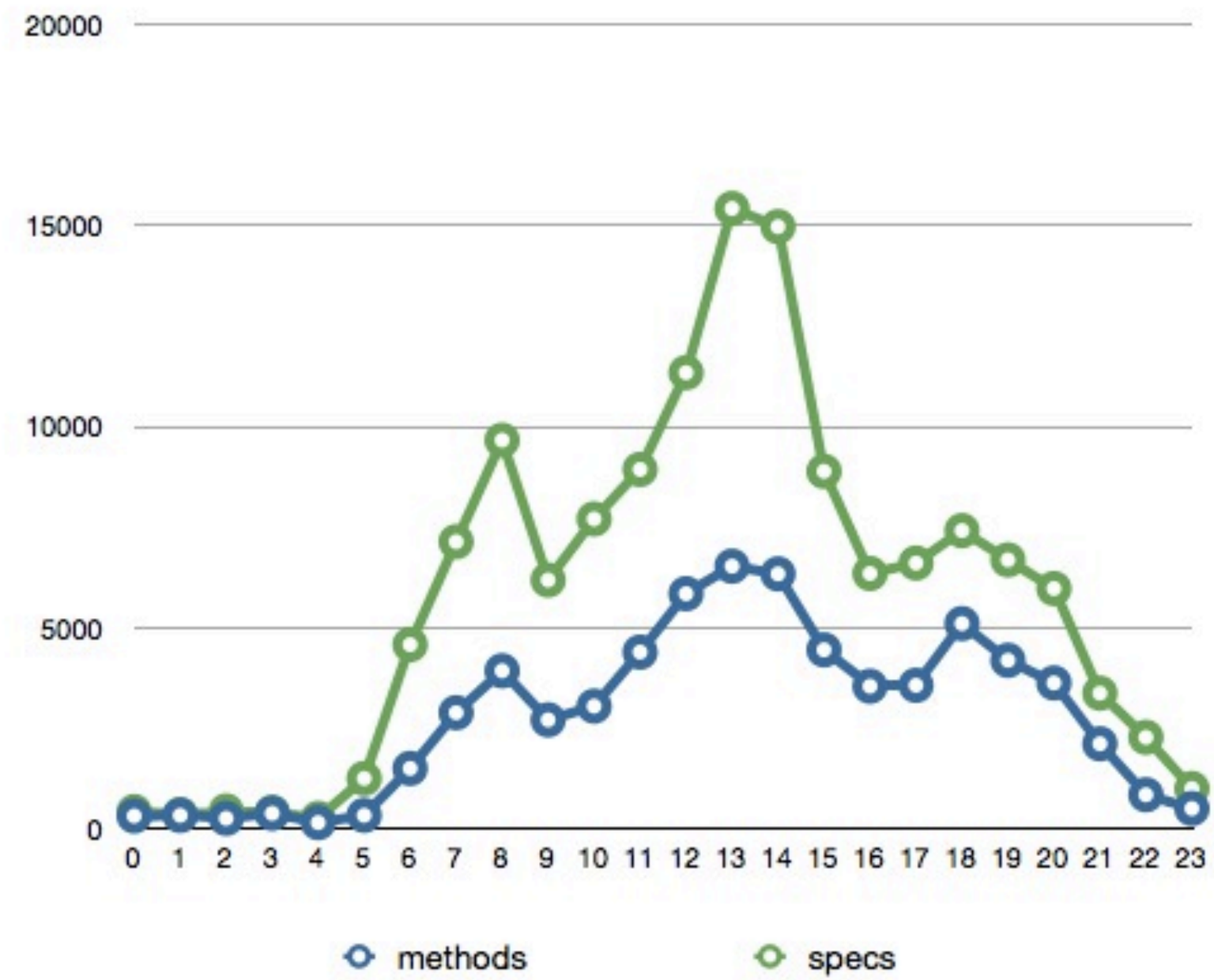
chart(['hour','commits'],method_events(events).freq_by {|e| e.date.hour })

Commits By Minute

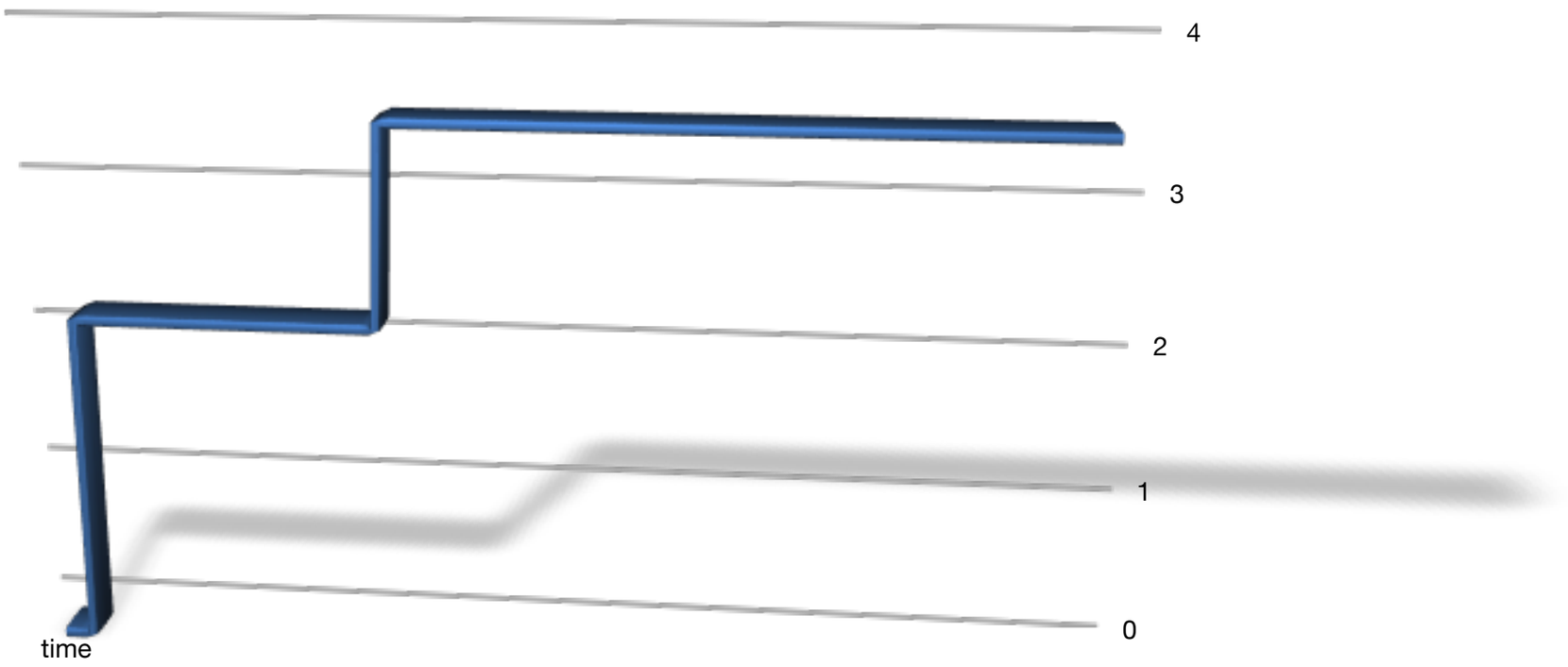


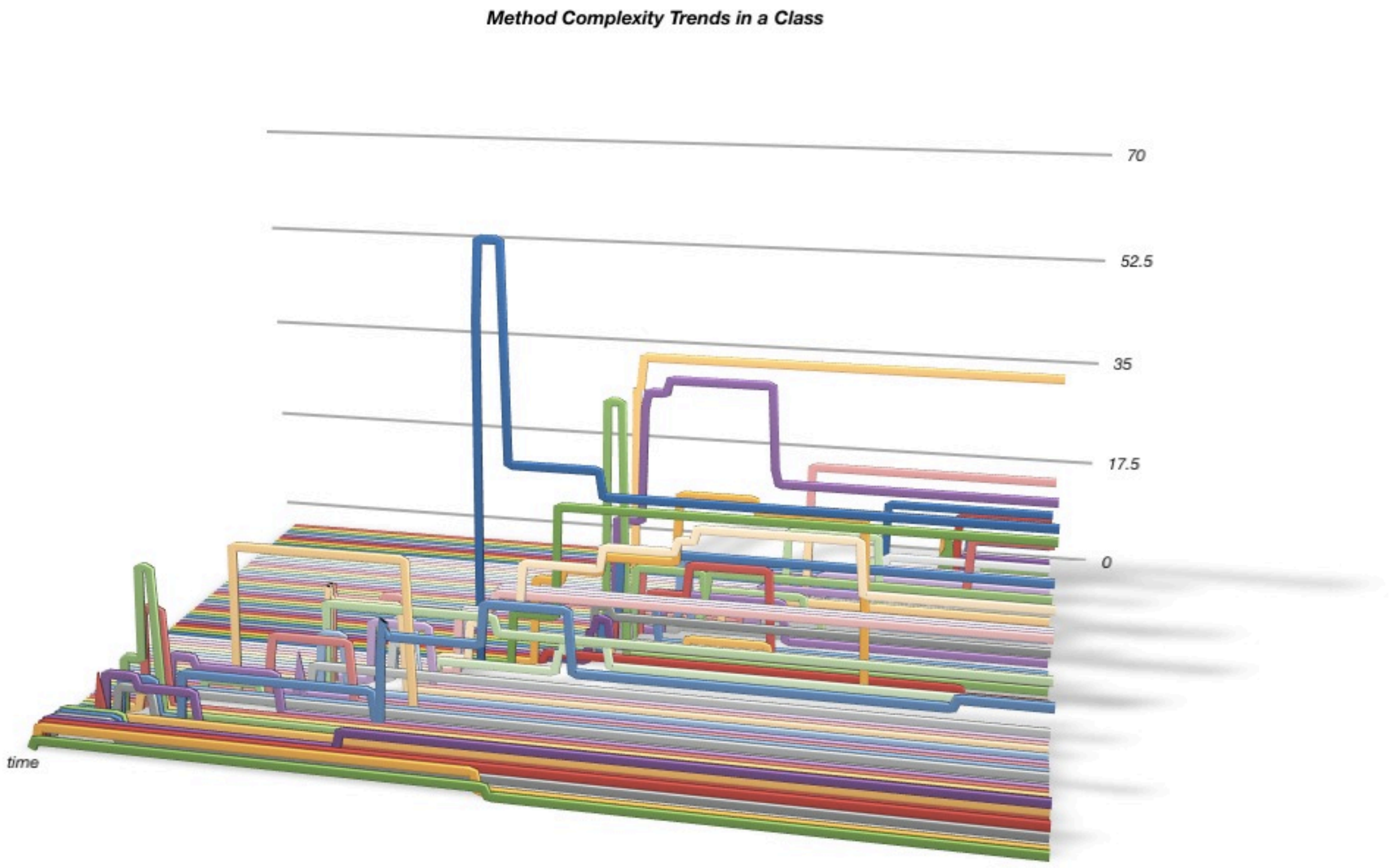

```
chart(['hour','commits'],method_events(events).freq_by {|e| e.date.min })
```

Method and Spec Changes Per Hour of Day



Another Method

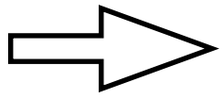



Method Complexity Trends in a Class
70
52.5
35
17.5
0
time

Code Blindness

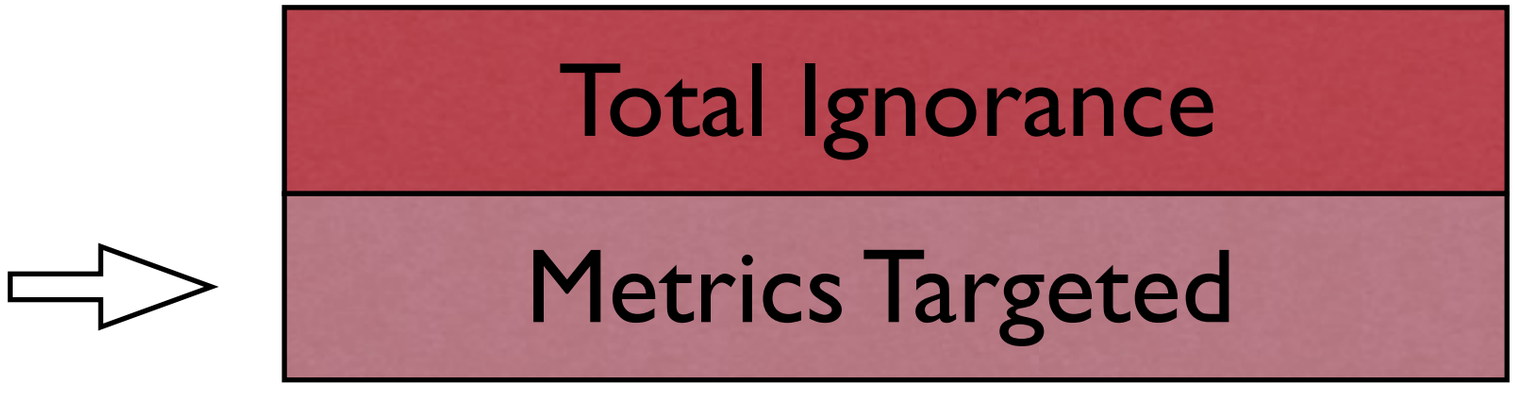
Walking Out of Code Blindness

Walking Out of Code Blindness



Total Ignorance

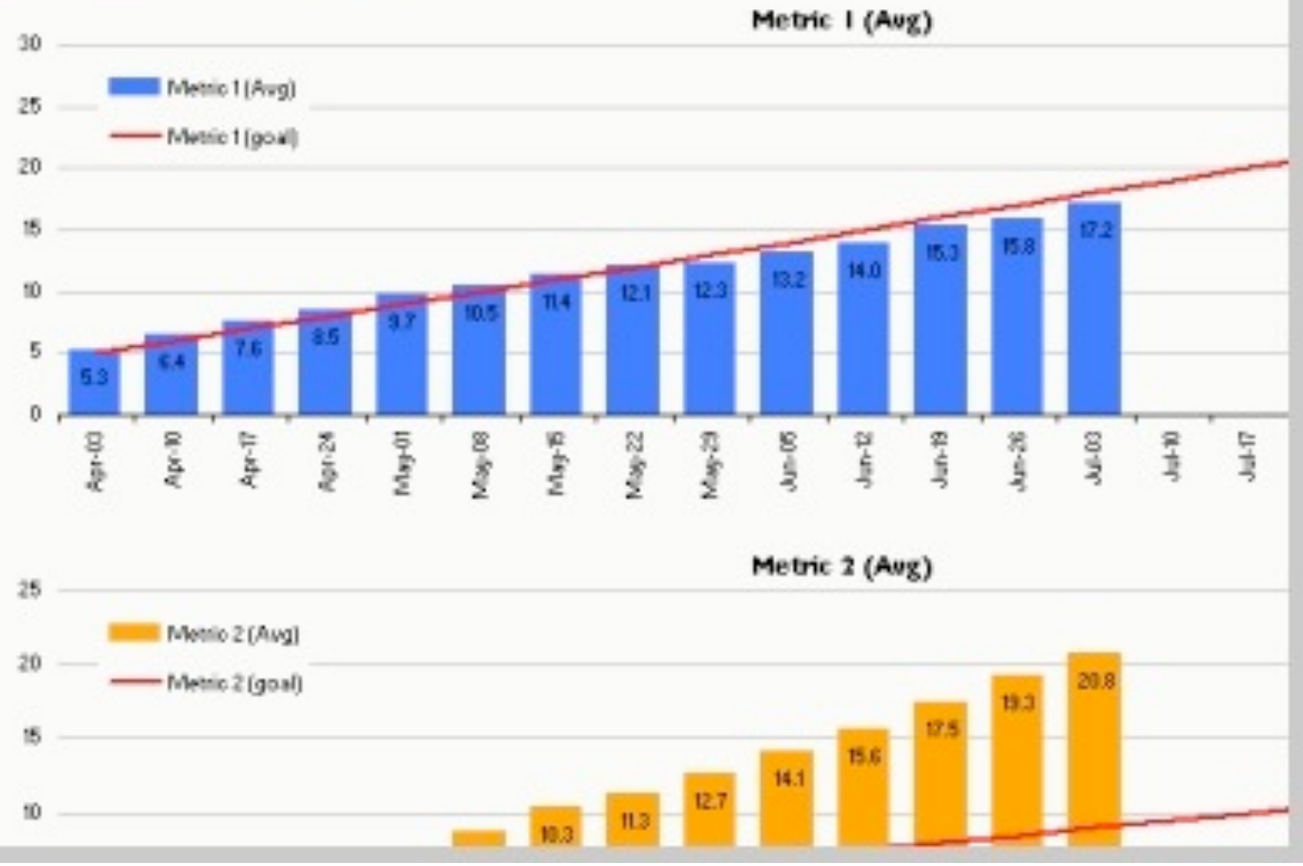
Walking Out of Code Blindness



Organization Success Metrics

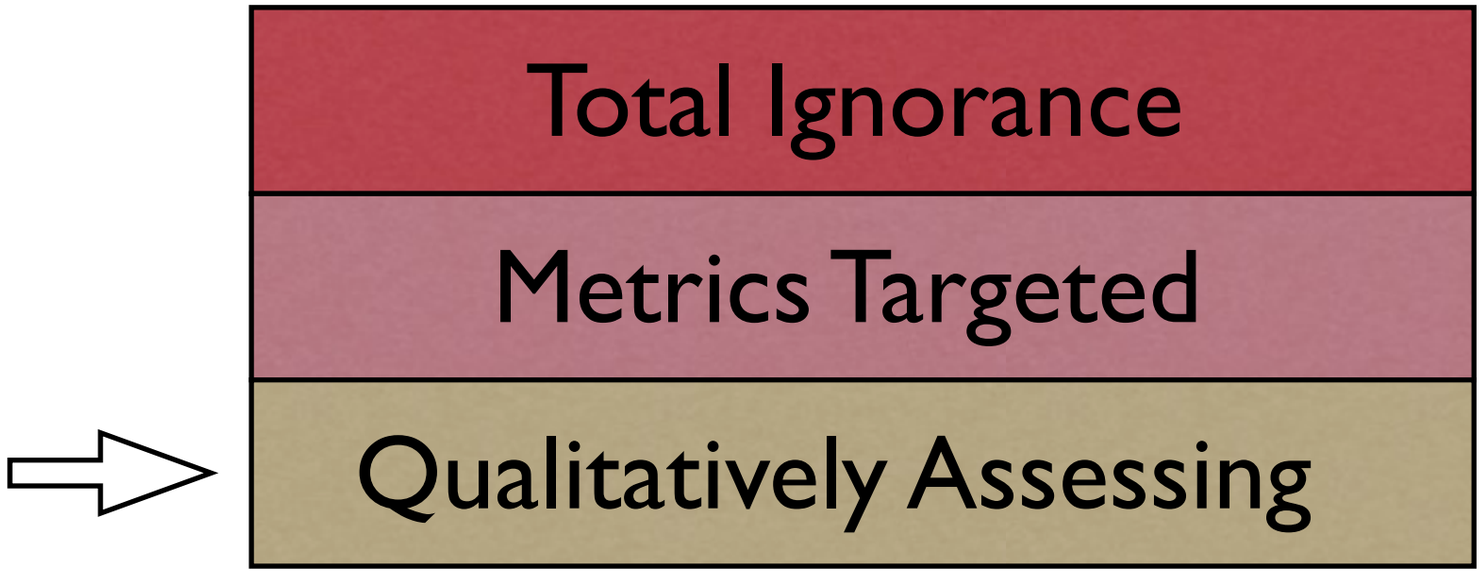
Dimension 1	(All)
Dimension 2	(All)

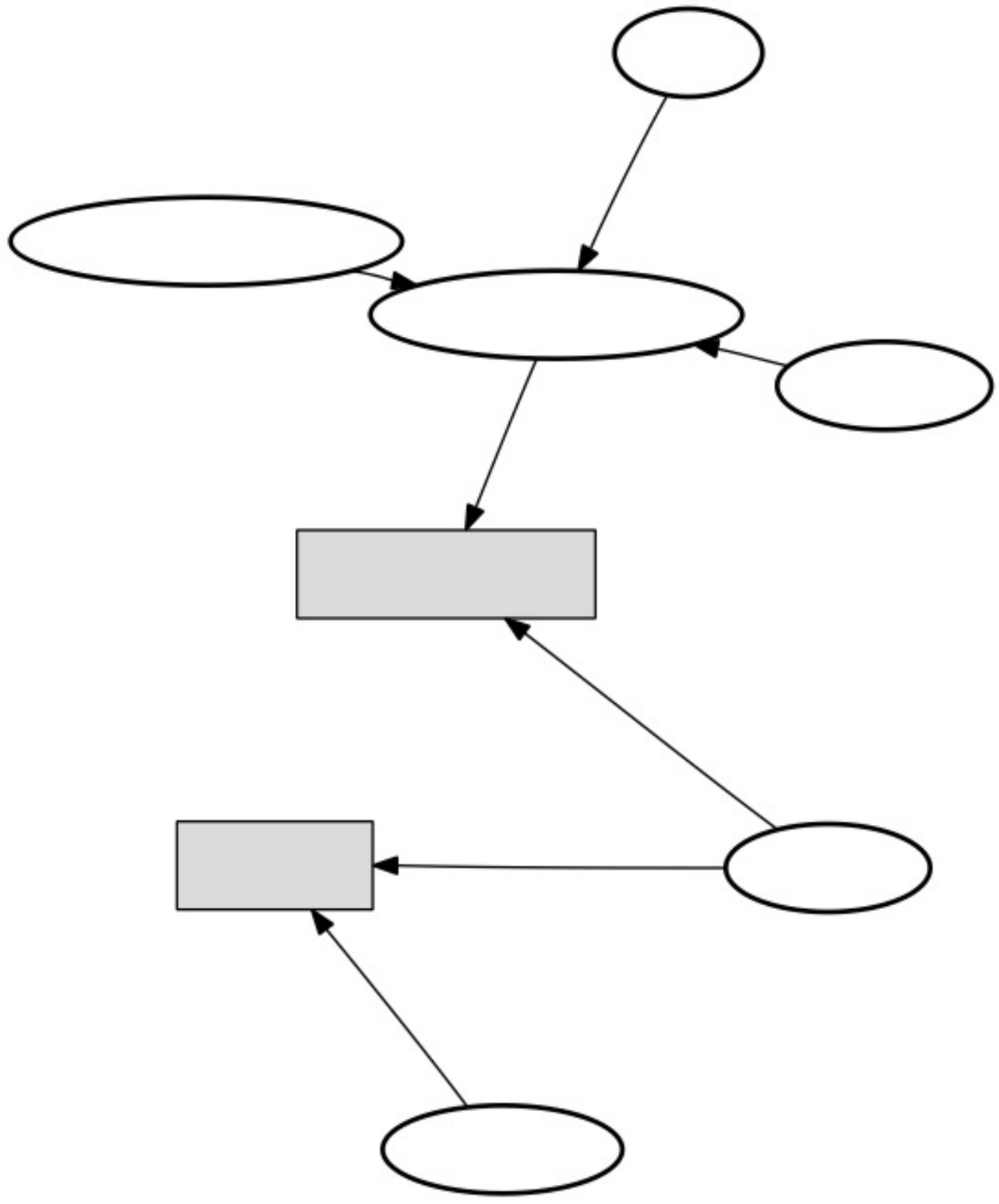
<-- use these dimensions to drill into the metrics

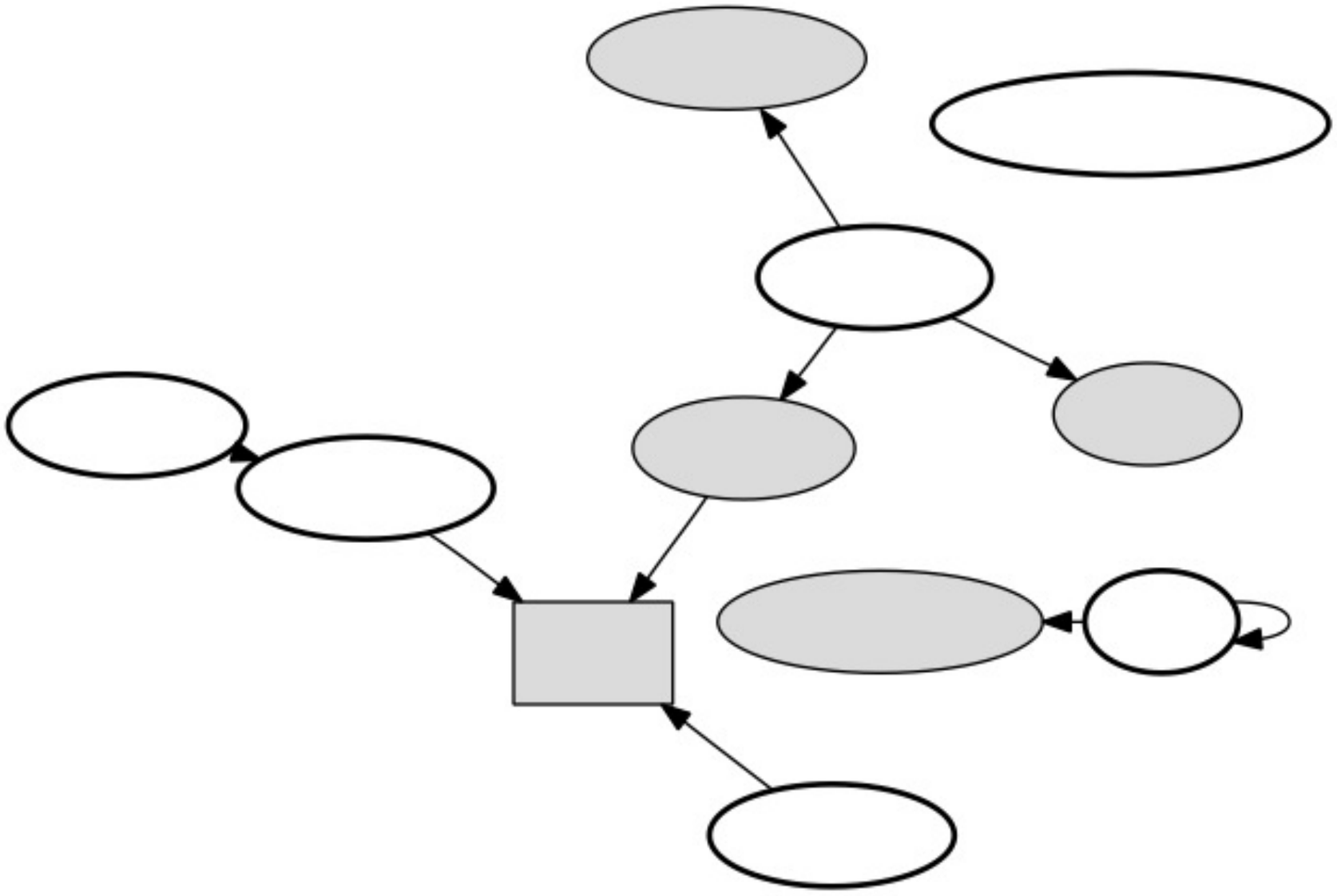


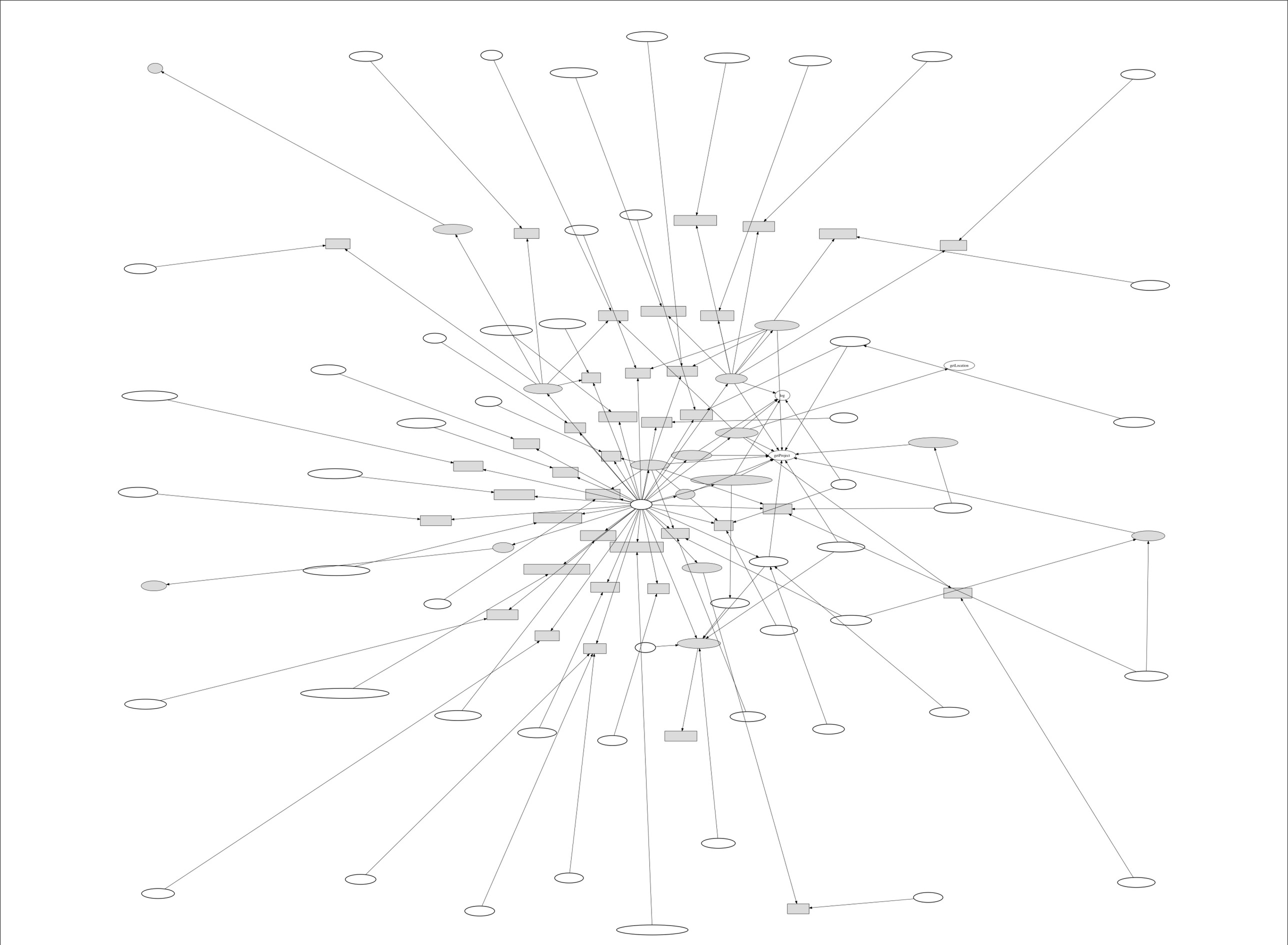
There is nothing more abstract than 4 (except maybe 5).

Walking Out of Code Blindness

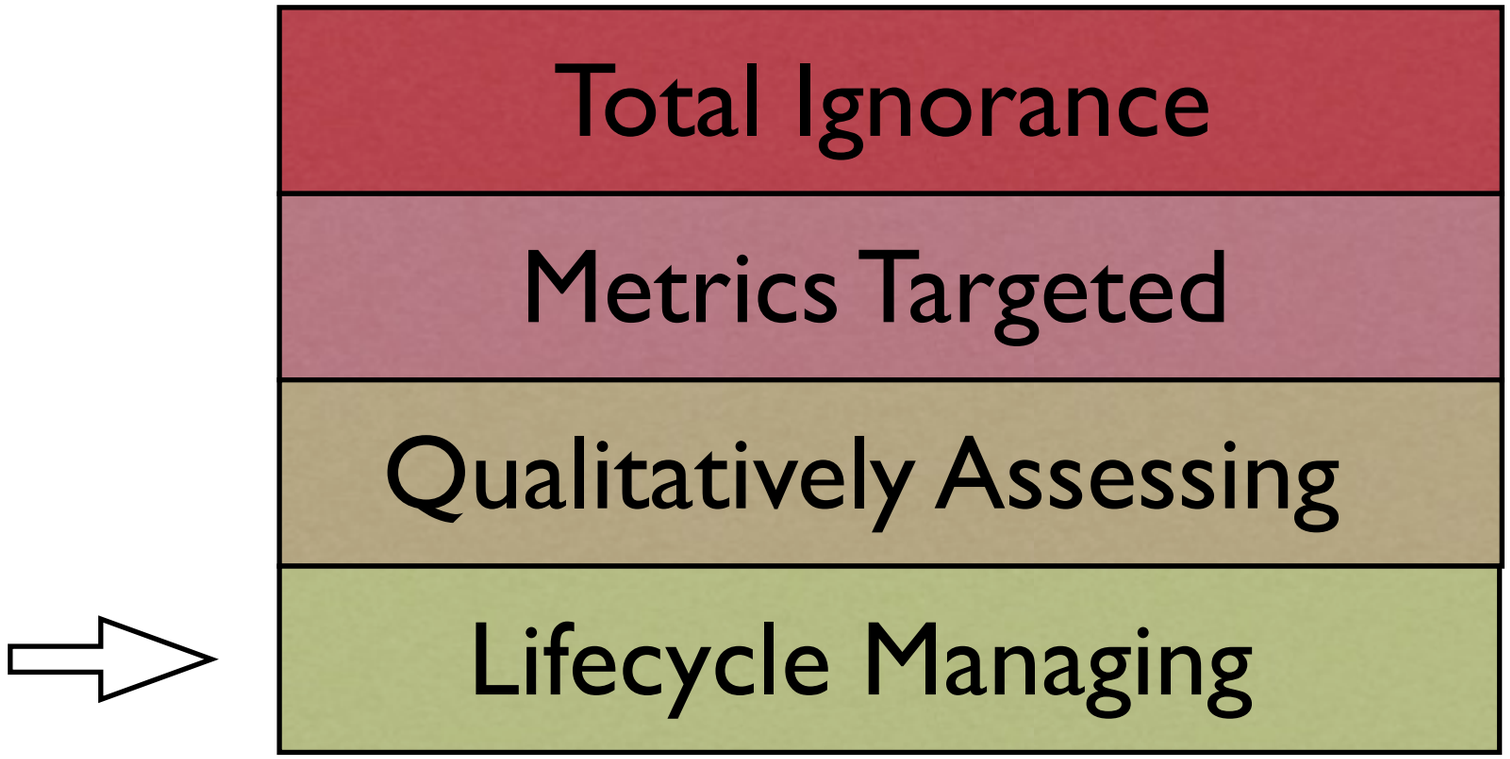


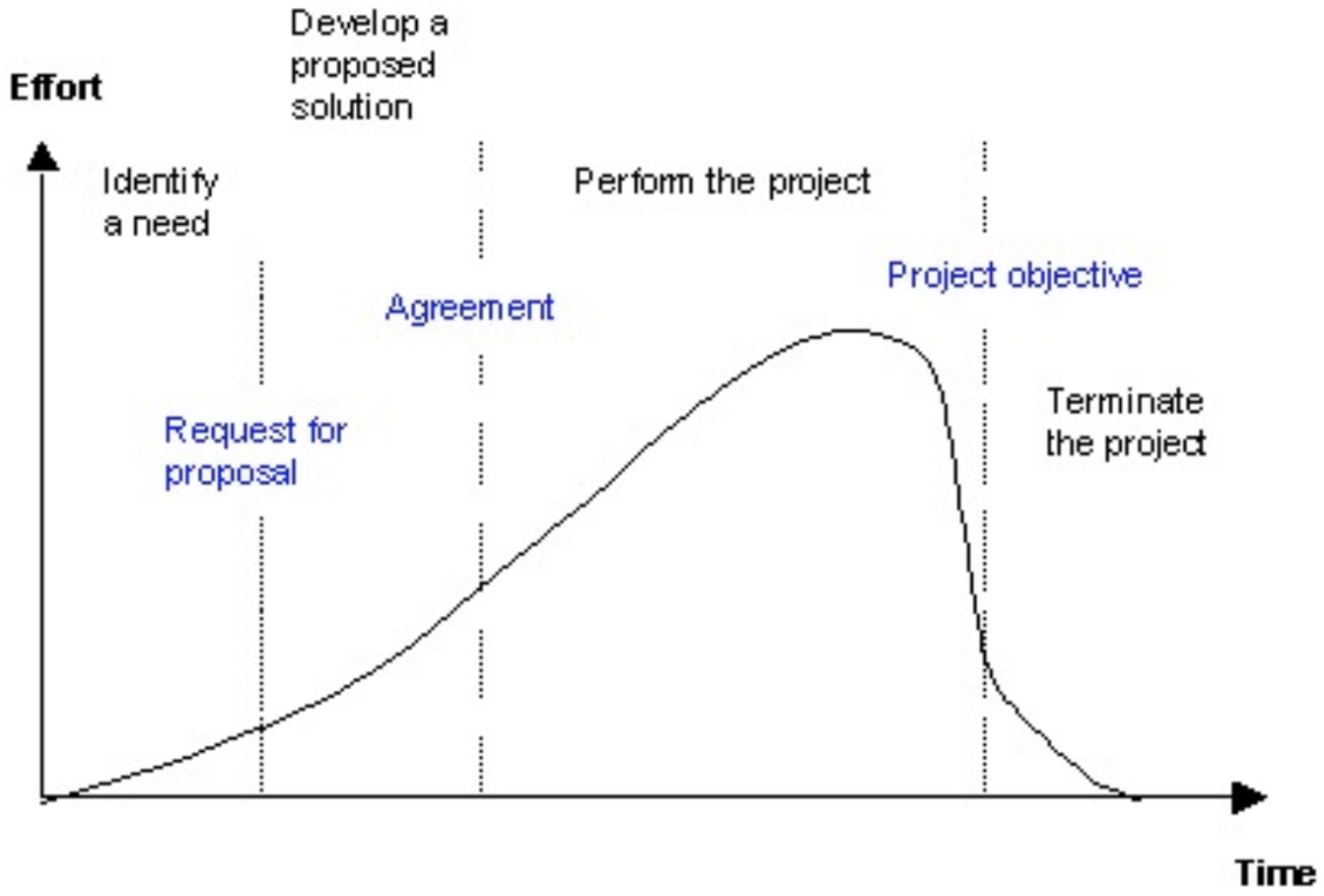




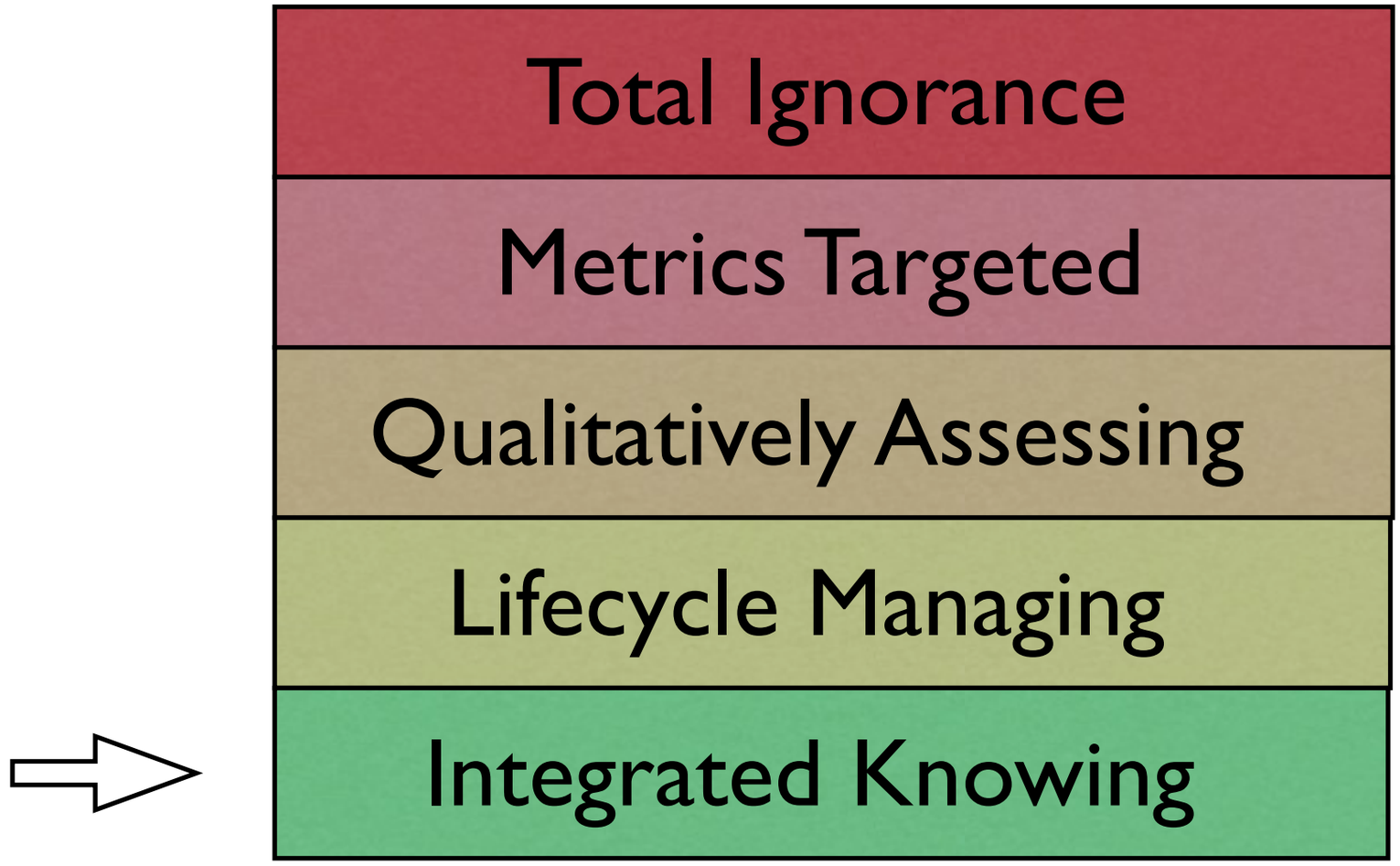

getLocation
log
getProject

Walking Out of Code Blindness



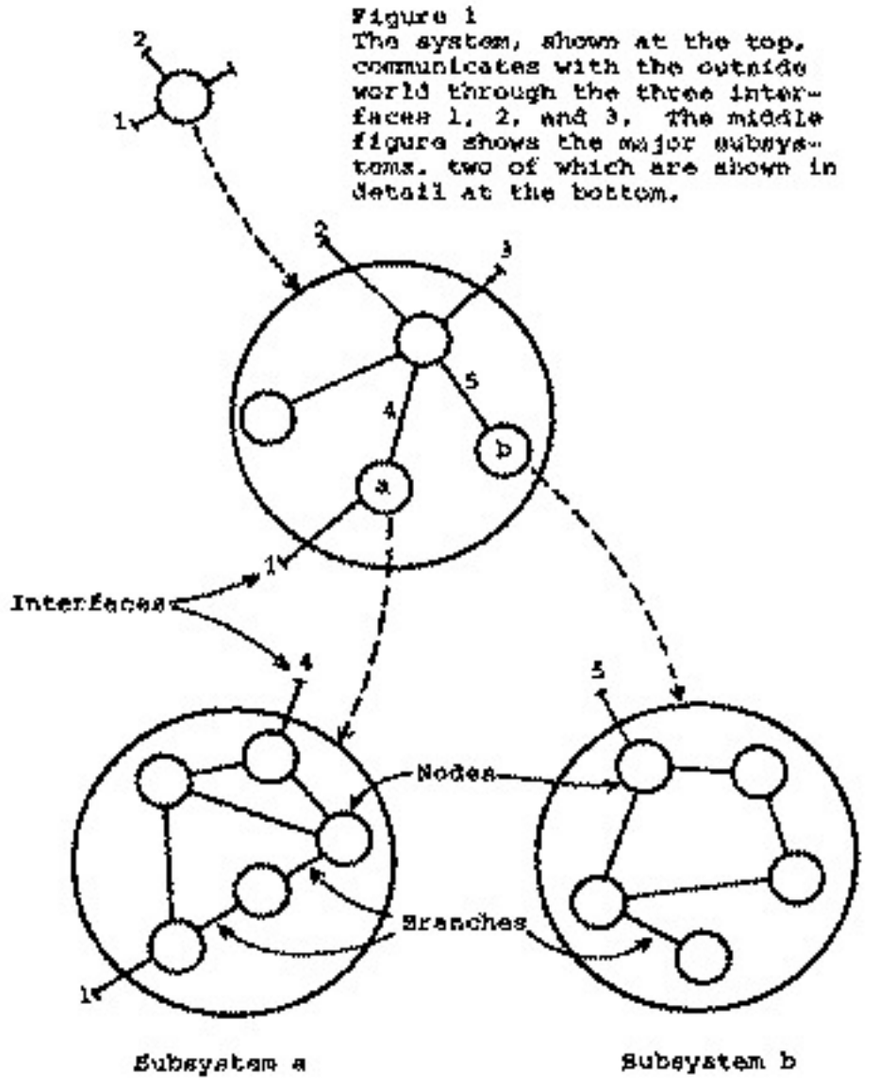

Effort
Develop a proposed solution
Identify a need
Perform the project
Project objective
Agreement
Request for proposal
Terminate the project
Time

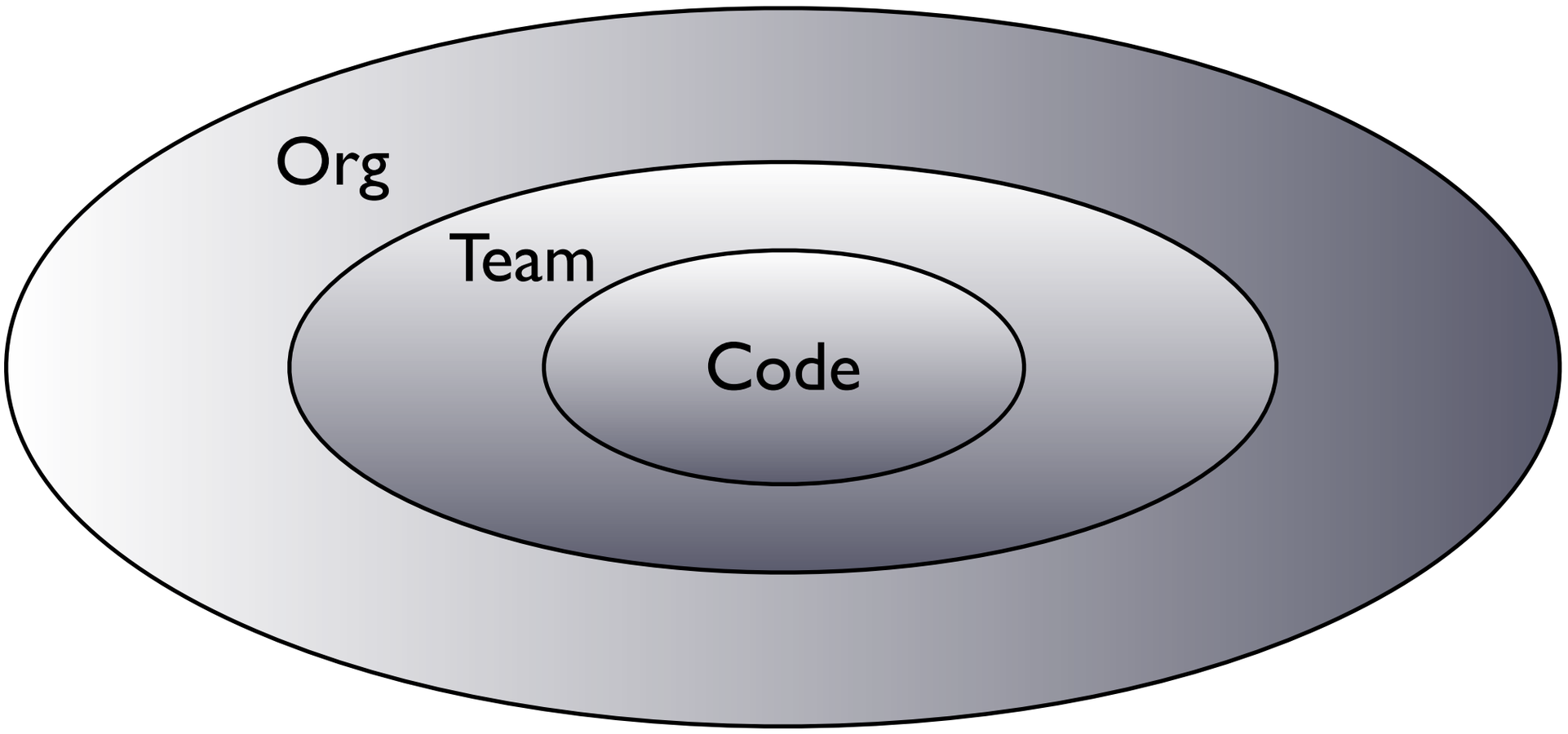
Walking Out of Code Blindness

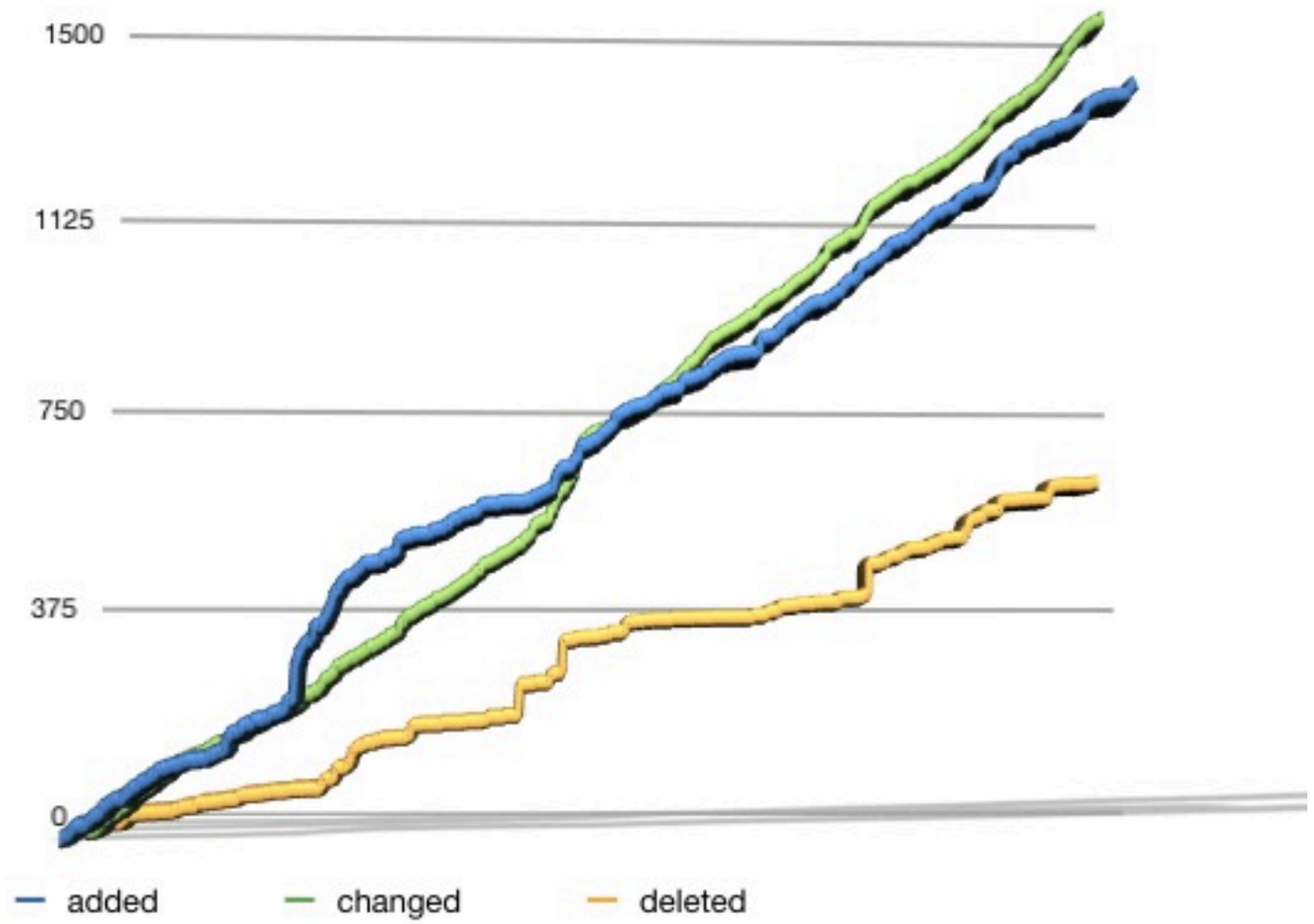


Conway's Law

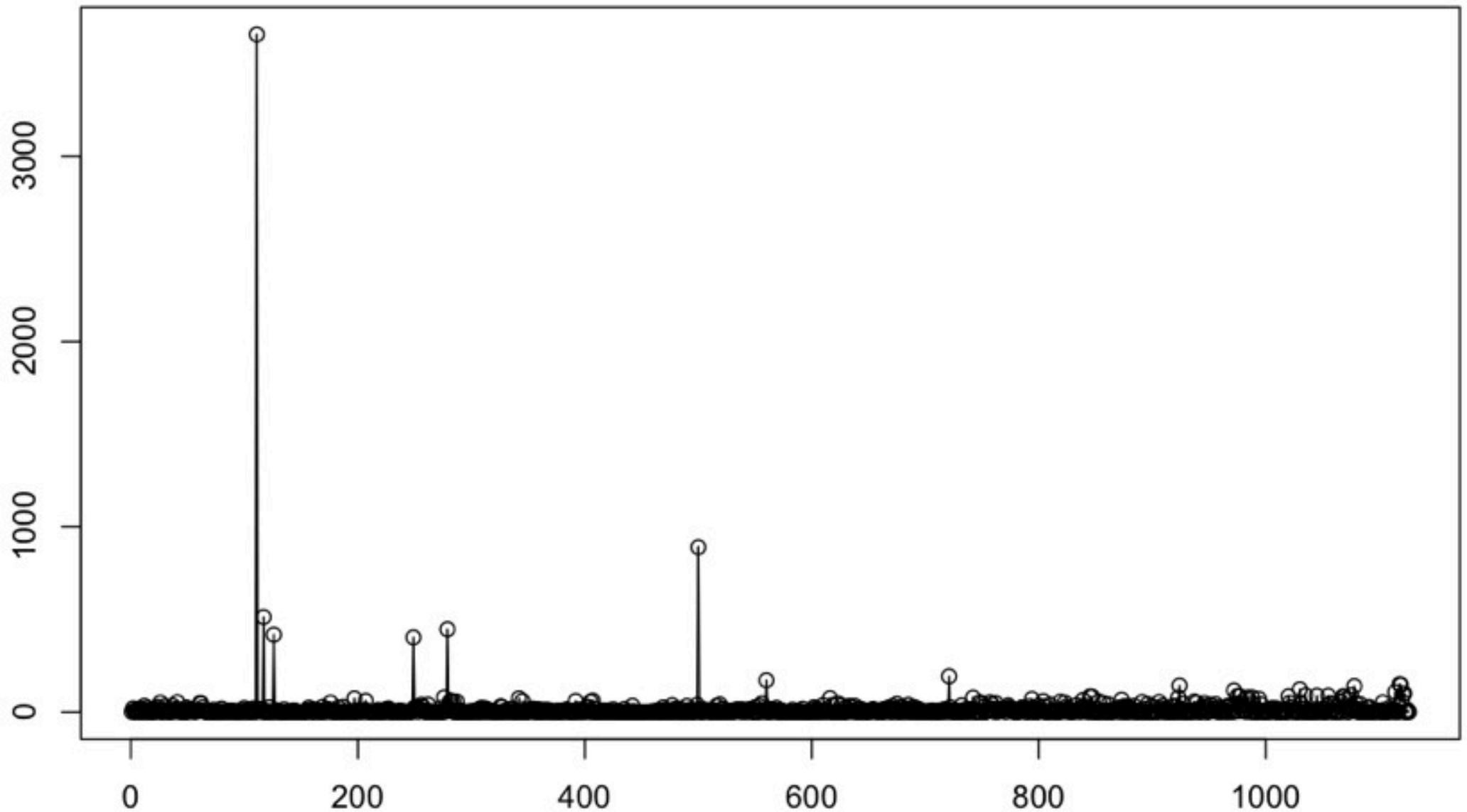
Any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure.



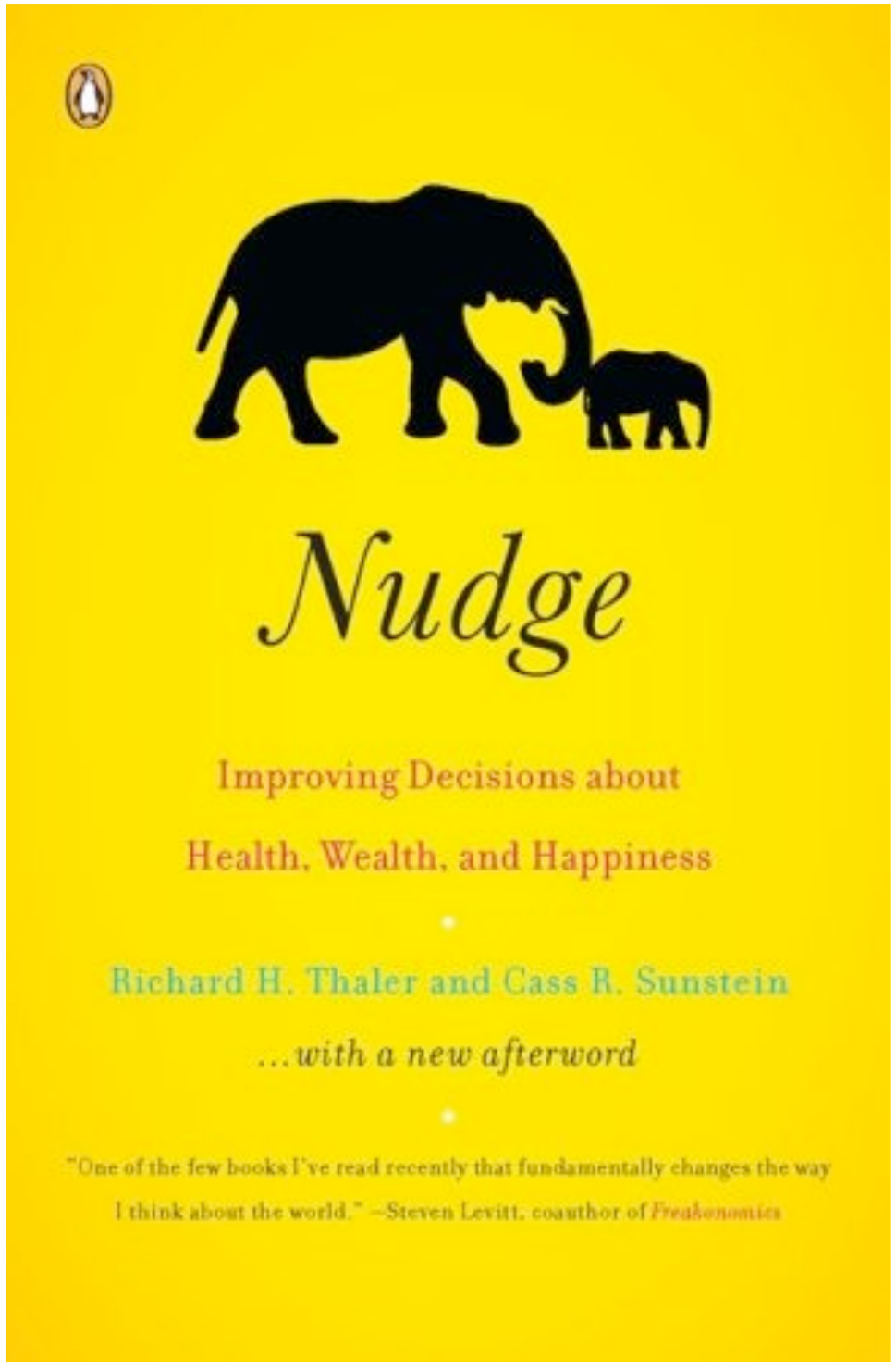

Org
Team
Code


1500
1125
750
375
0
added
changed
deleted

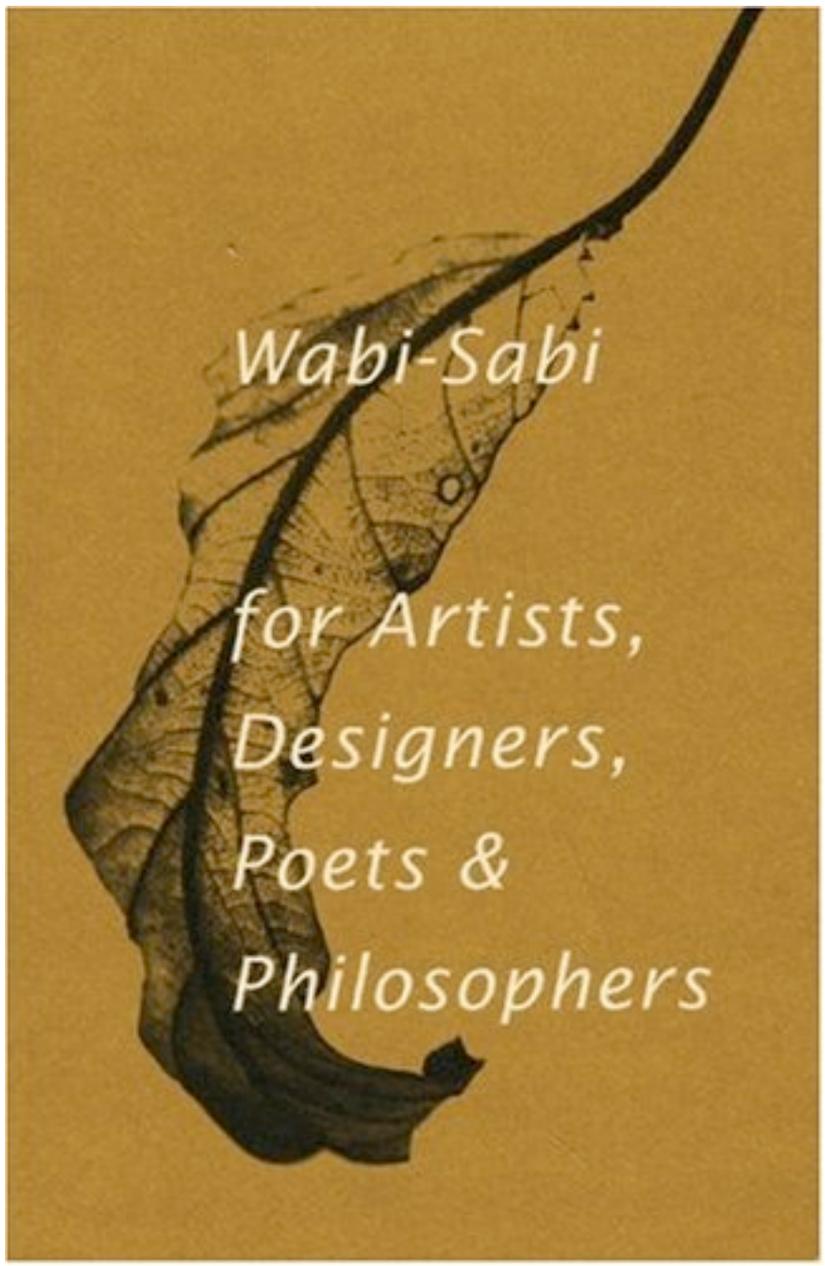
Added Complexity Over Time



repo.commits.map {|c,_| repo.commit(c).added_complexity.to_i }


Nudge
Improving Decisions about
Health, Wealth, and Happiness
Richard H. Thaler and Cass R. Sunstein
...with a new afterword
"One of the few books I've read recently that fundamentally changes the way I think about the world." —Steven Levitt, coauthor of Freakonomics

Peter Provost - The Bufferfly Effect


Wabi-Sabi
for Artists,
Designers,
Poets &
Philosophers