“Through the Looking Glass”

Applying Analytics to Development

Michael Feathers
Independent Consultant

How Much Do We Know?

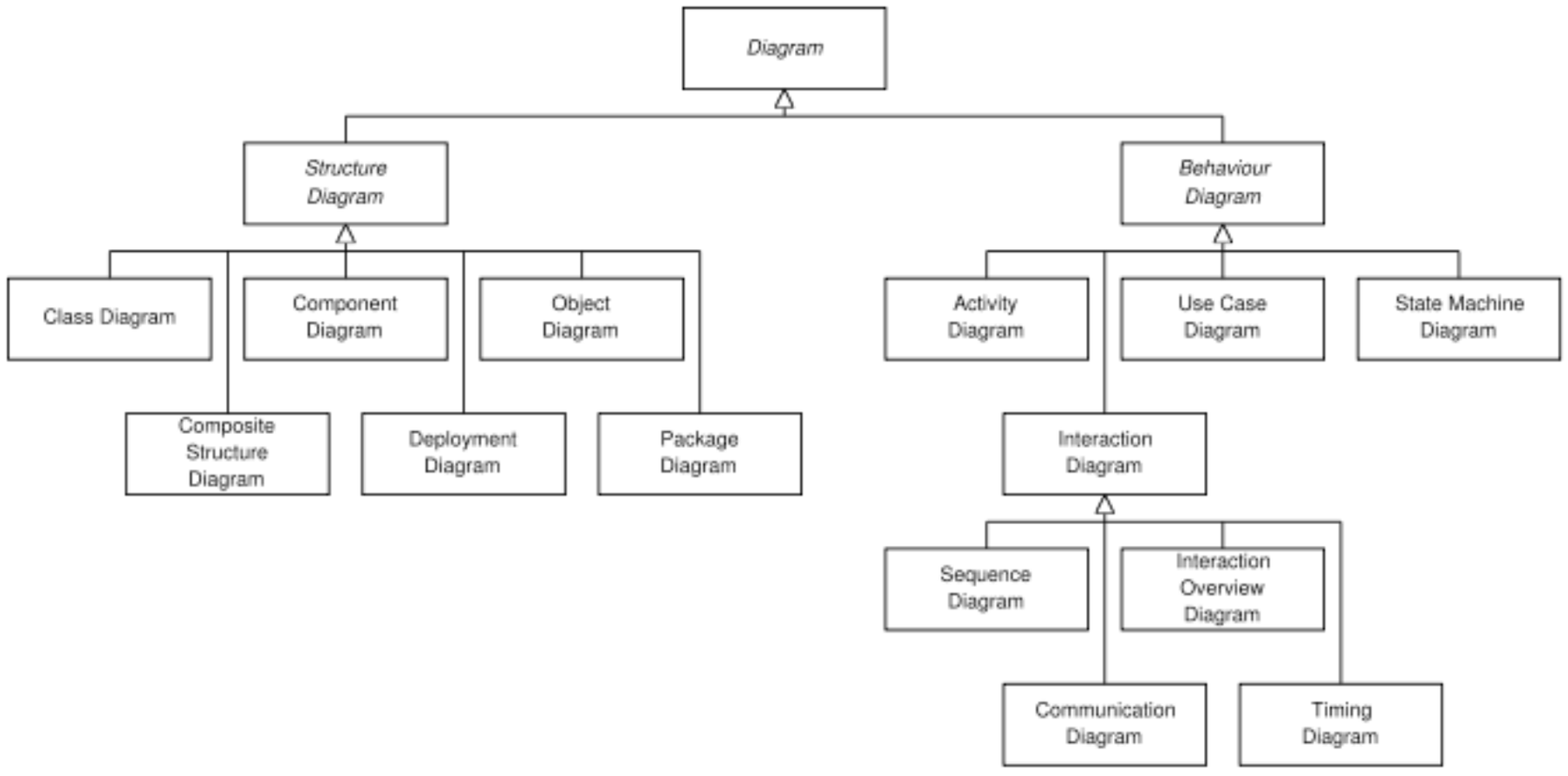
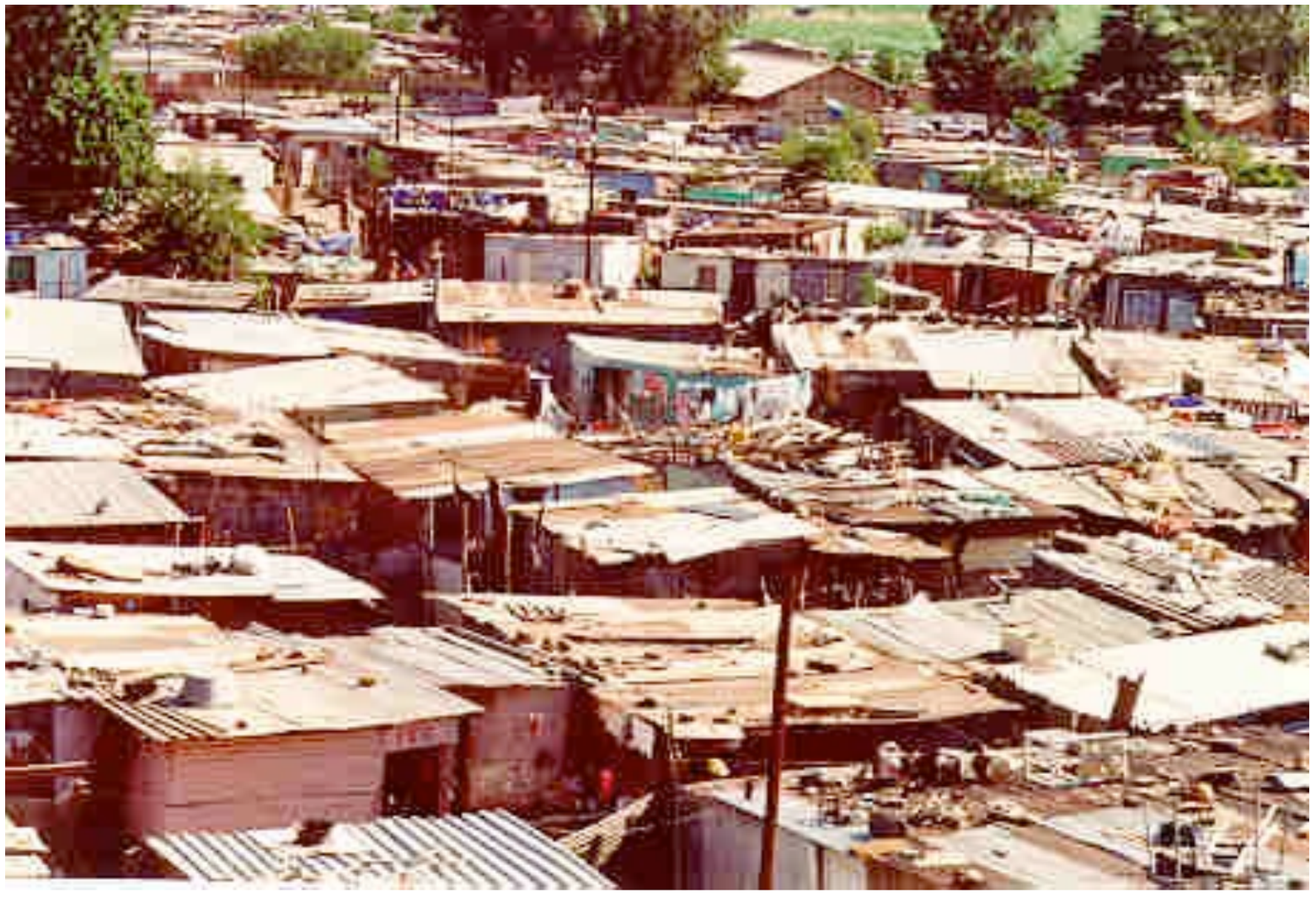


Diagram
Structure Diagram
Behaviour Diagram
Class Diagram
Component Diagram
Object Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Activity Diagram
Use Case Diagram
State Machine Diagram
Interaction Diagram
Sequence Diagram
Interaction Overview Diagram
Communication Diagram
Timing Diagram



Rationalistic Bias

Version Control History

Source Control
Change Events as Data

Git

Git

Commit

Git

Commit

commit hash (sha1)
time/date stamp
committer
files
actual change

Git

Method Event

commit hash (sha1)
time/date stamp
committer
method name
method body
add/change/delete

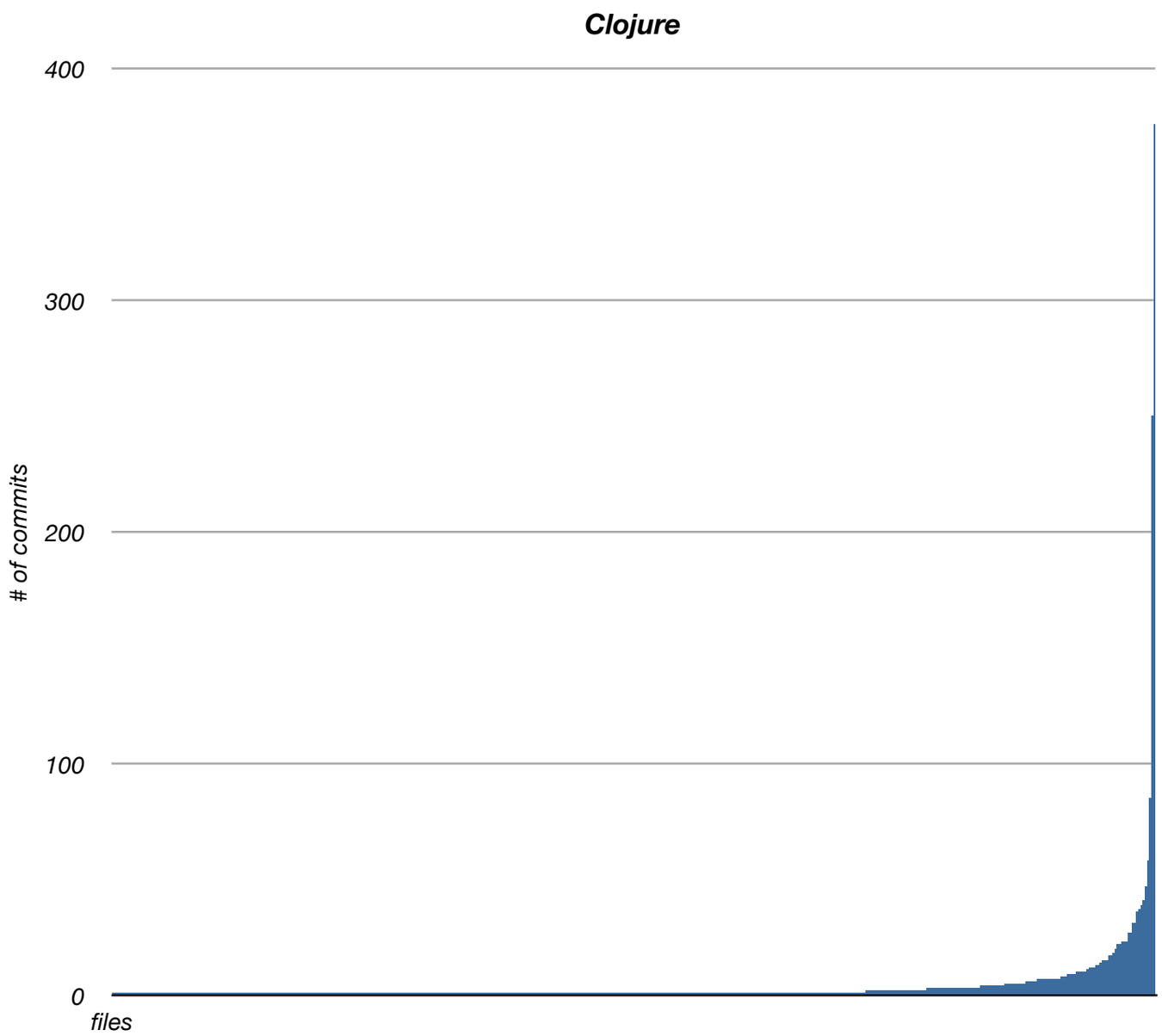
Commit

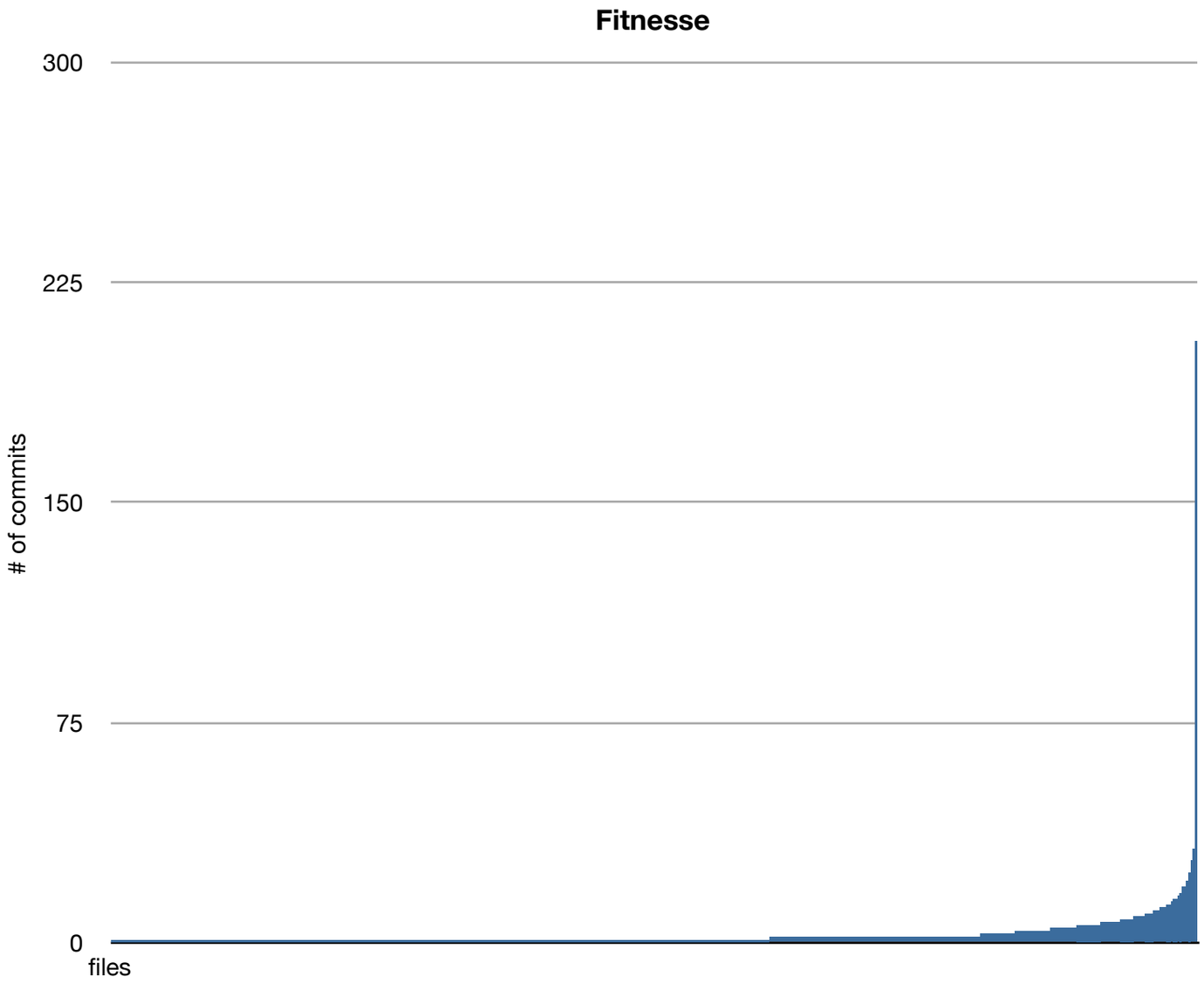
commit hash (sha1)
time/date stamp
committer
files
actual change

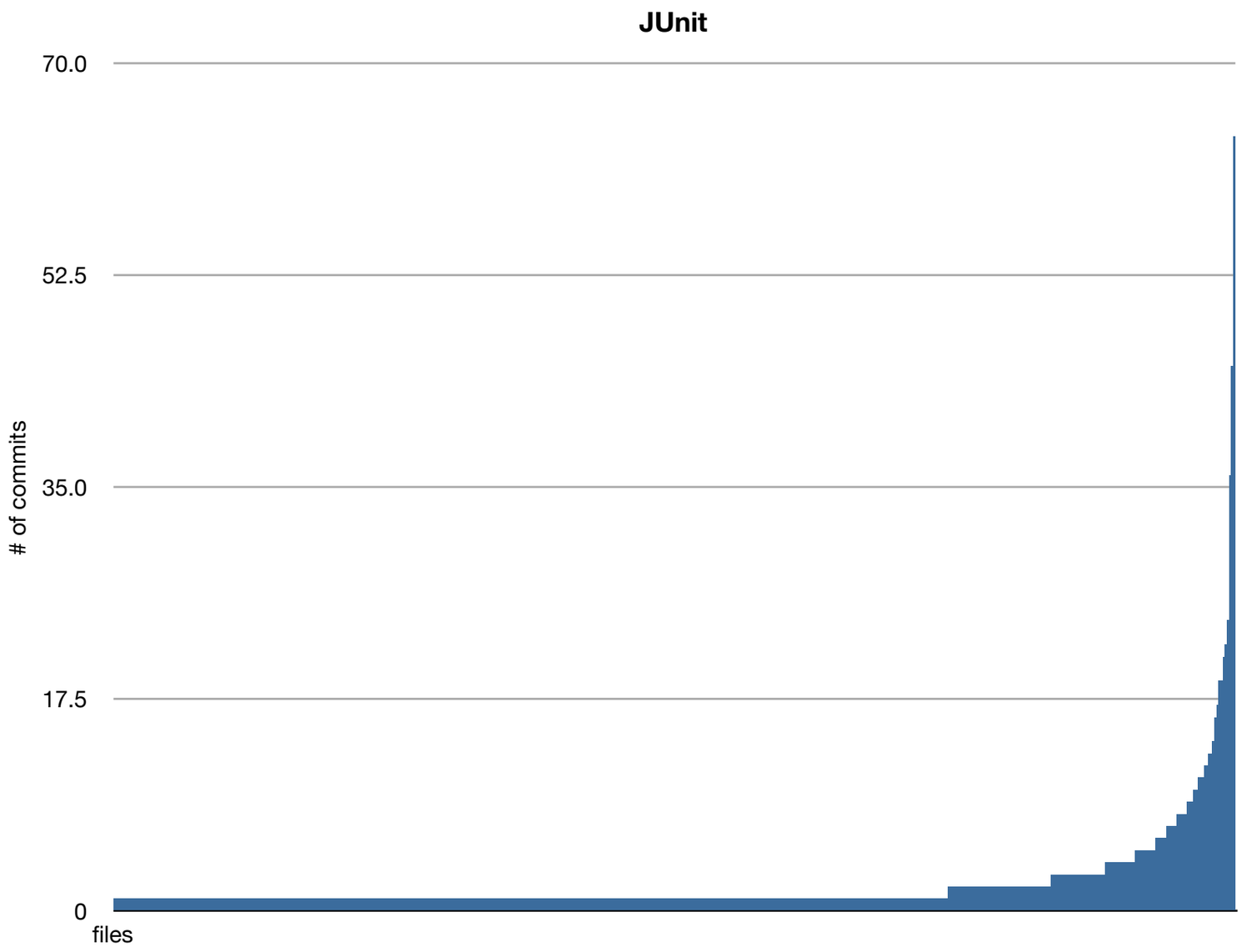
The Open/Closed Principle

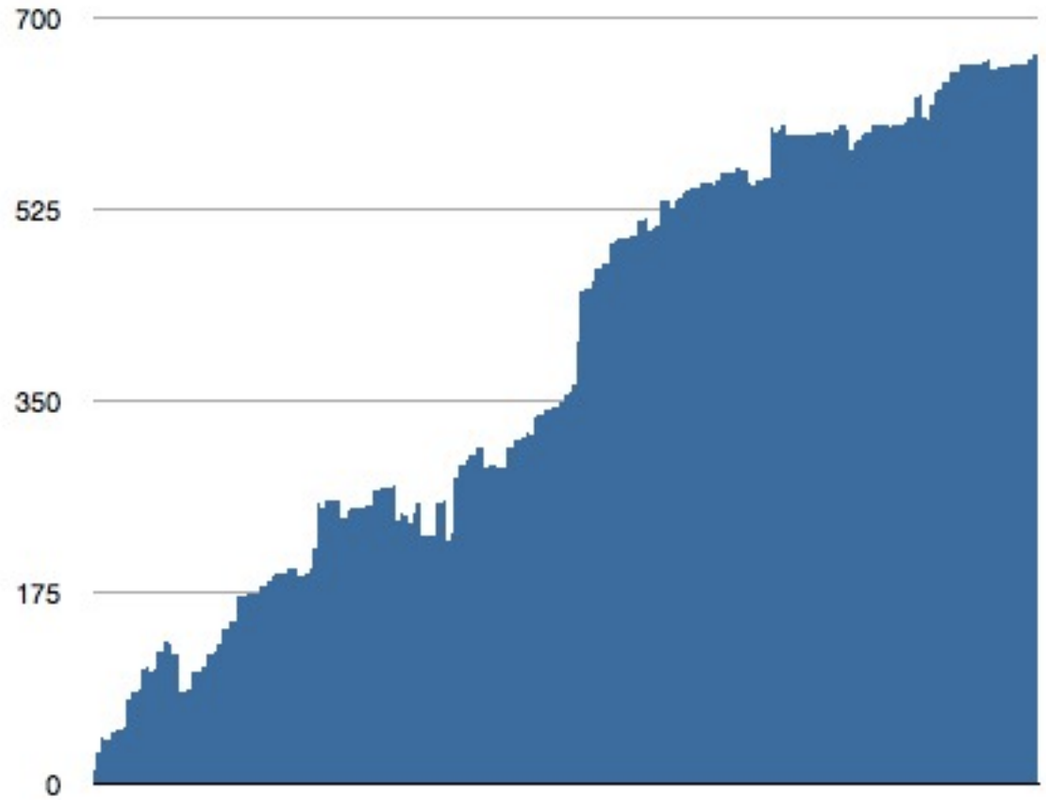
"*software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification*"

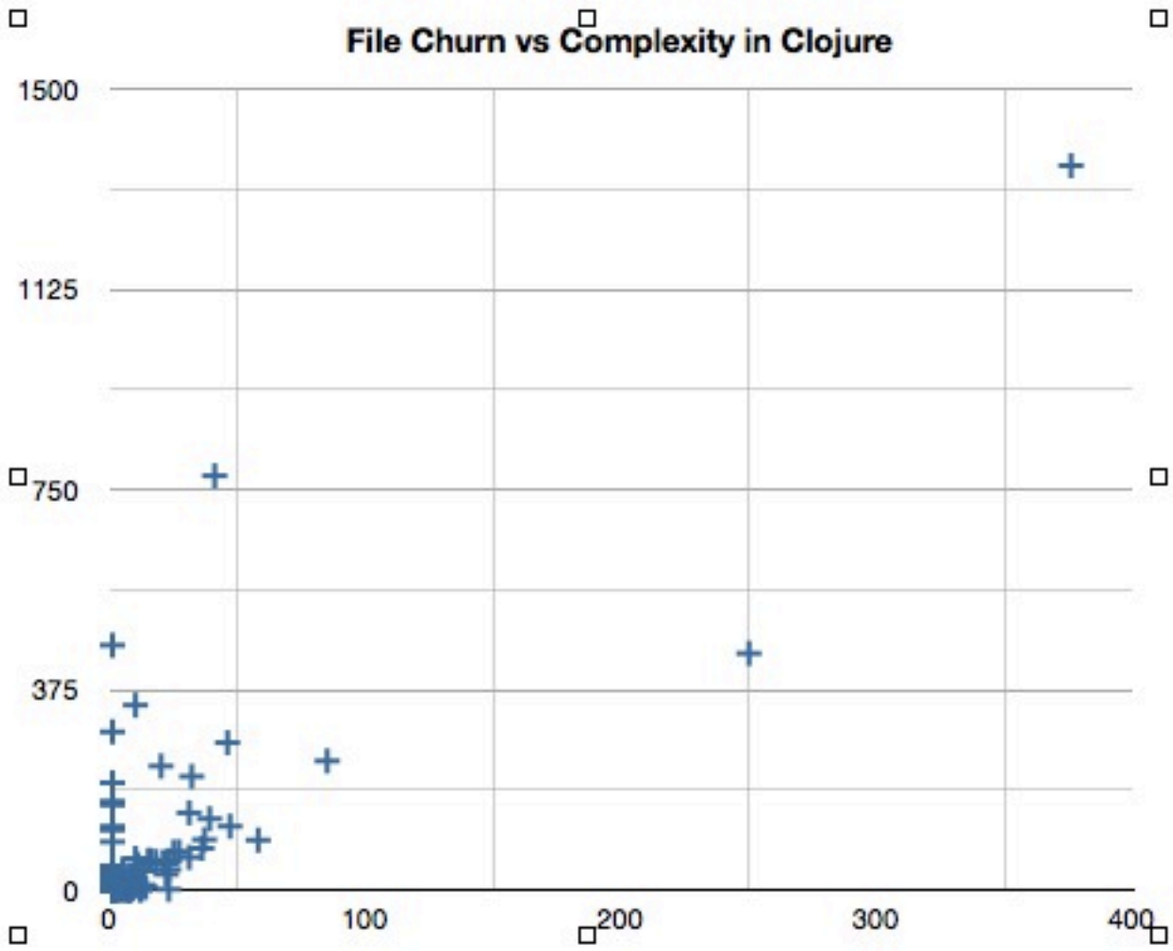
- Bertrand Meyer

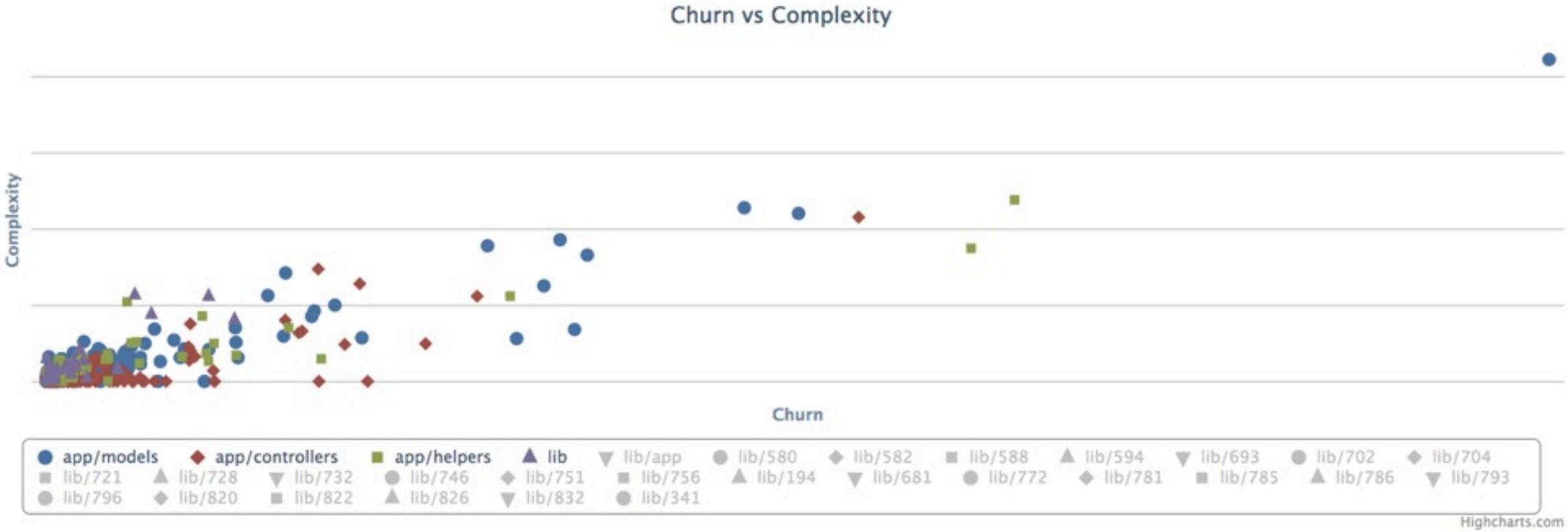

Clojure
400
300
200
100
0
# of commits
files

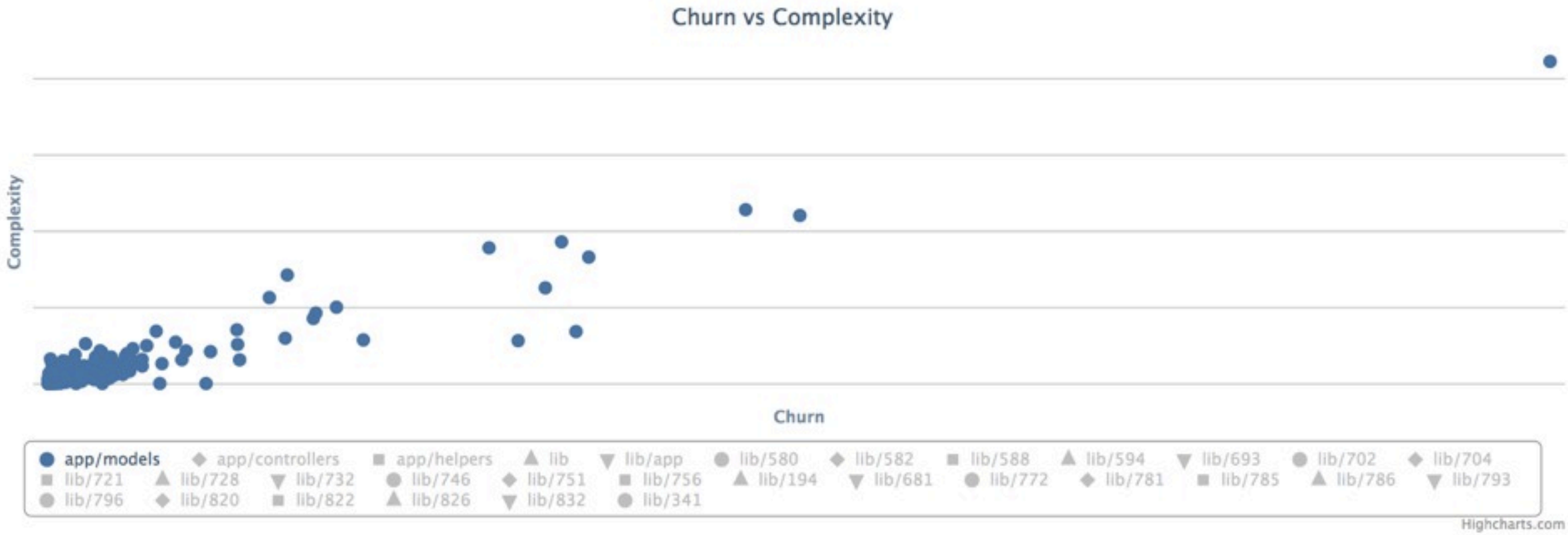

Fitnesse
300
225
150
75
0
# of commits
files

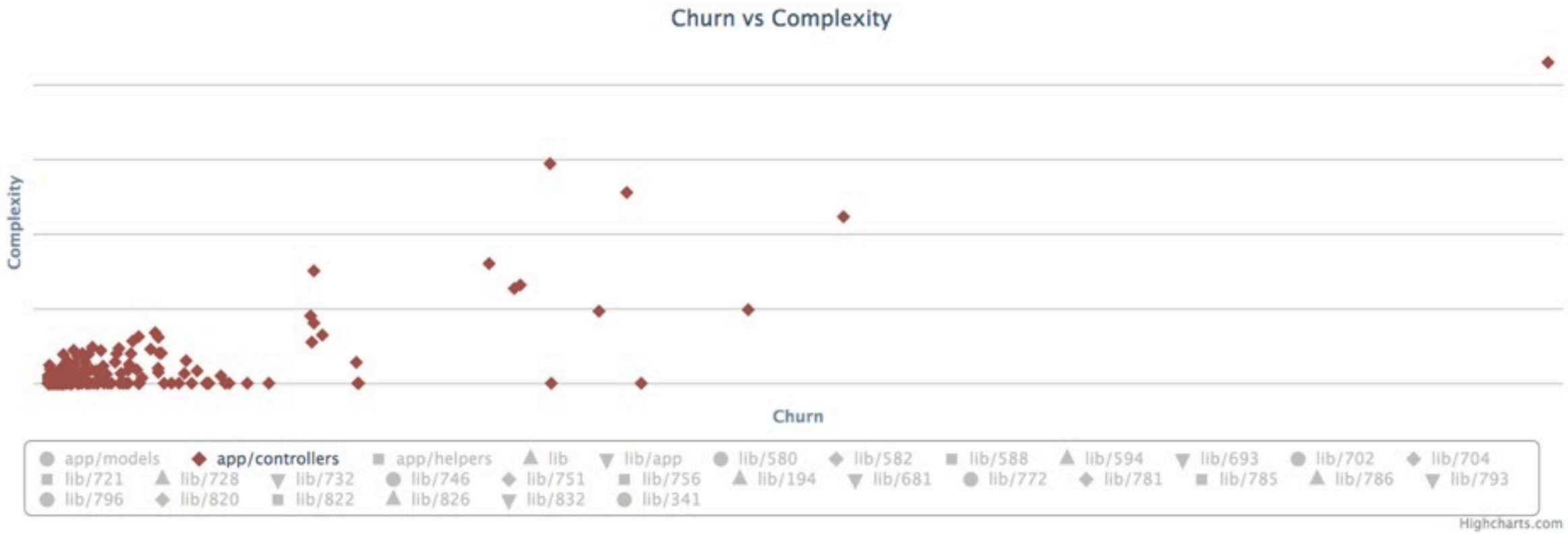

JUnit
70.0
52.5
35.0
17.5
0
# of commits
files


700
525
350
175
0

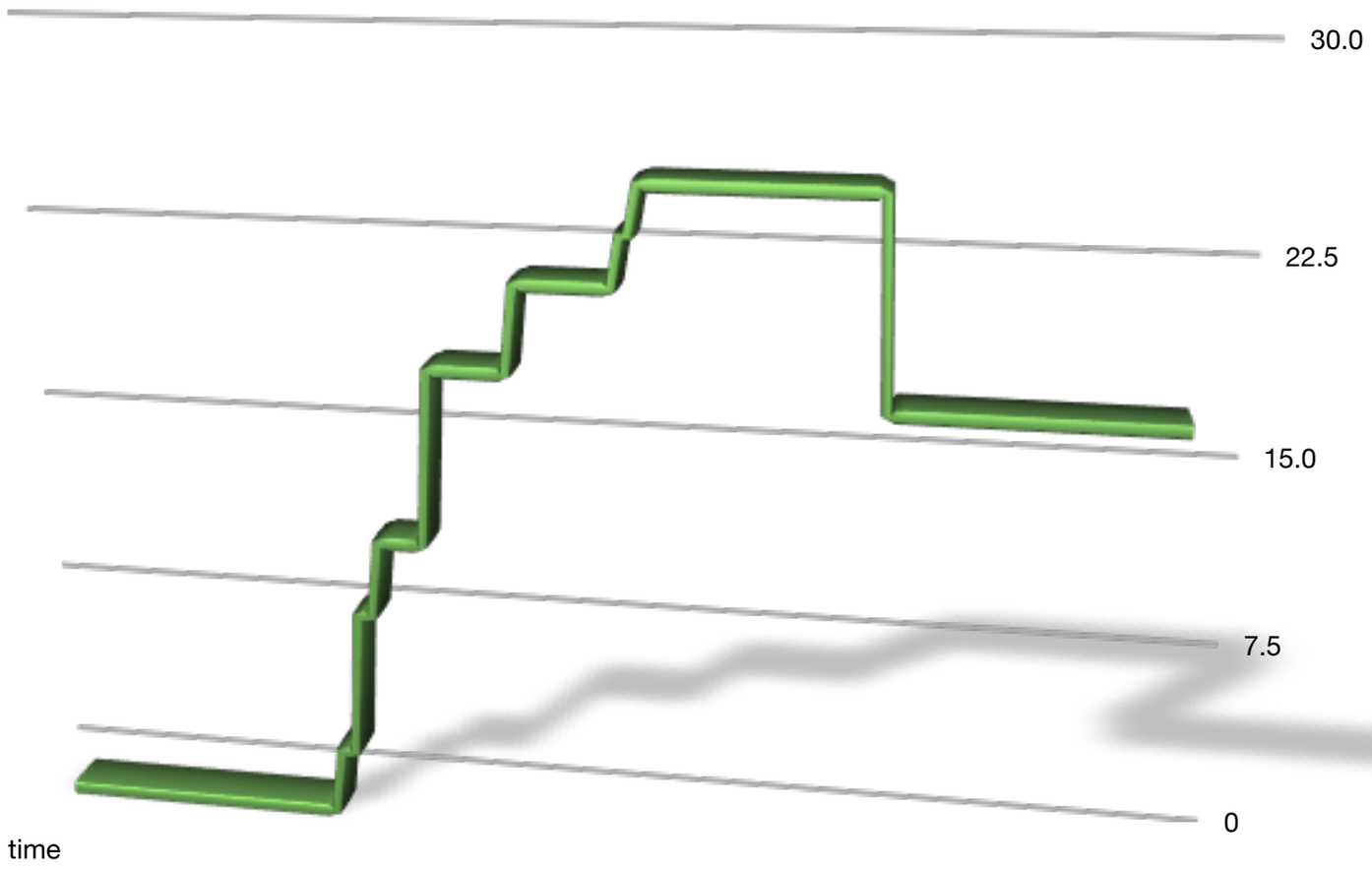

File Churn vs Complexity in Clojure
1500
1125
750
375
0
0
100
200
300
400
For each .java file, X is the the number of commits and Y is the total complexity of the file


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

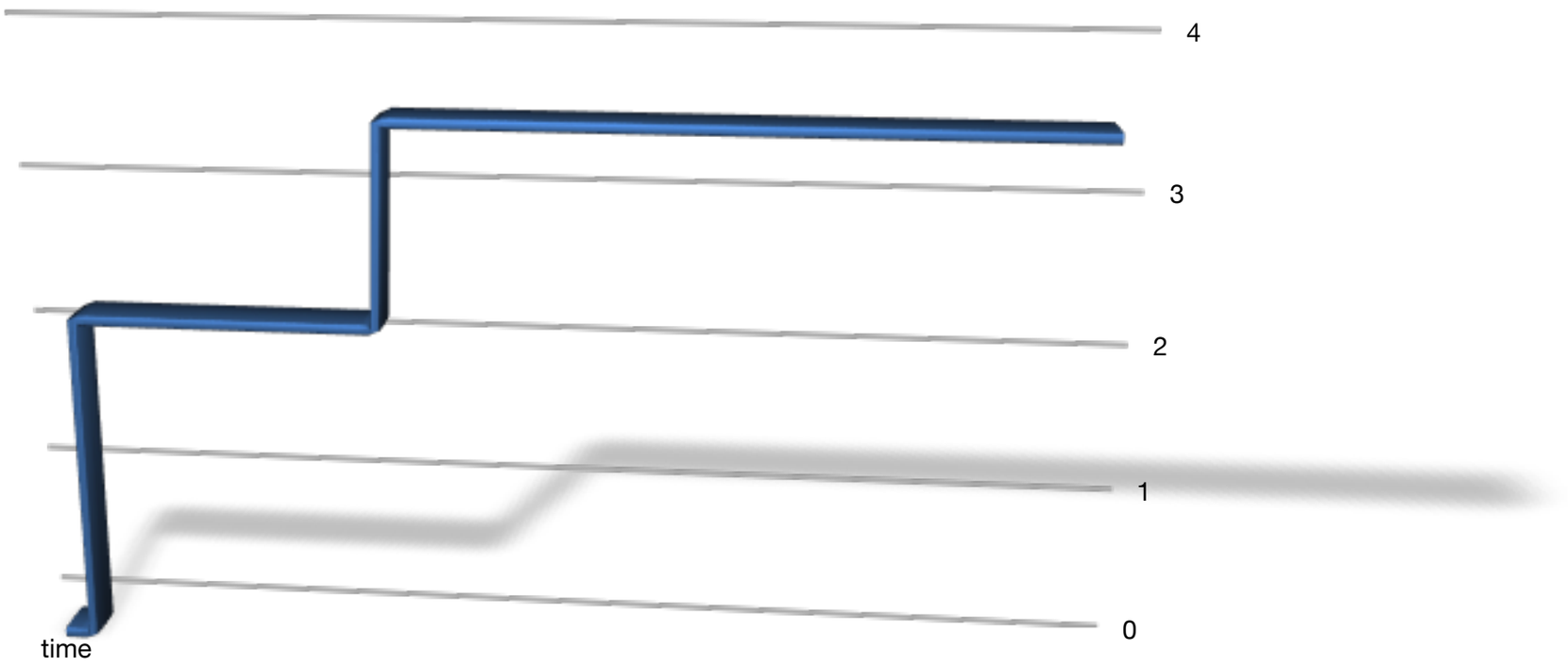

Churn vs Complexity
Complexity
Churn
app/models
app/controllers
app/helpers
lib
lib/app
lib/580
lib/582
lib/588
lib/594
lib/693
lib/702
lib/704
lib/721
lib/728
lib/732
lib/746
lib/751
lib/756
lib/194
lib/681
lib/772
lib/781
lib/785
lib/786
lib/793
lib/796
lib/820
lib/822
lib/826
lib/832
lib/341
Highcharts.com

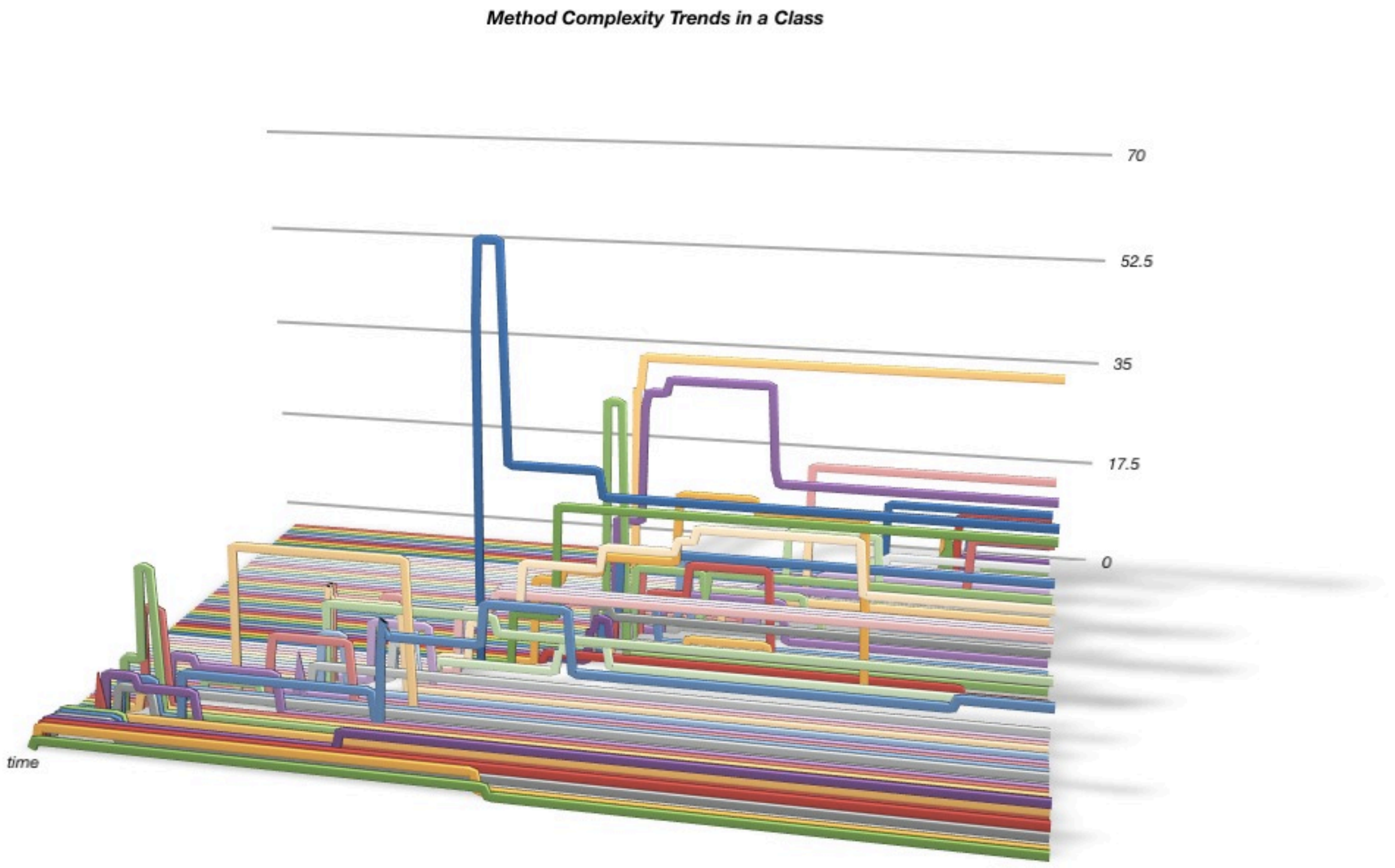

Churn vs Complexity
Complexity
Churn
app/models
app/controllers
app/helpers
lib
lib/app
lib/580
lib/582
lib/588
lib/594
lib/693
lib/702
lib/704
lib/721
lib/728
lib/732
lib/746
lib/751
lib/756
lib/194
lib/681
lib/772
lib/781
lib/785
lib/786
lib/793
lib/796
lib/820
lib/822
lib/826
lib/832
lib/341
Highcharts.com

A Method Lifeline



Another Method




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

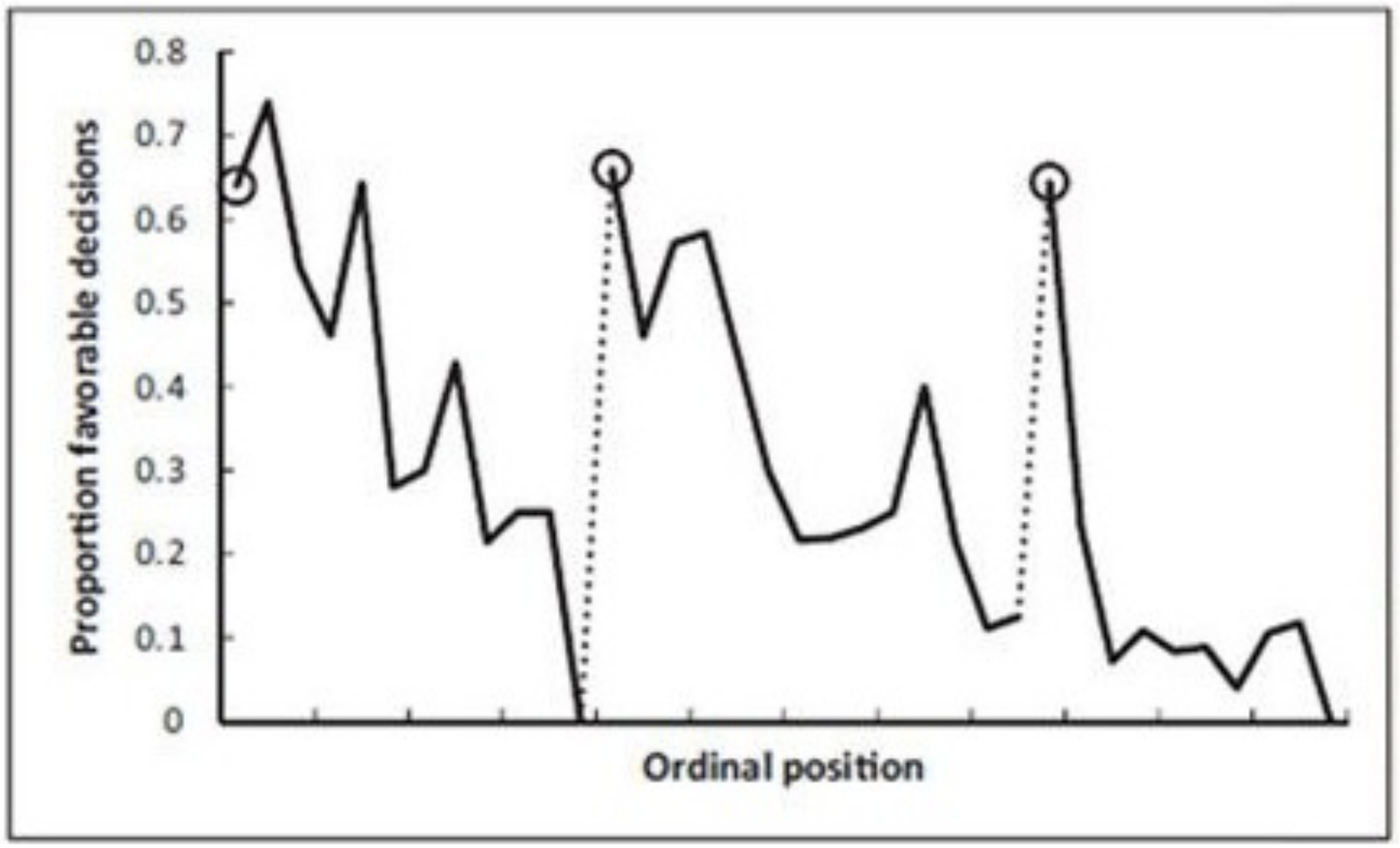
If you want parole, have your case heard right after lunch

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.

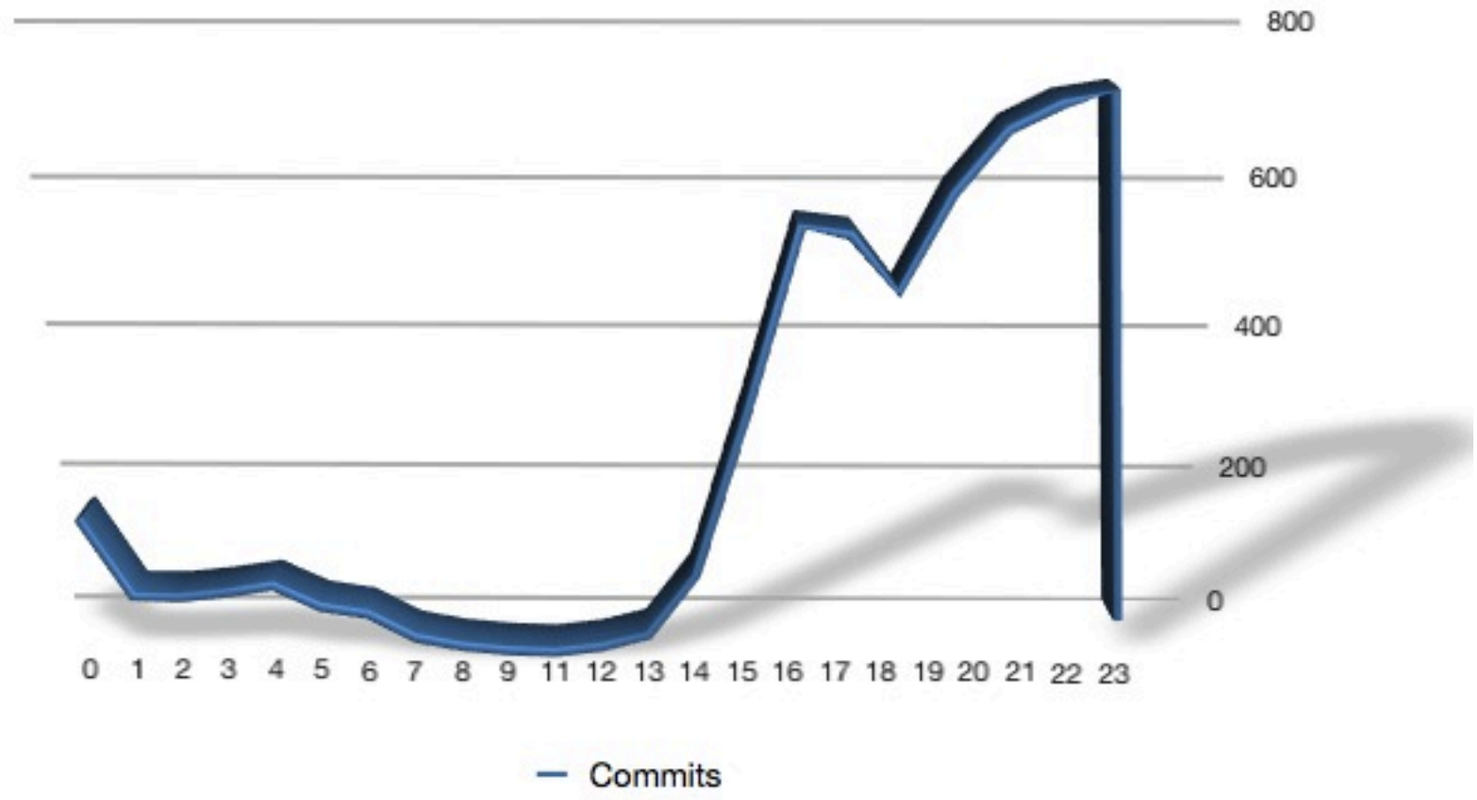
Overall, judges were much more likely to accept prisoners' requests for parole at the beginning of the day than the at end. Moreover, a prisoner's chances of receiving parole more than doubled if his case was heard at the beginning of one of the three sessions, rather than later on in the session. More 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.

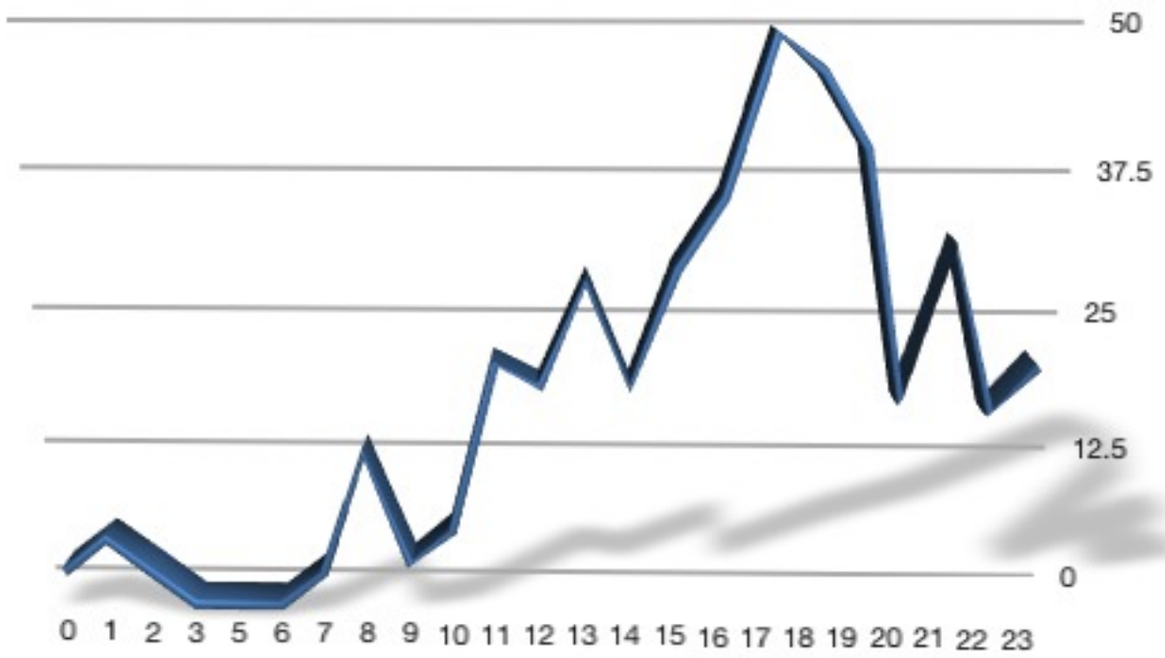


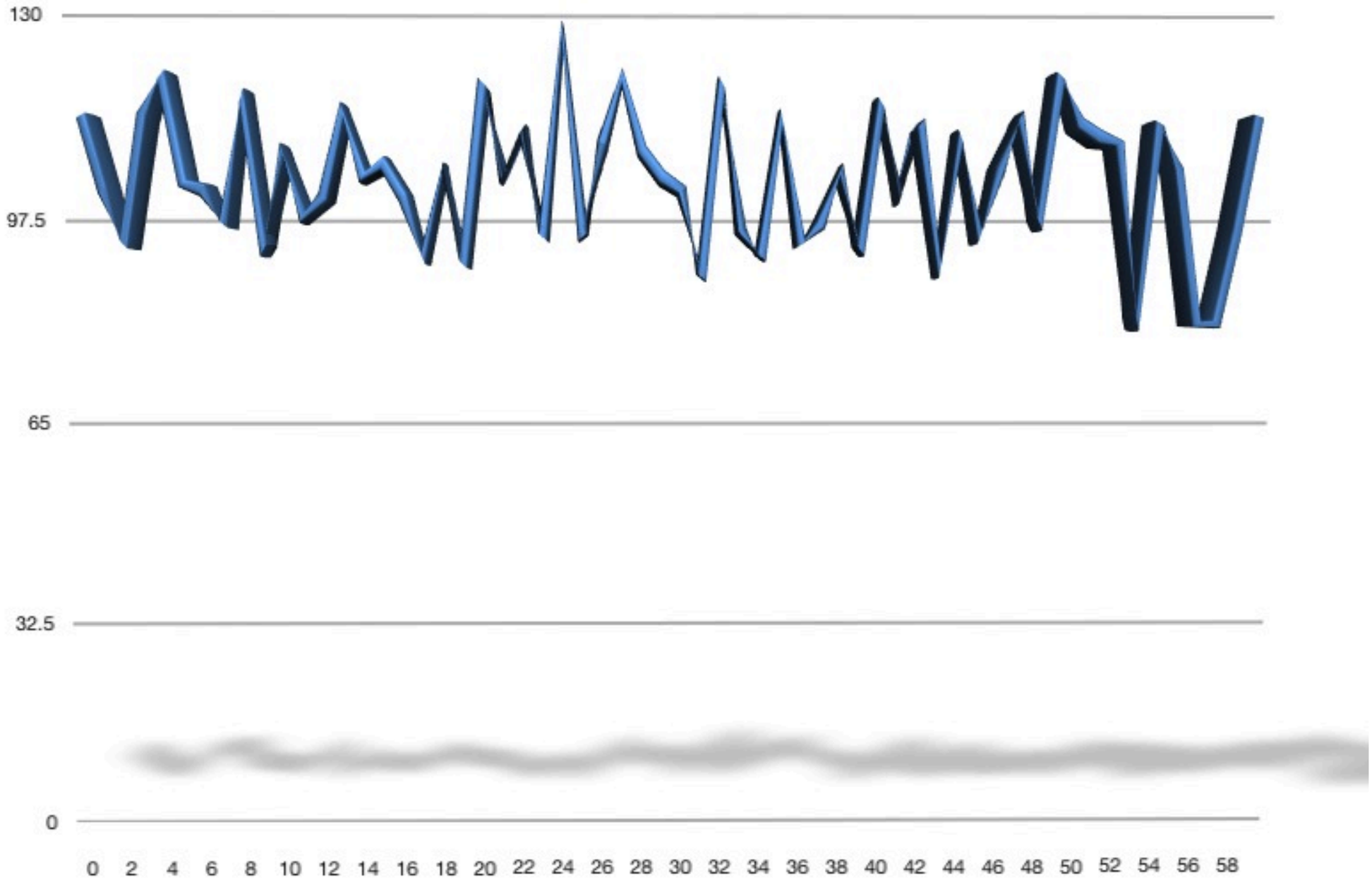
Each judge took two breaks. One at mid-morning beginning as early as 9:45 a.m. or as late as 10:30 a.m., and a lunch break that began between 12:45 p.m. and 2:10 p.m.

http://www.physorg.com/news/2011-04-early-lunch.html

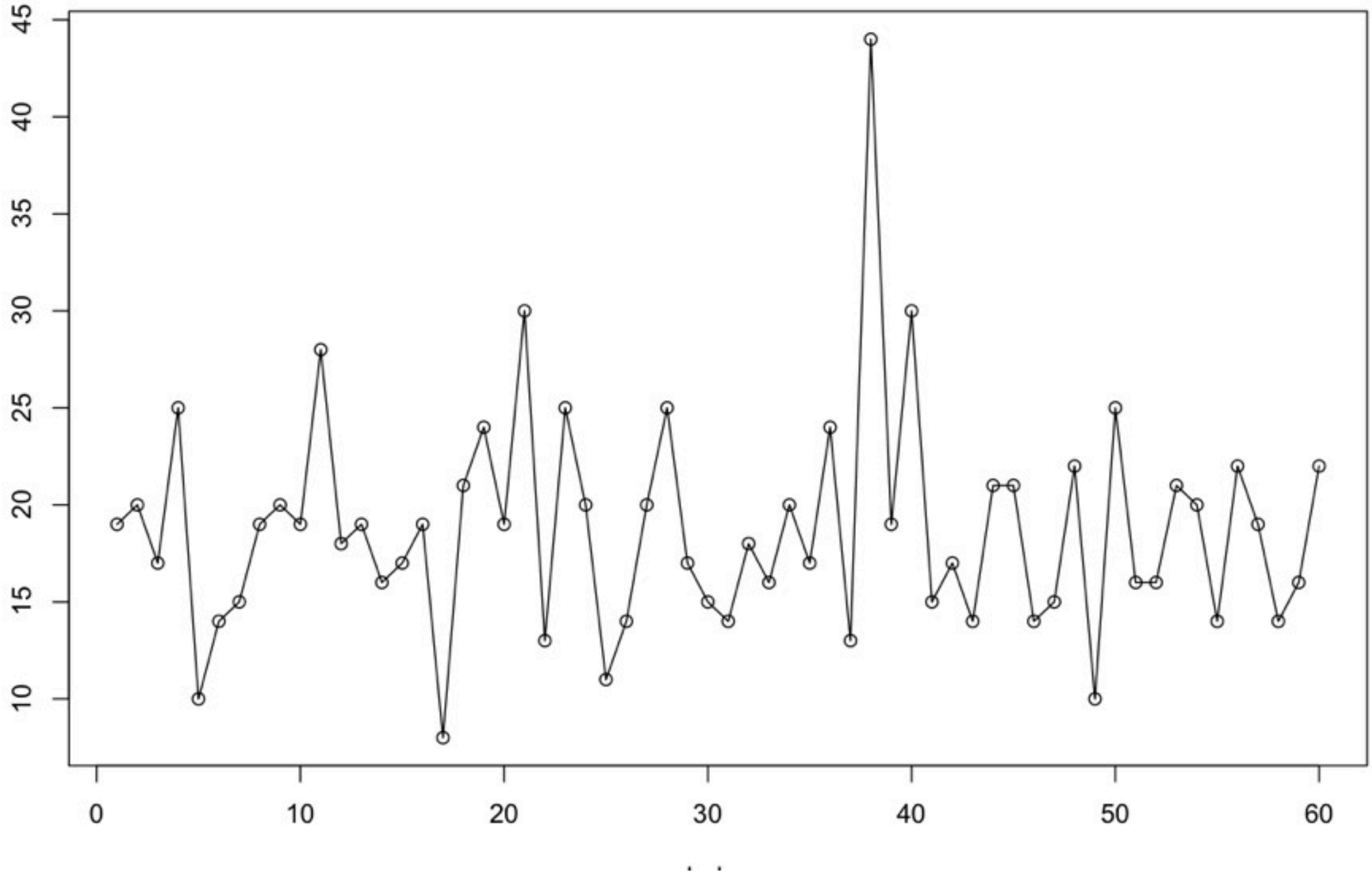
"You're always surprised when you find effects where you don't want to find them," Jonathan Levav of Columbia University said in a telephone interview. "If you're a social scientist it gets you excited. But, as an ordinary citizen, you don't want to find this."


800
600
400
200
0
0 1 2 3 4 5 6 7 8 9 11 12 13 14 15 16 17 18 19 20 21 22 23
Commits


50
37.5
25
12.5
0
0 1 2 3 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

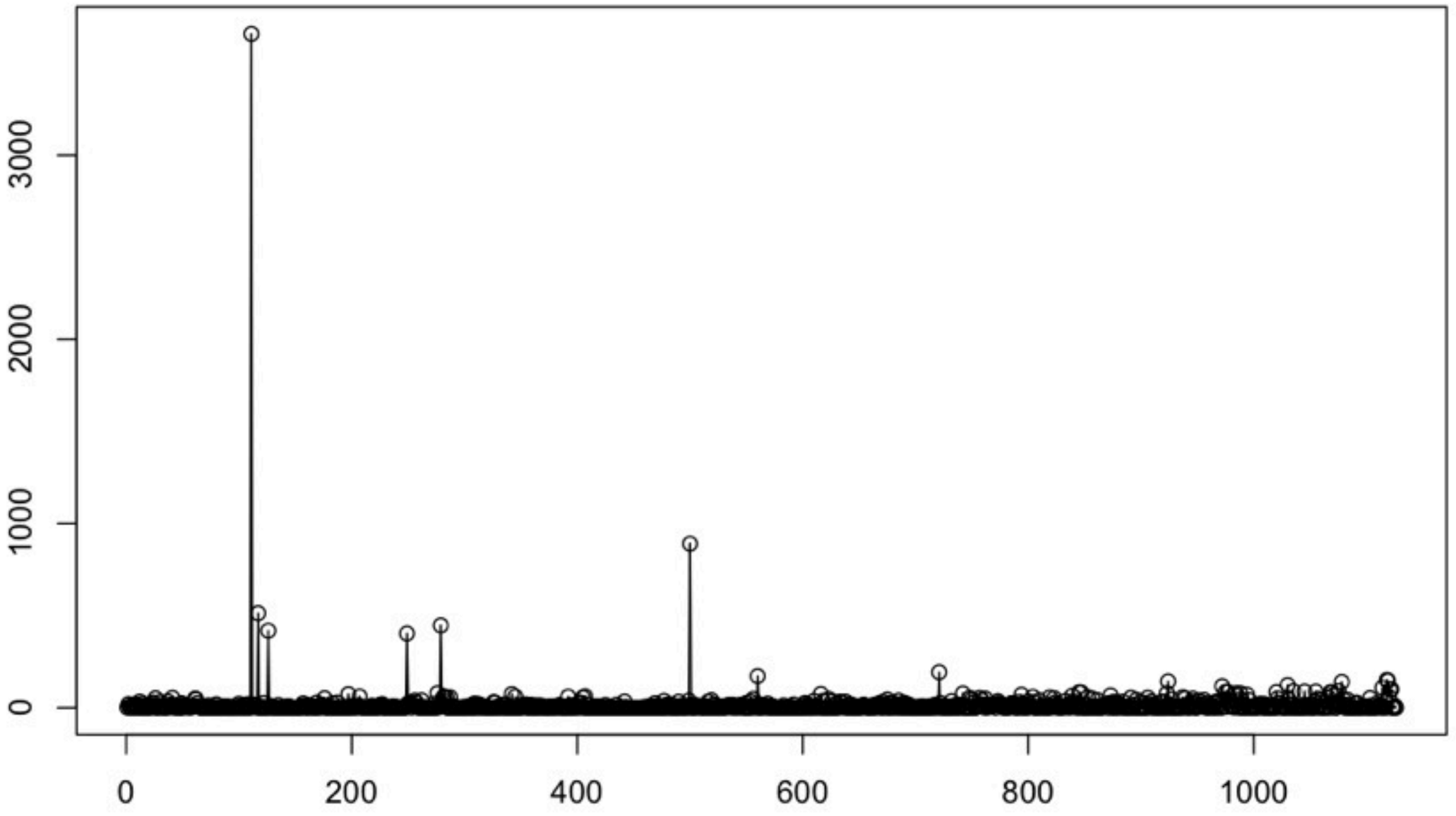

130
97.5
65
32.5
0
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58

Commits per minute of each hour



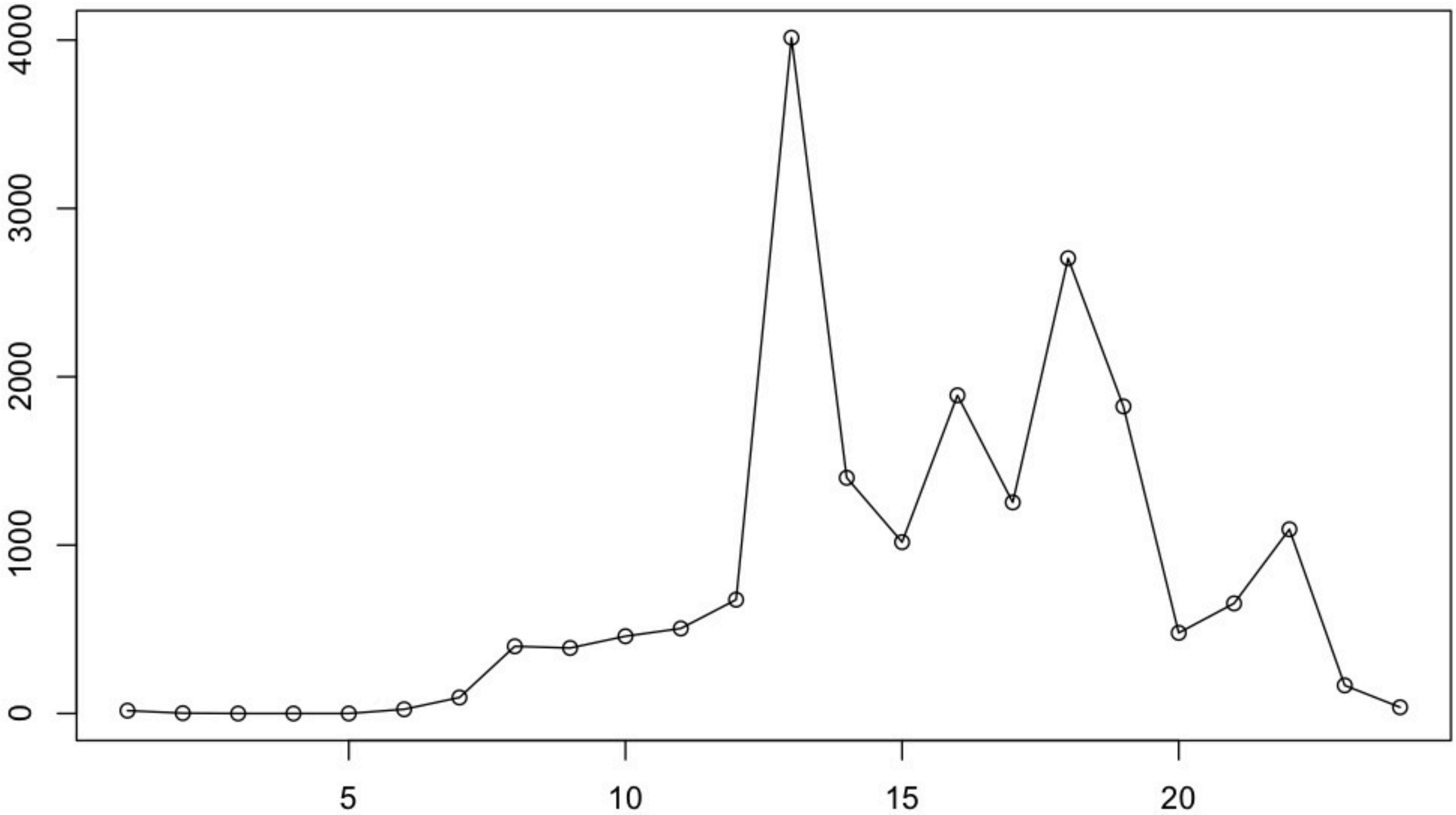
repo.commits.map {|_,d| d.min }.freq

Added Complexity Over Time

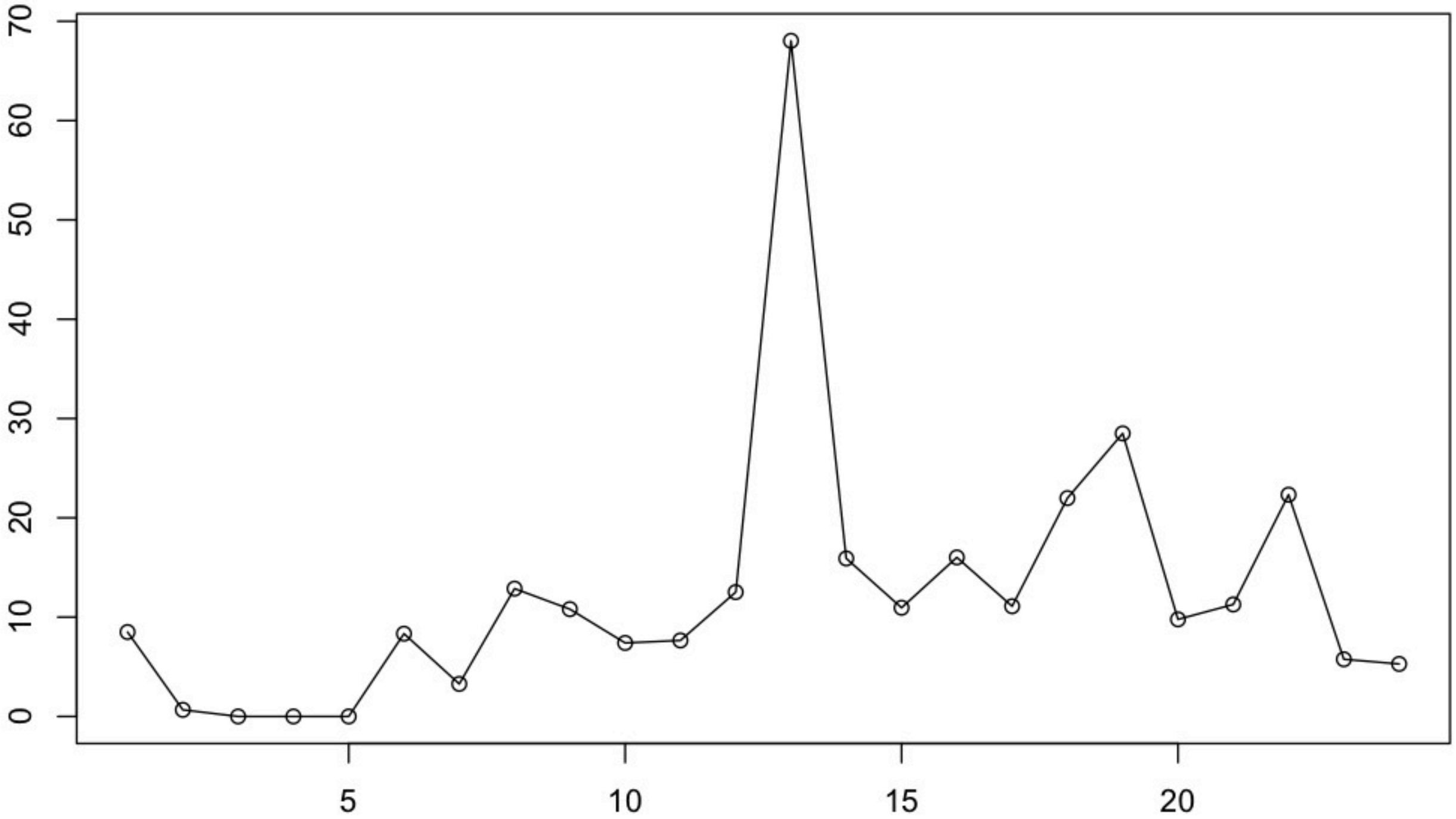


repo.commits.map {|c,_| repo.commit(c).added_complexity.to_i }

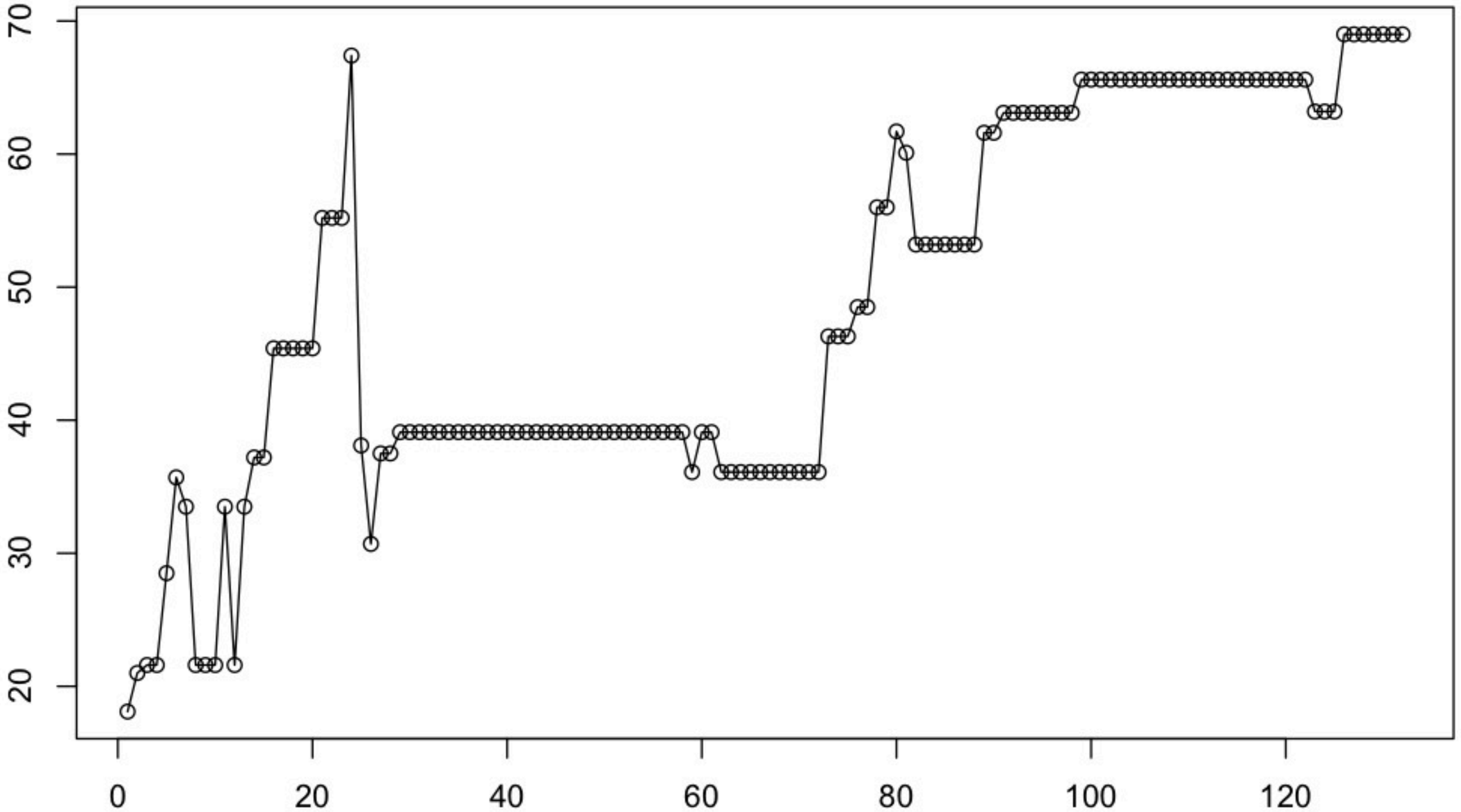
Amount of Complexity Added by Hour of Day



Normalized by Commits

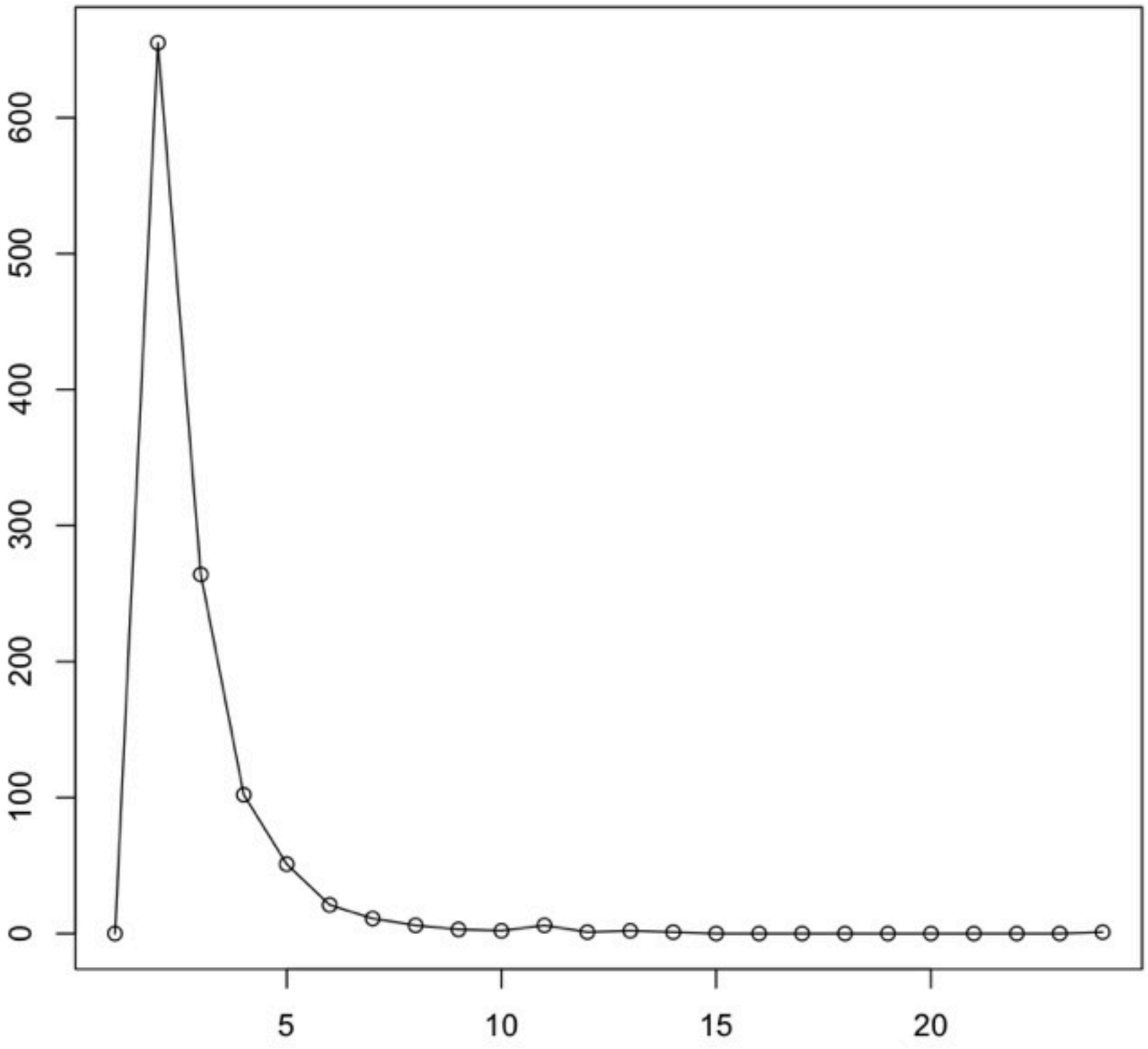


A Lifeline



repo.methods.select {|m| m.full_name == "FeelingsController#create" }.first.life_line

Number of Files Touched Per Commit



repo.events.group_by(&:commit).map { |sha, events| events.map(&:file_name).uniq.count }.freq

The Trending View

Methods Ascending

```
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
```

Static Views

Classes By Closure Date

[["DummiesController", 2008-04-21 13:03:08 -0700],
["Core::ActiveRecord::AttributeDefaults::ClassMethods", 2008-04-22 16:02:54 -0700],
["Legacy::Database", 2008-04-24 15:37:51 -0700],
["Core::ActiveRecord::AttributeDelegation::ClassMethods", 2008-04-24 20:46:58 -0700],
["Core::ActiveRecord::SkipValidationForHasOnes", 2008-04-29 21:54:32 -0700]]

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

[[["App", "Inventory"], 277],
[["Inventory", "Object"], 216],
[["Admin", "Inventory"], 195],
[["Inventory", "User"], 188],
[["Inventory", "Users"], 171],
[["Inventory", "Deals"], 167],
[["App", "Object"], 159],
[["App", "InventoryController"], 152],
[["Inventory", "Order"], 149],
[["User", "Users"], 149],
[["App", "User"], 143],
[["Inventory", "InventoryController"], 143],
[["Api", "Inventory"], 141],
[["Admin", "App"], 136],
[["Campaign", "Orders"], 134]]

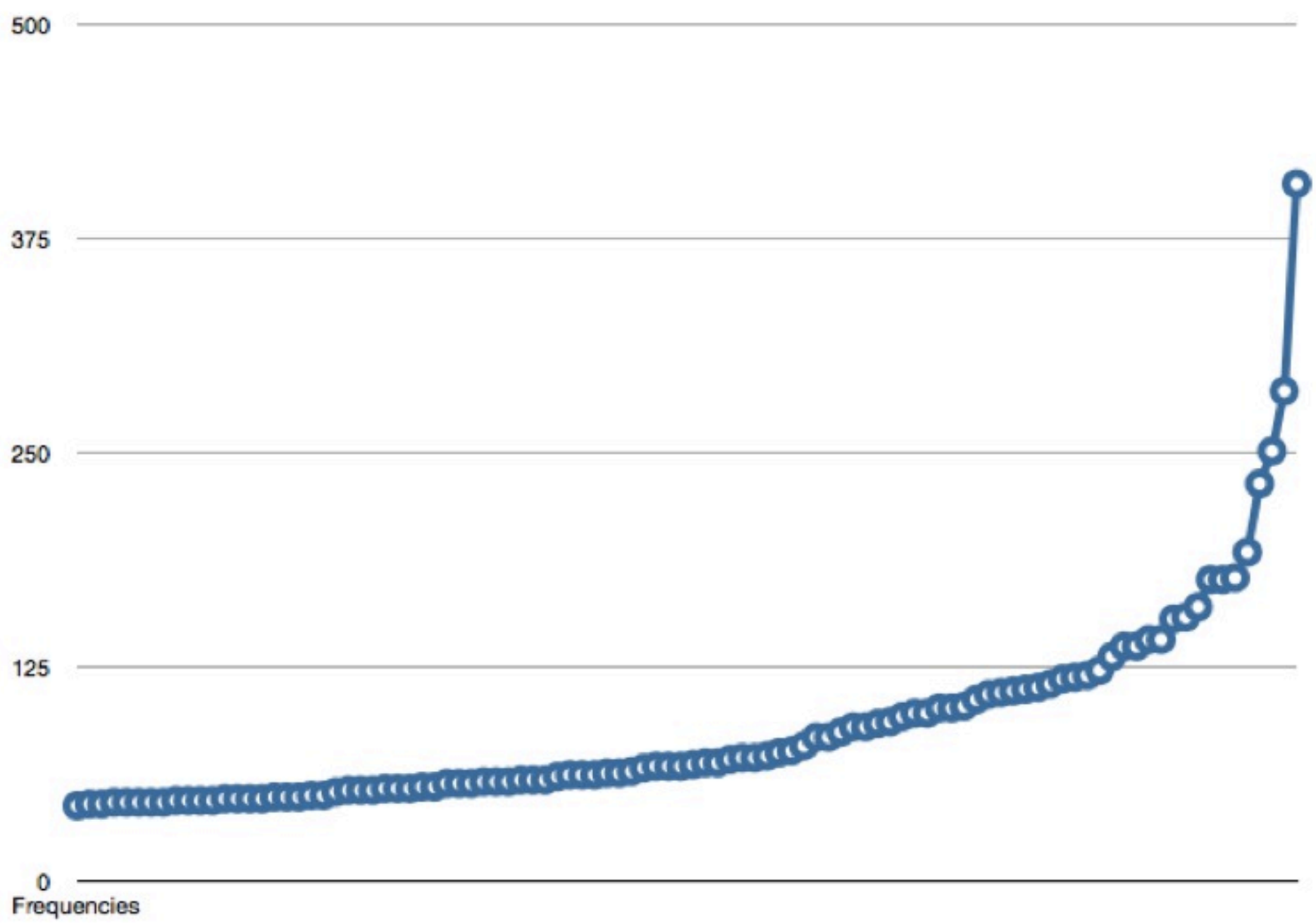
Temporal Correlation of Class Changes

```
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
```

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:



Enki - A Rails Blogging Platform

5 Unique Committers ["Xavier", "Jason", "Zach", "Pedro", "Gaelian"]

637 method events

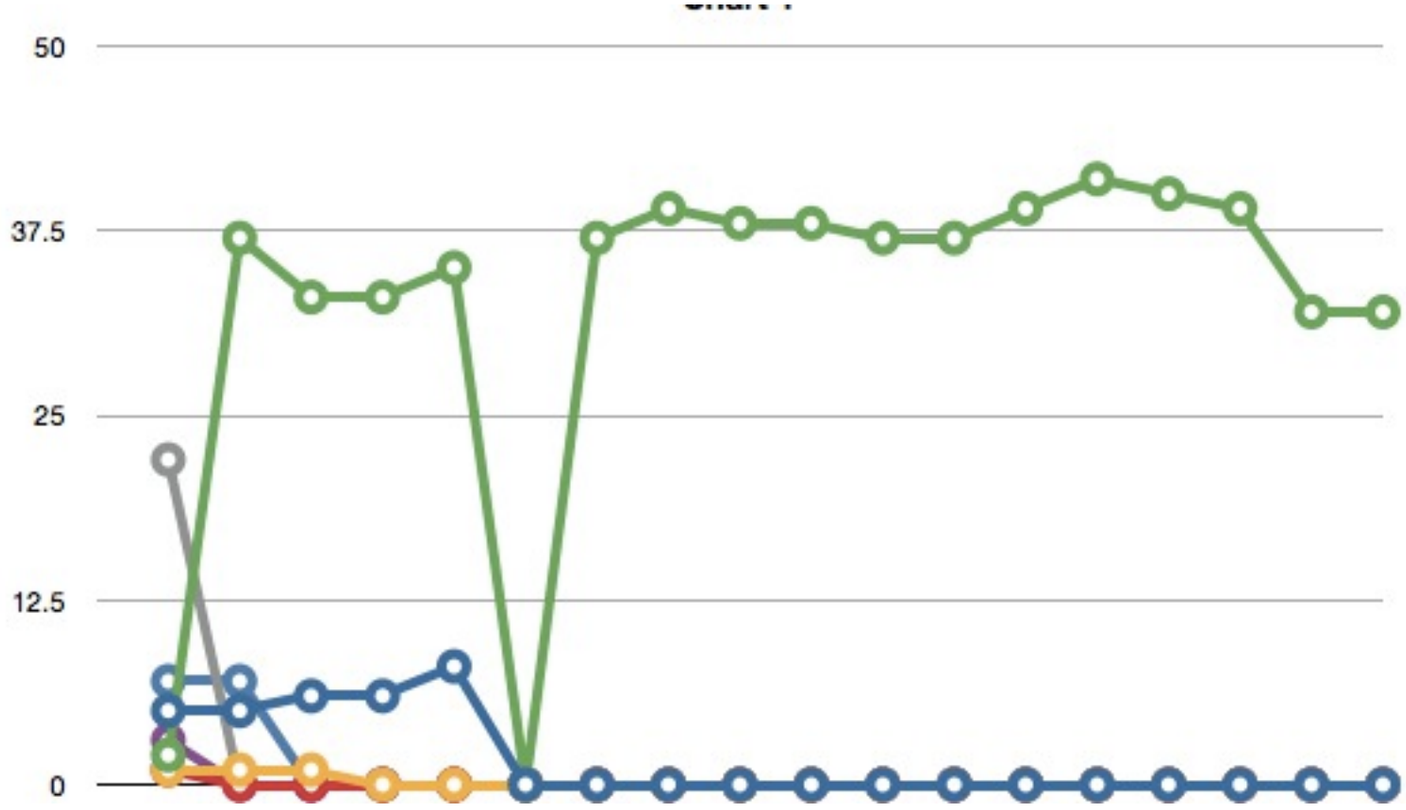
Spec to method ratios by committer:

[0.09245283018867924, "Xavier"],
[0.05084745762711865, "Jason"],
[0.0, "Zach"],
[0.6666666666666666, "Pedro"],
[0.0, "Gaelian"]]

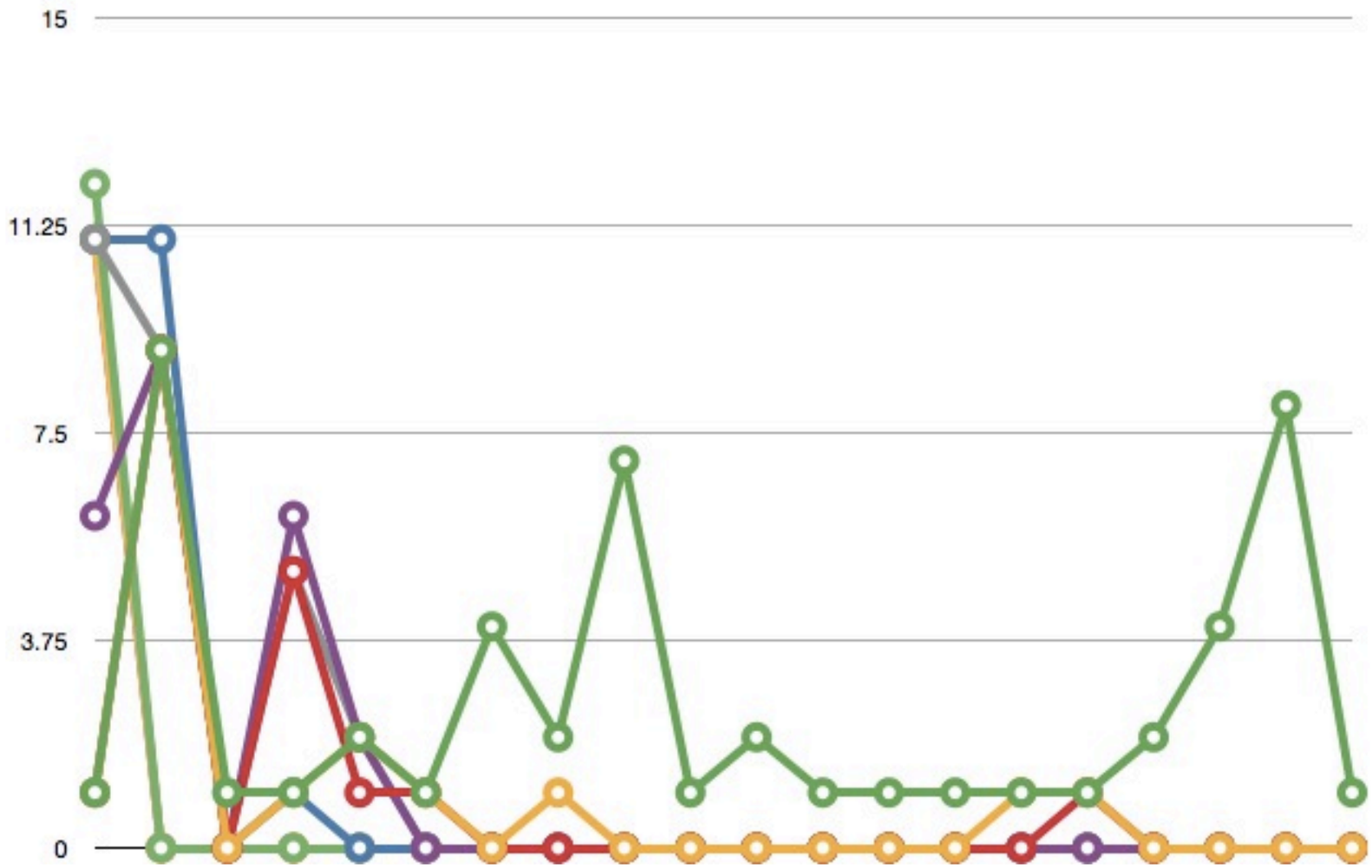
Number of Method Modifications:

Zach => 6
Xavier=> 167
Jason => 10
Pedro => 1

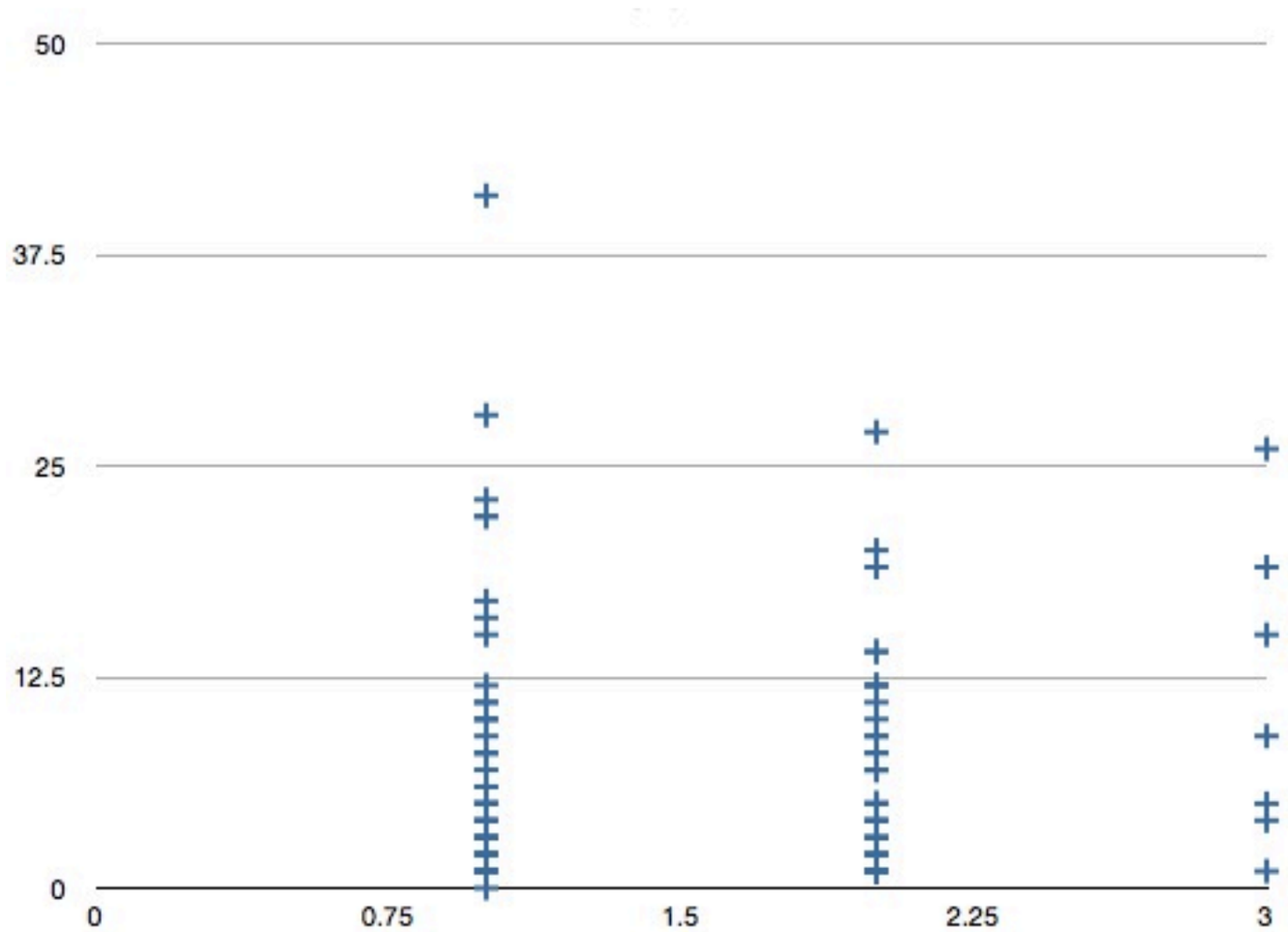
Enki - CommentController Class



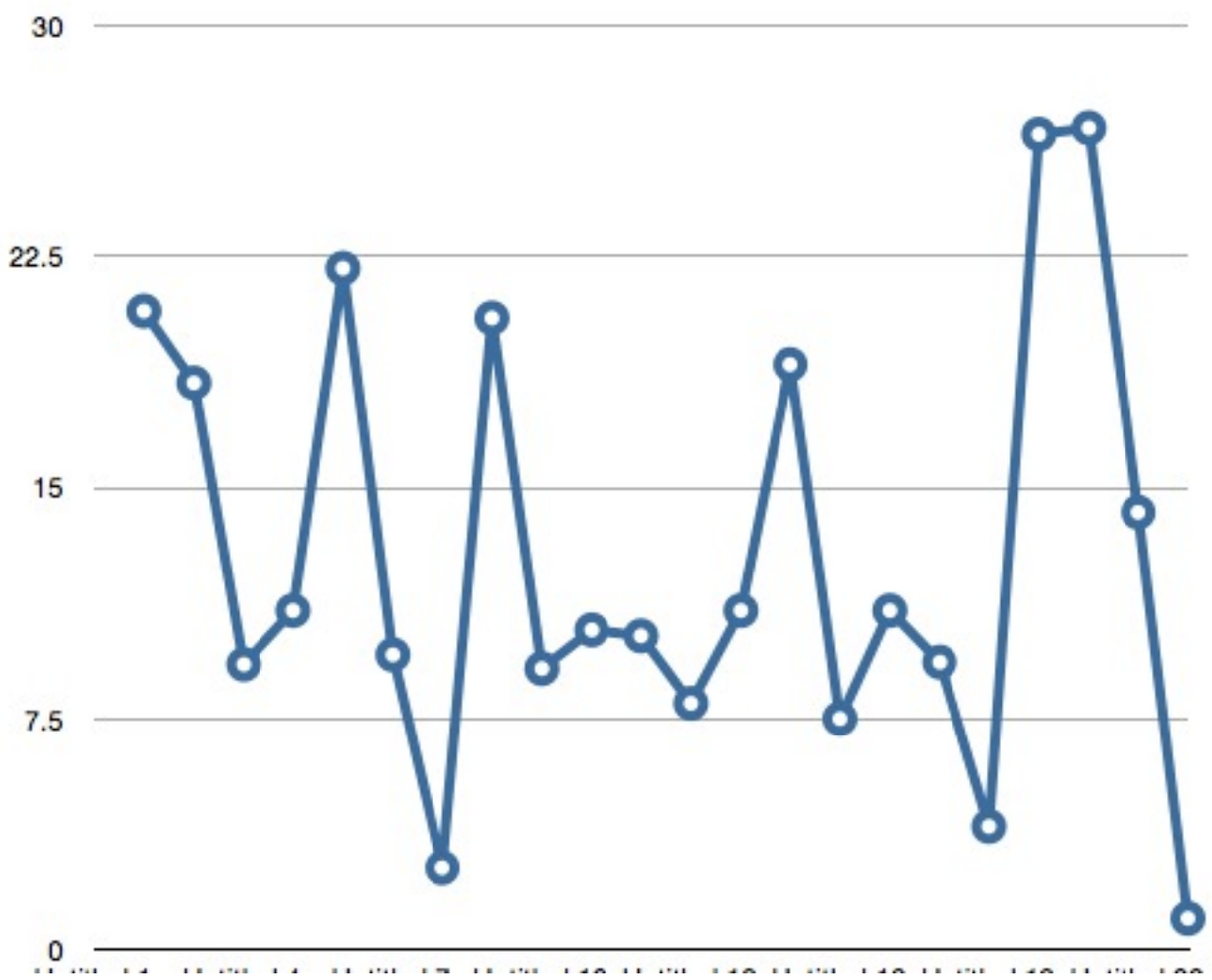
Enki - Post Class



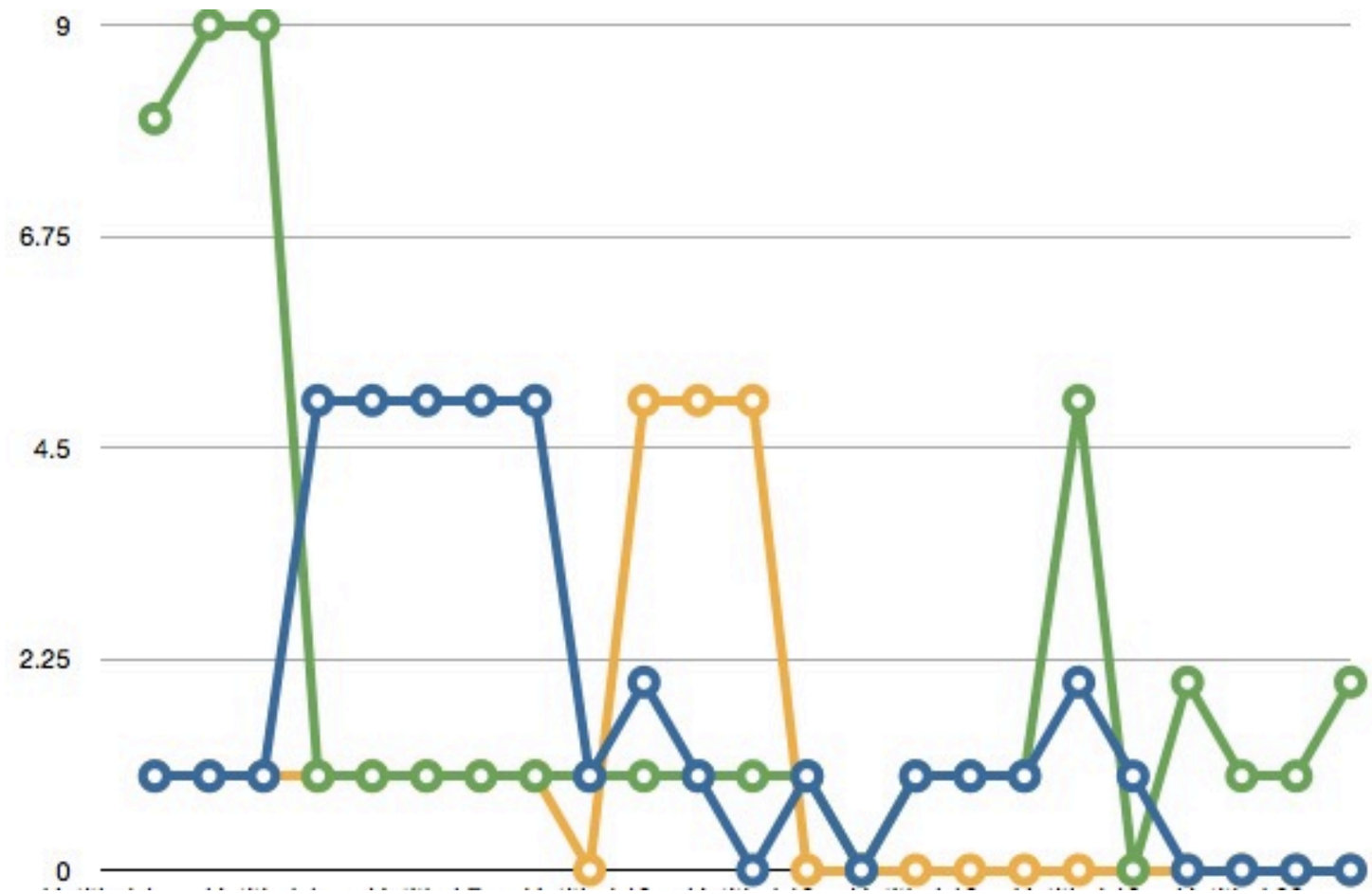
Enki - Ownership Effect



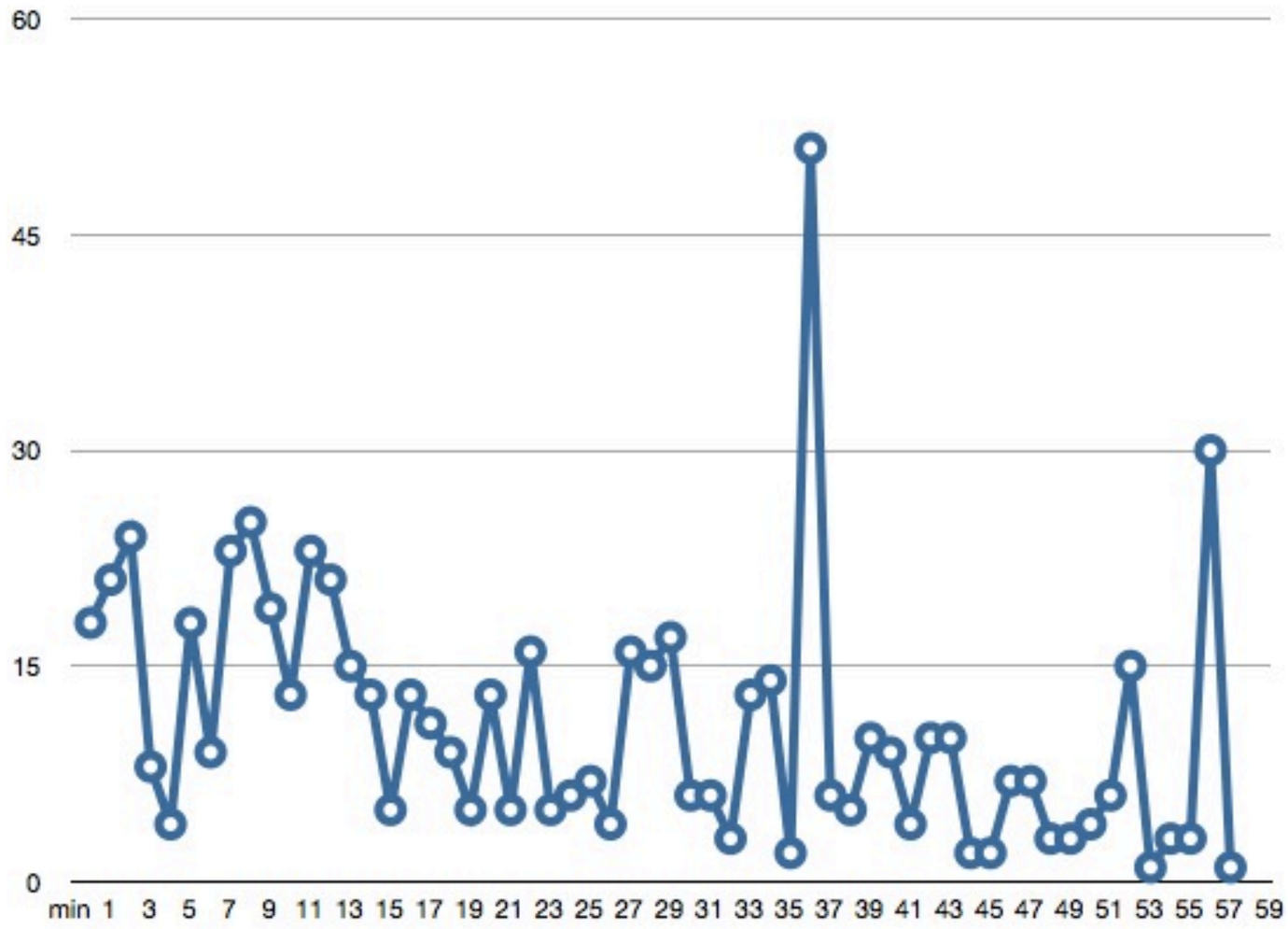
Enki - Average Lines Per Commit By Month



Enki - Spec Lifelines



Enki - Hour Profile



MercuryApp

5 Unique Committers ["Sarah", "coreyhaines", "Cory", "Spencer", "sarah"]

7788 method events

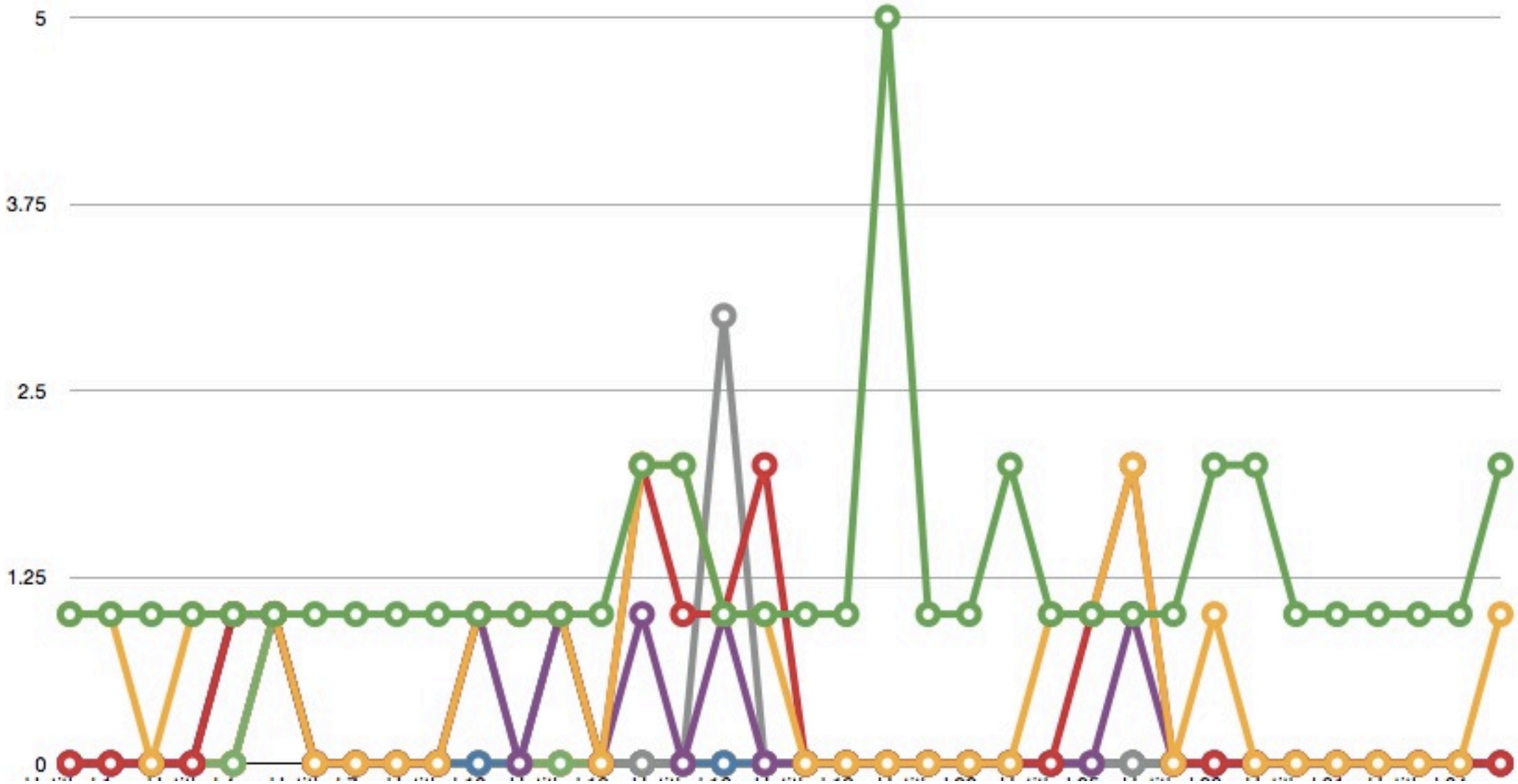
Spec to method ratios by committer:

[0.40381791483113066, "Sarah"],
[0.5220038748962081, "coreyhaines"],
[0.0, "Cory"],
[0.0, "Spencer"],
[0.517106200997861 8, "sarah"]

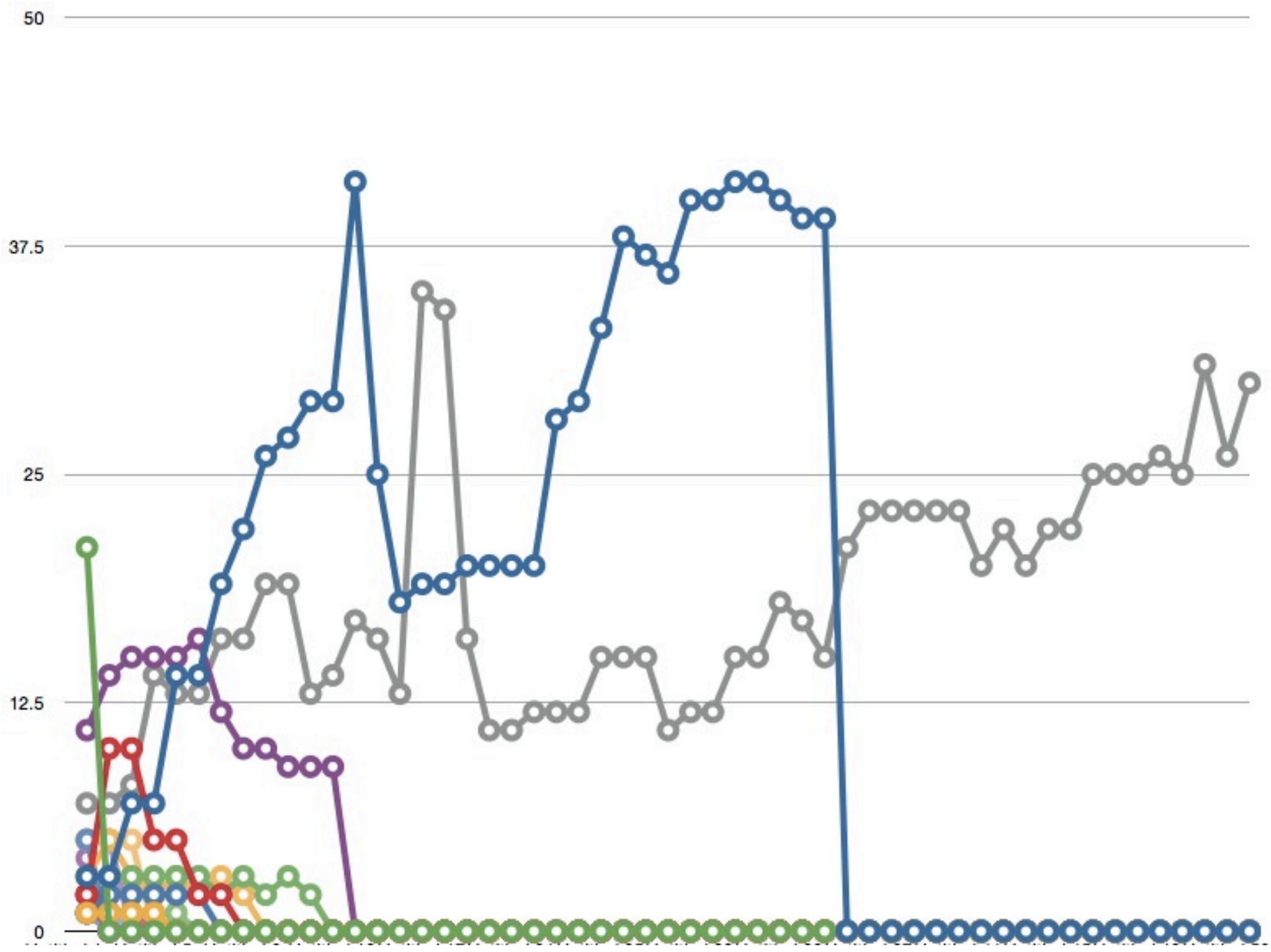
Number of Method Modifications:

Cory => 629
Sarah=> 739
Spencer => 2

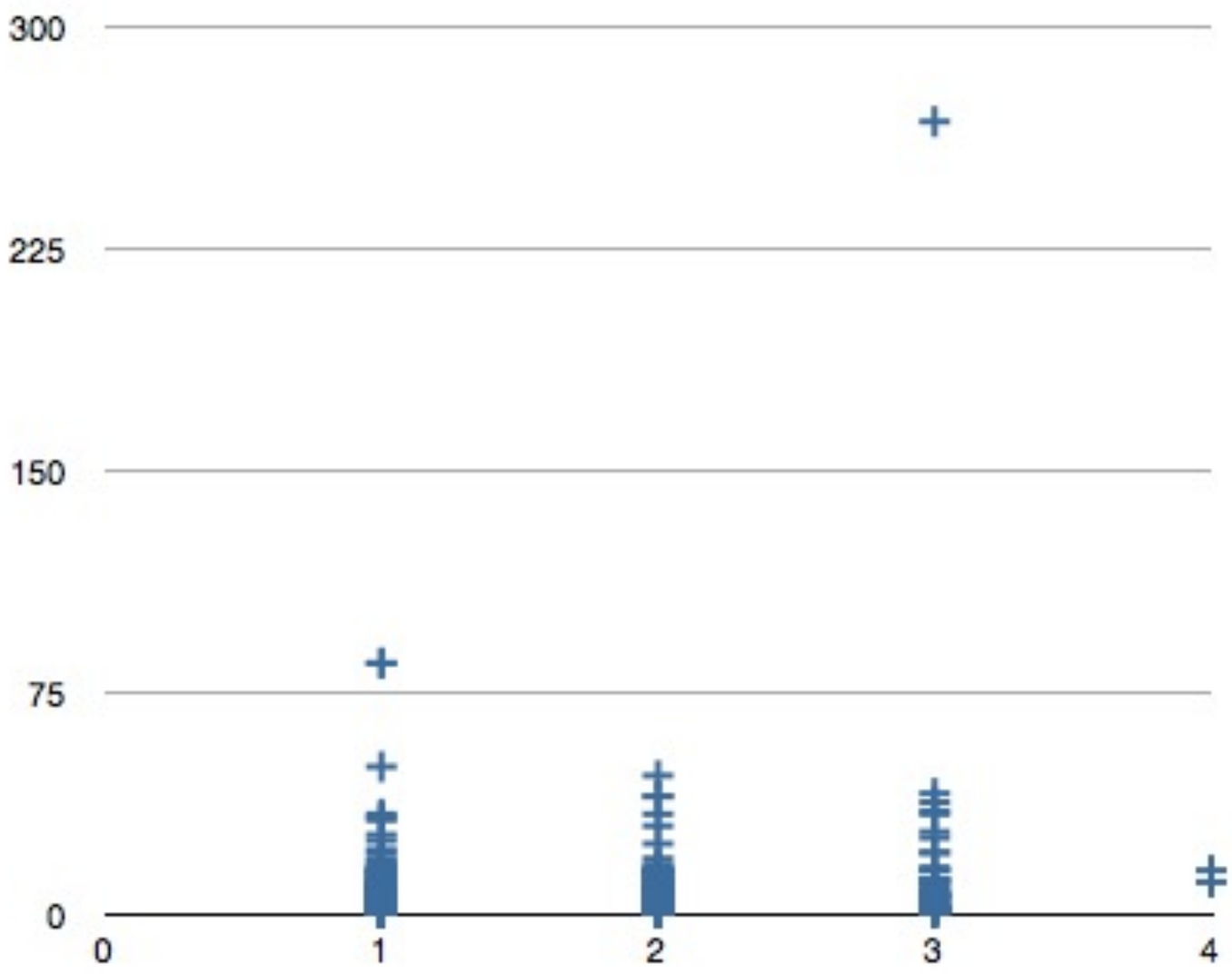
MercuryApp - User Class



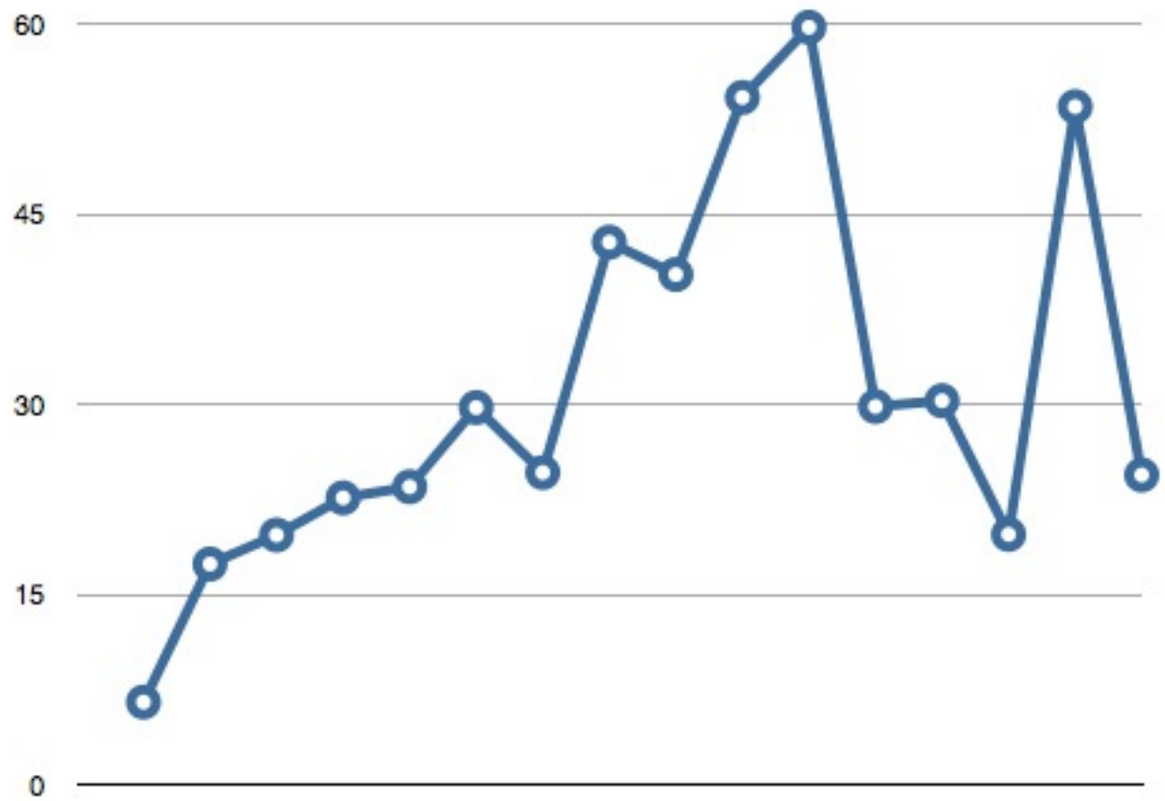
MercuryApp - FeelingsController Class



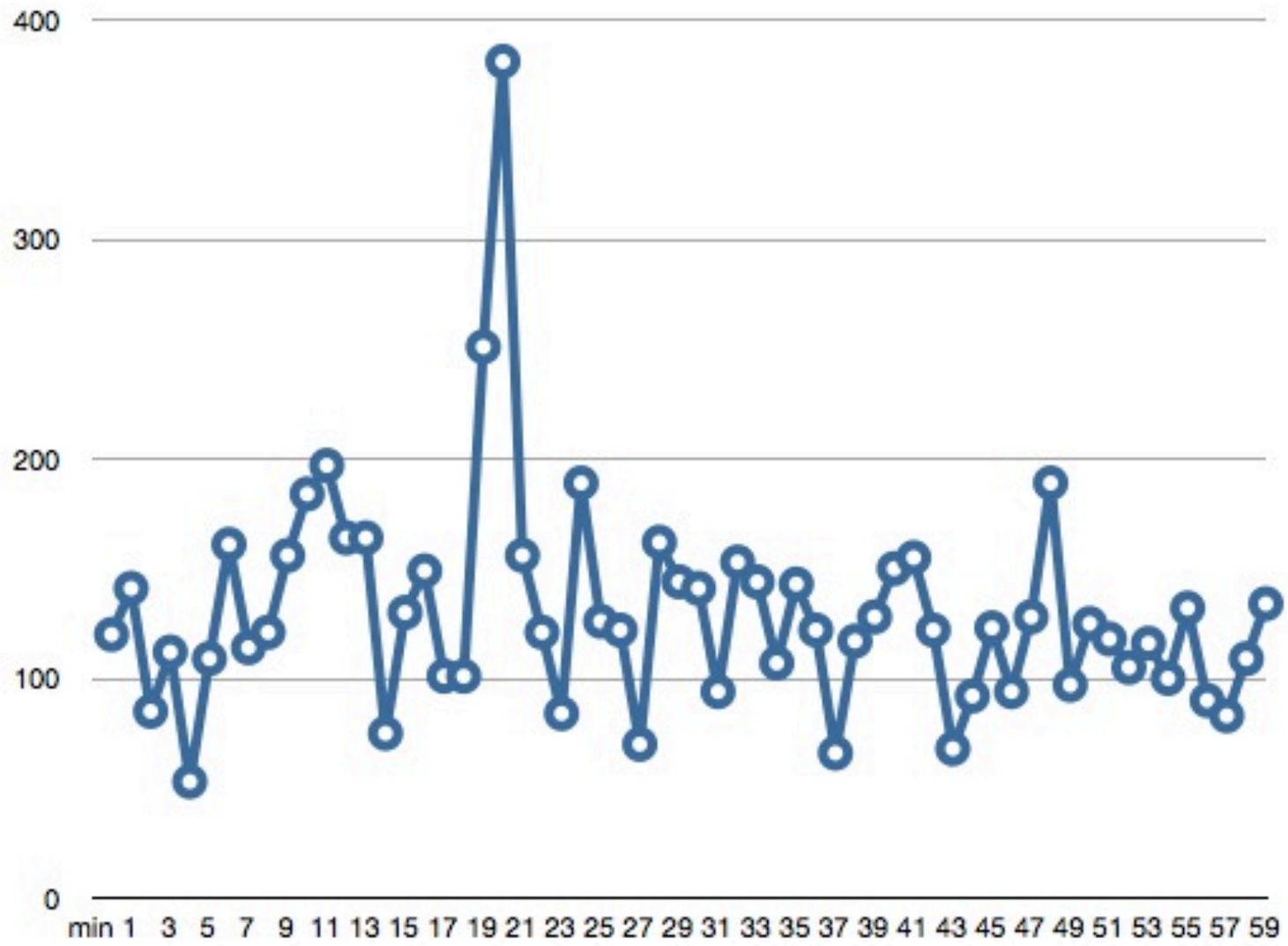
MercuryApp - Ownership Effect

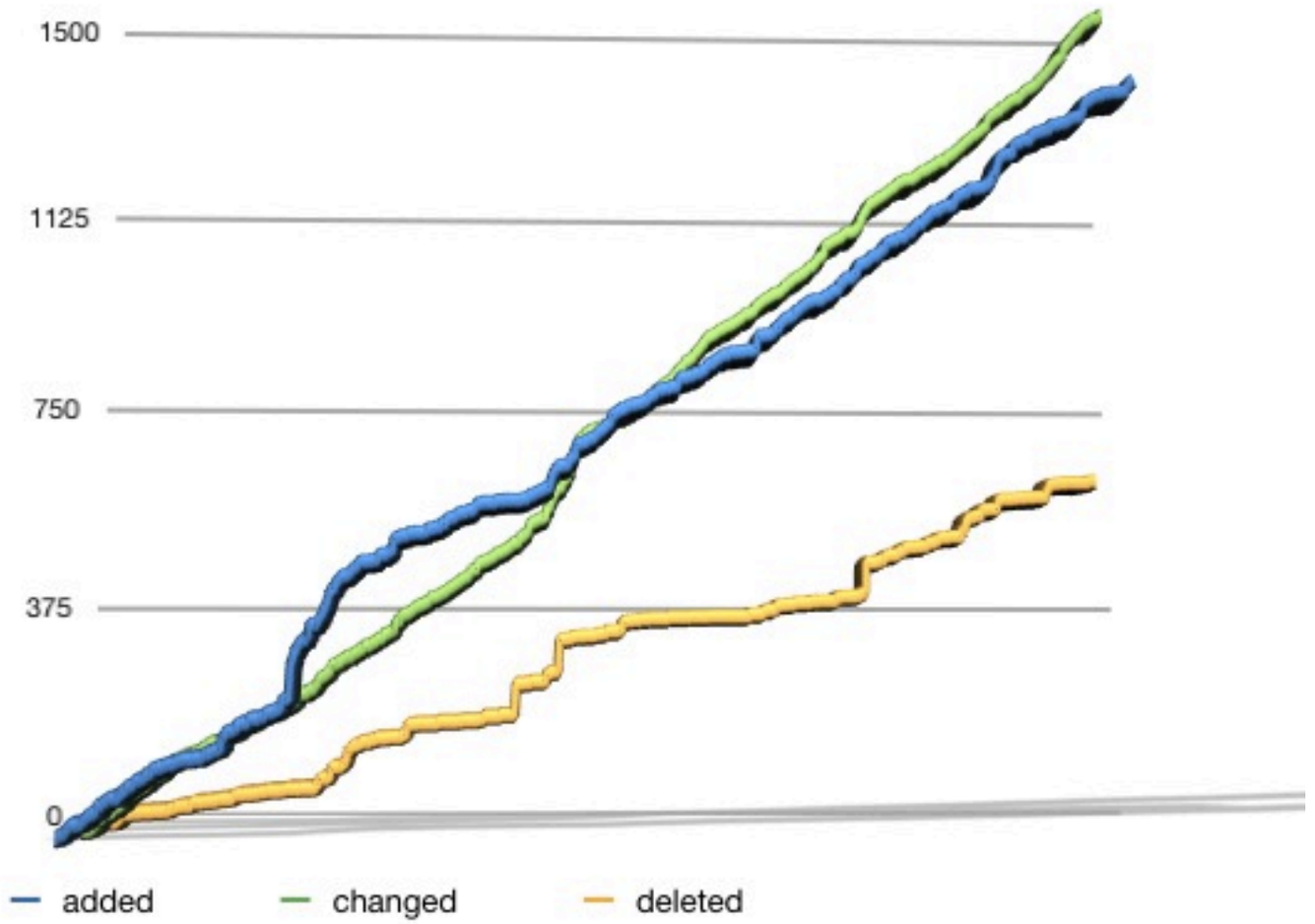


MercuryApp - Average Lines Per Commit By Month

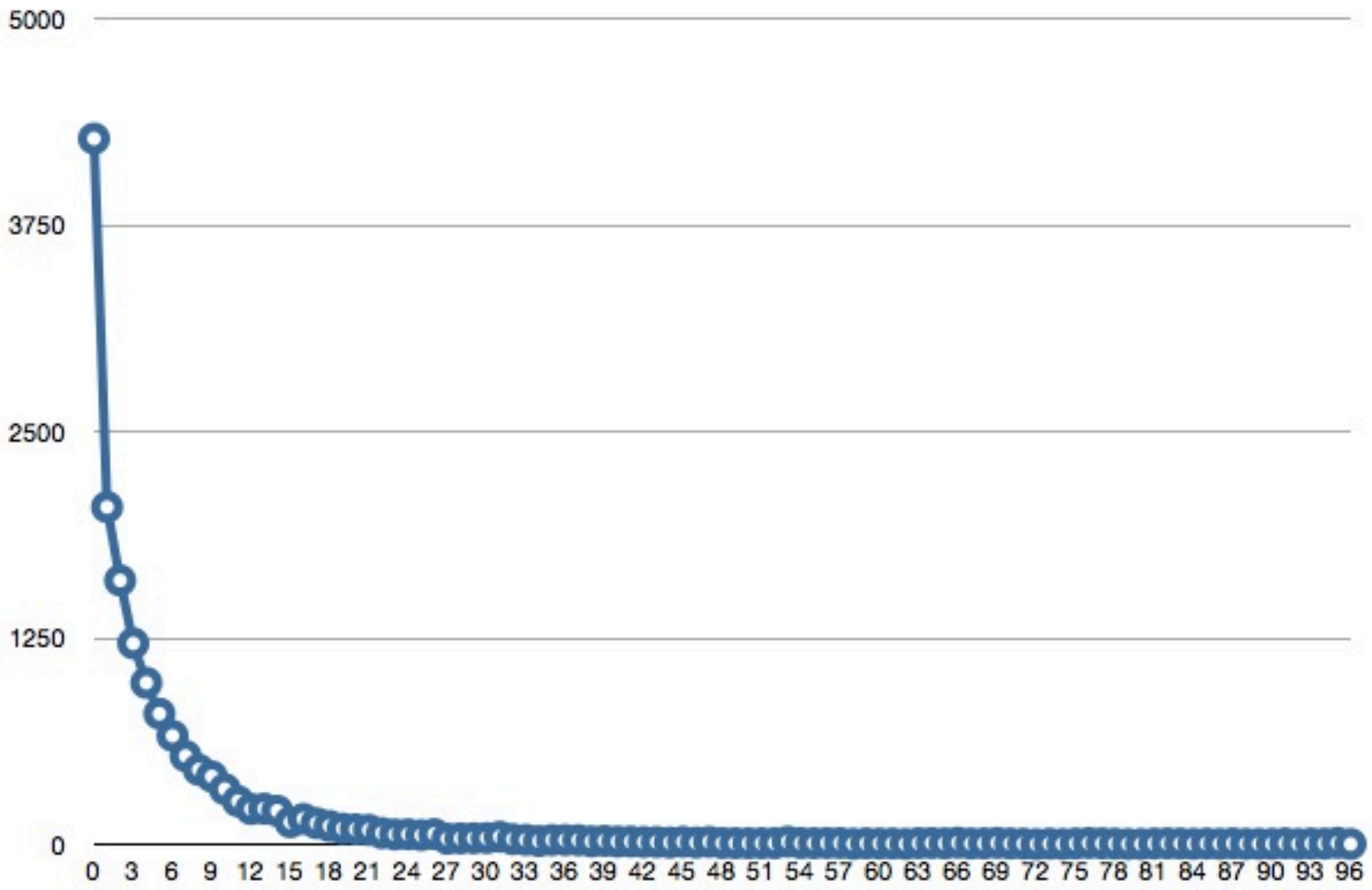


MercuryApp - Hour Profile

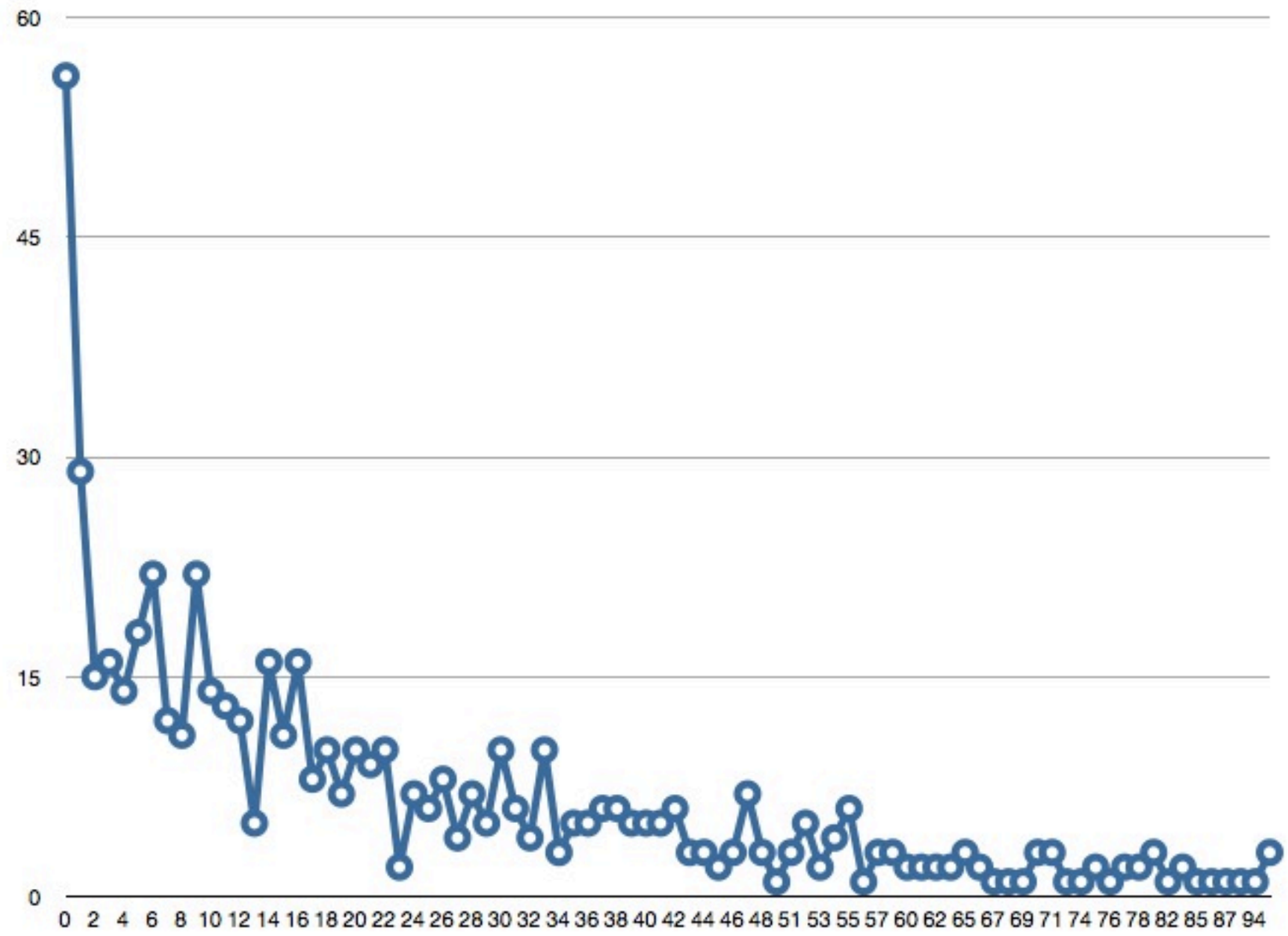



1500
1125
750
375
0
added
changed
deleted

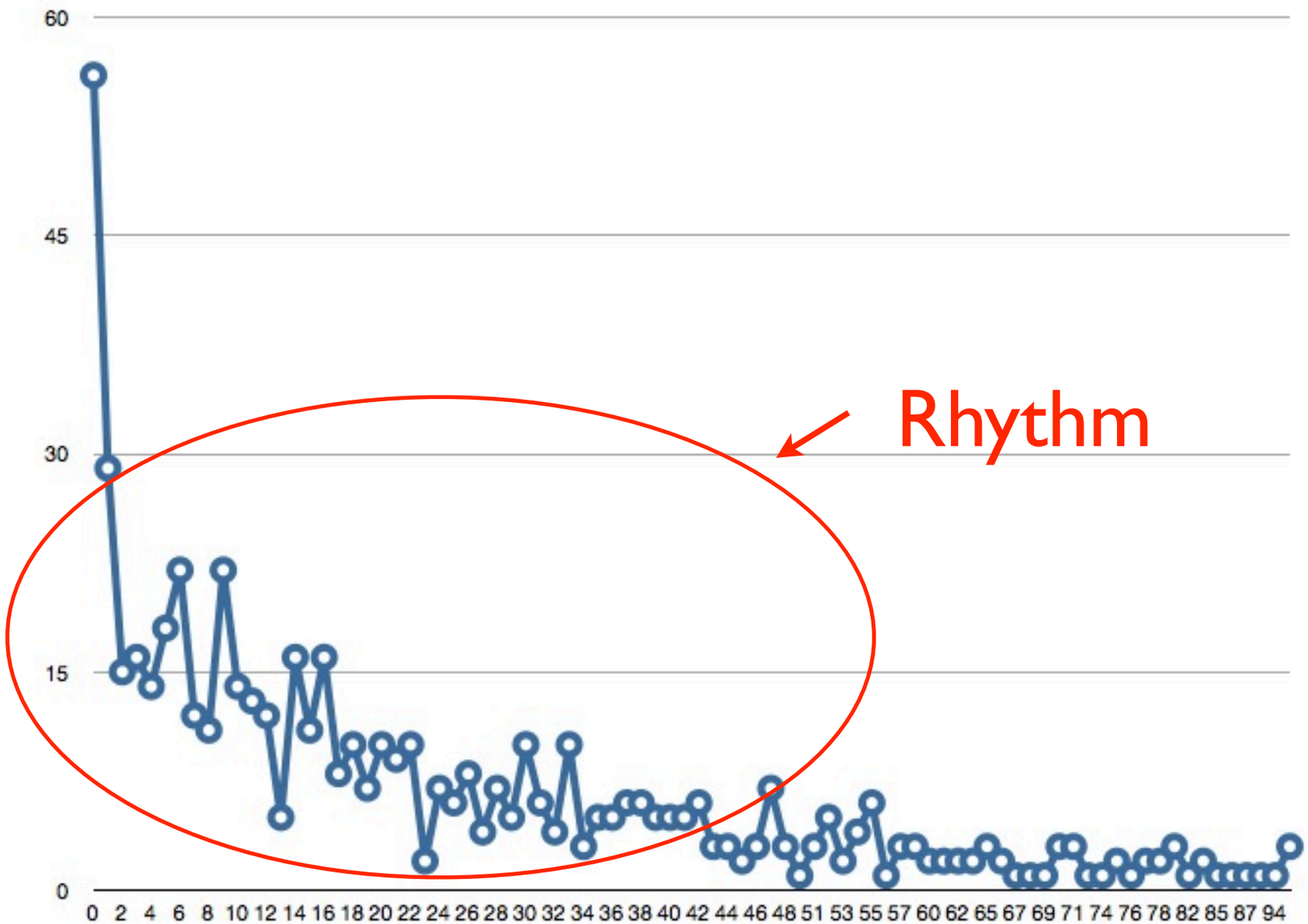
Frequency of Inter-commit Intervals



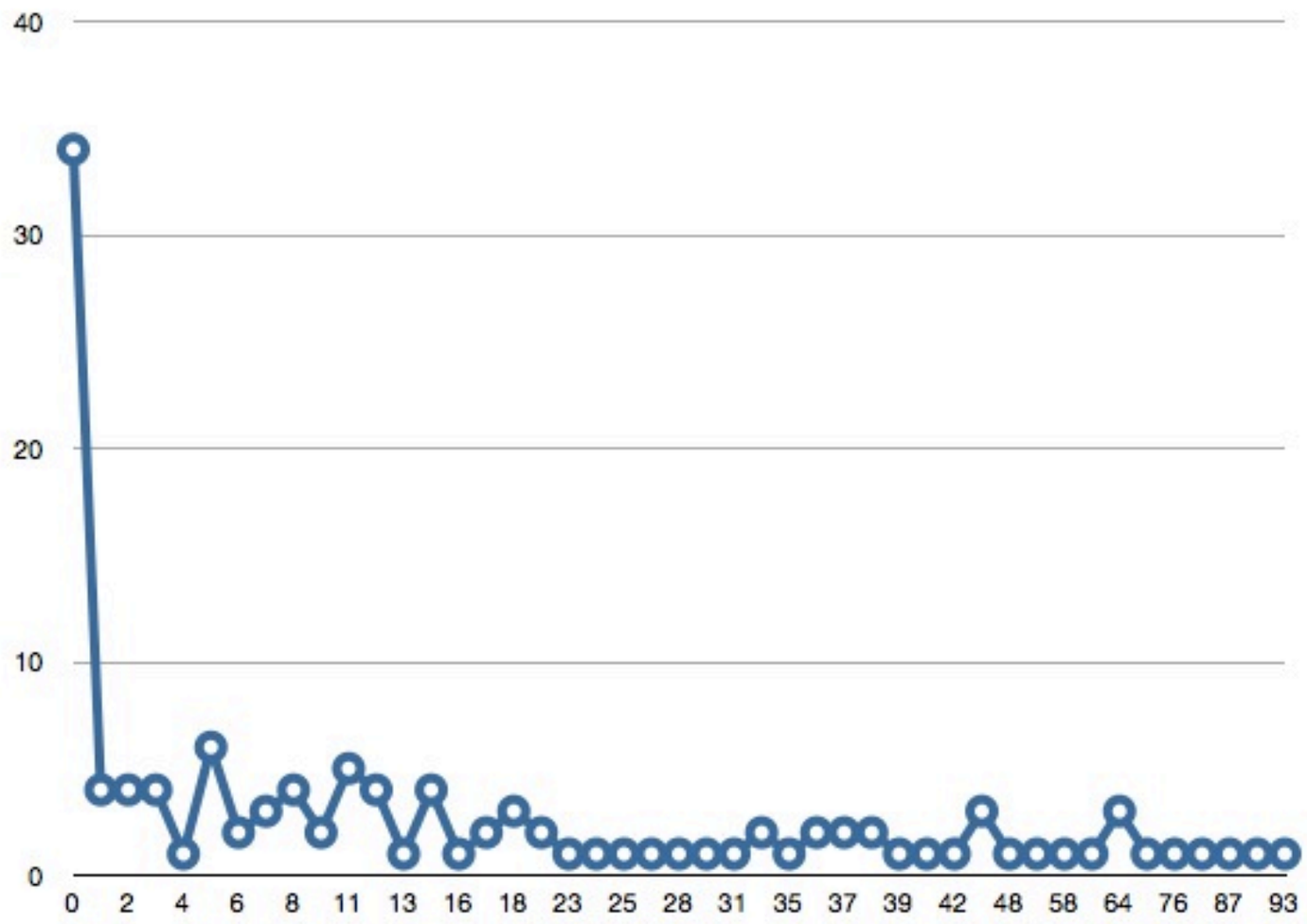
Frequency of Inter-commit Intervals



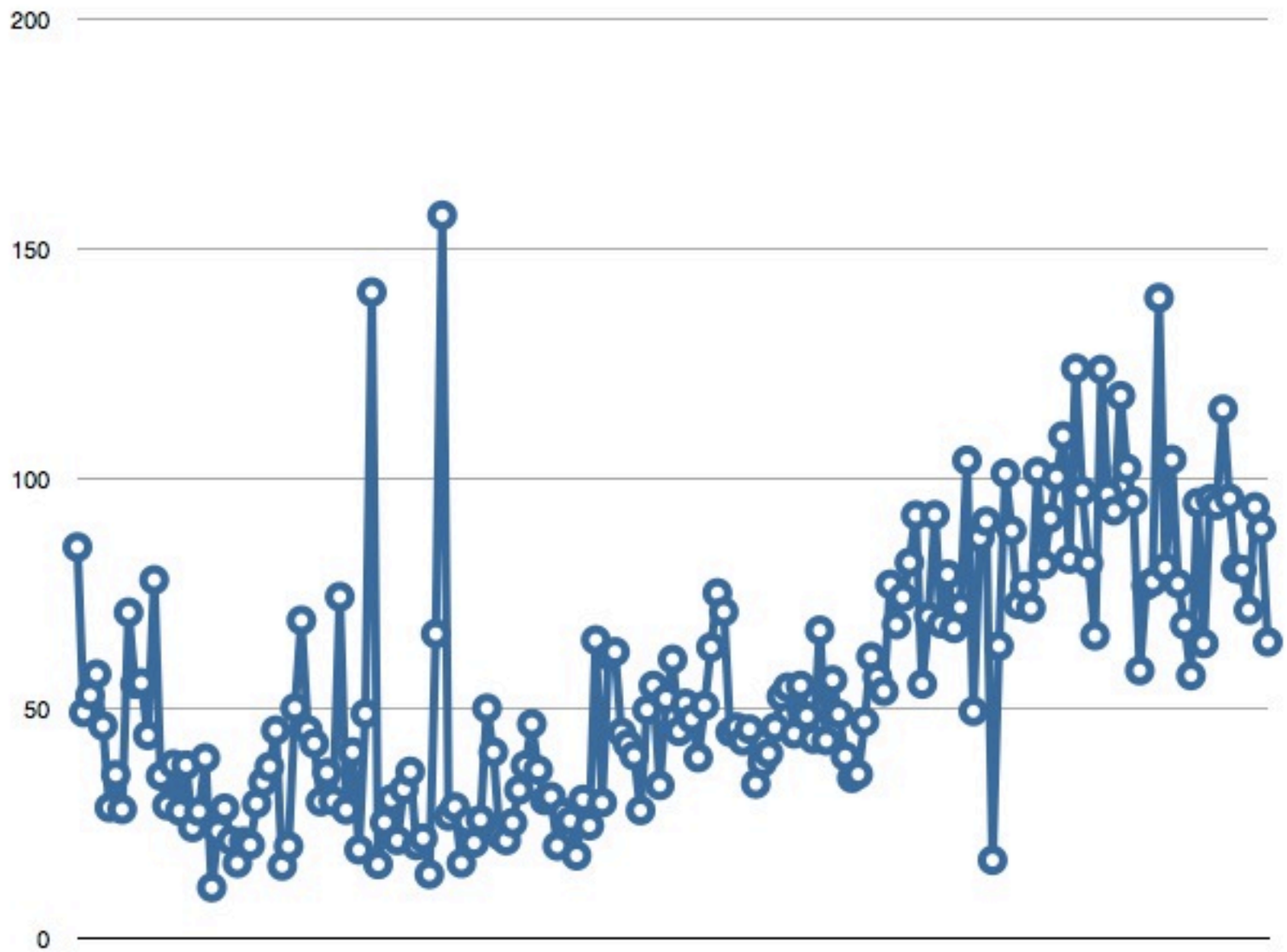
Frequency of Inter-commit Intervals



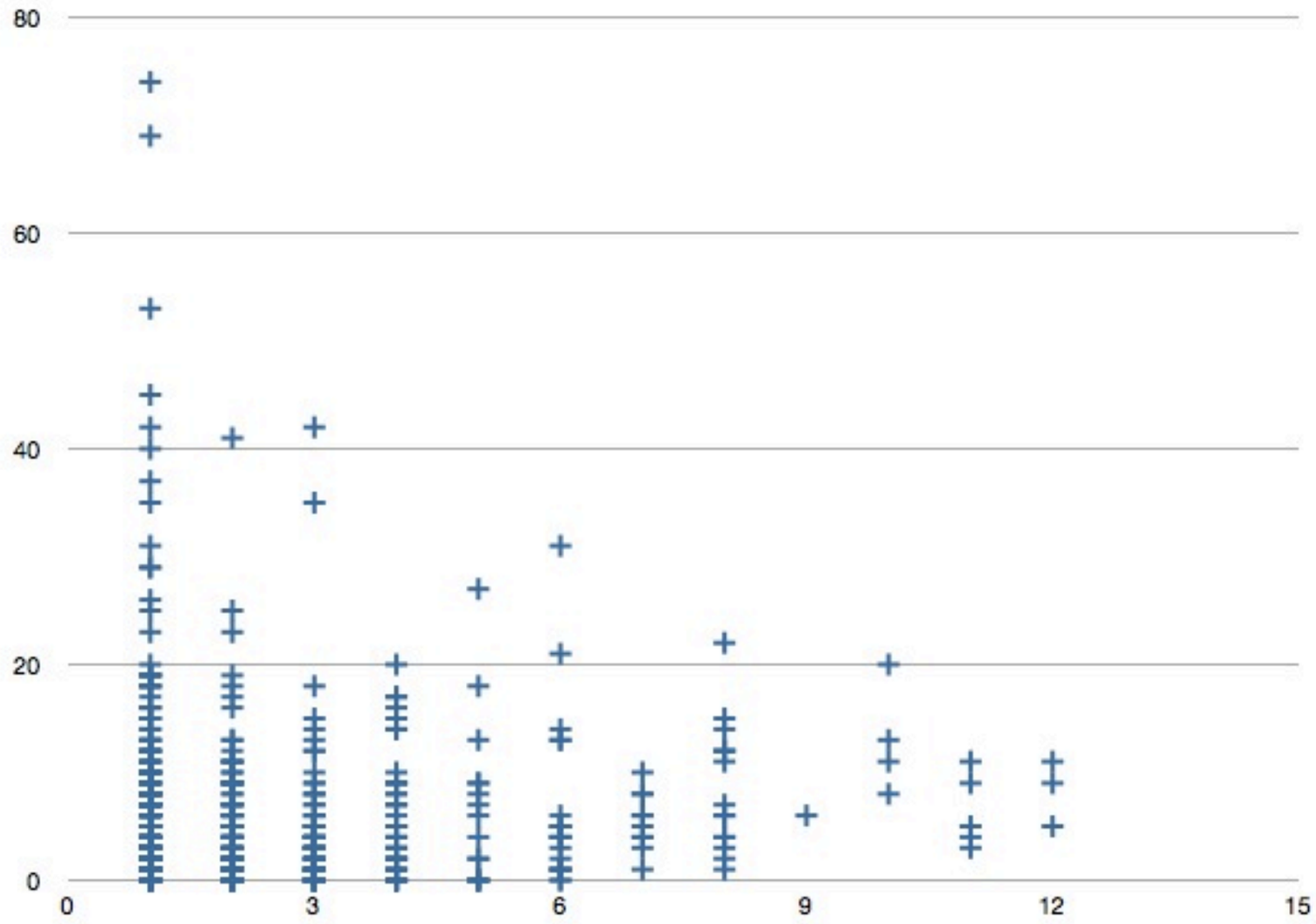
Frequency of Inter-commit Intervals



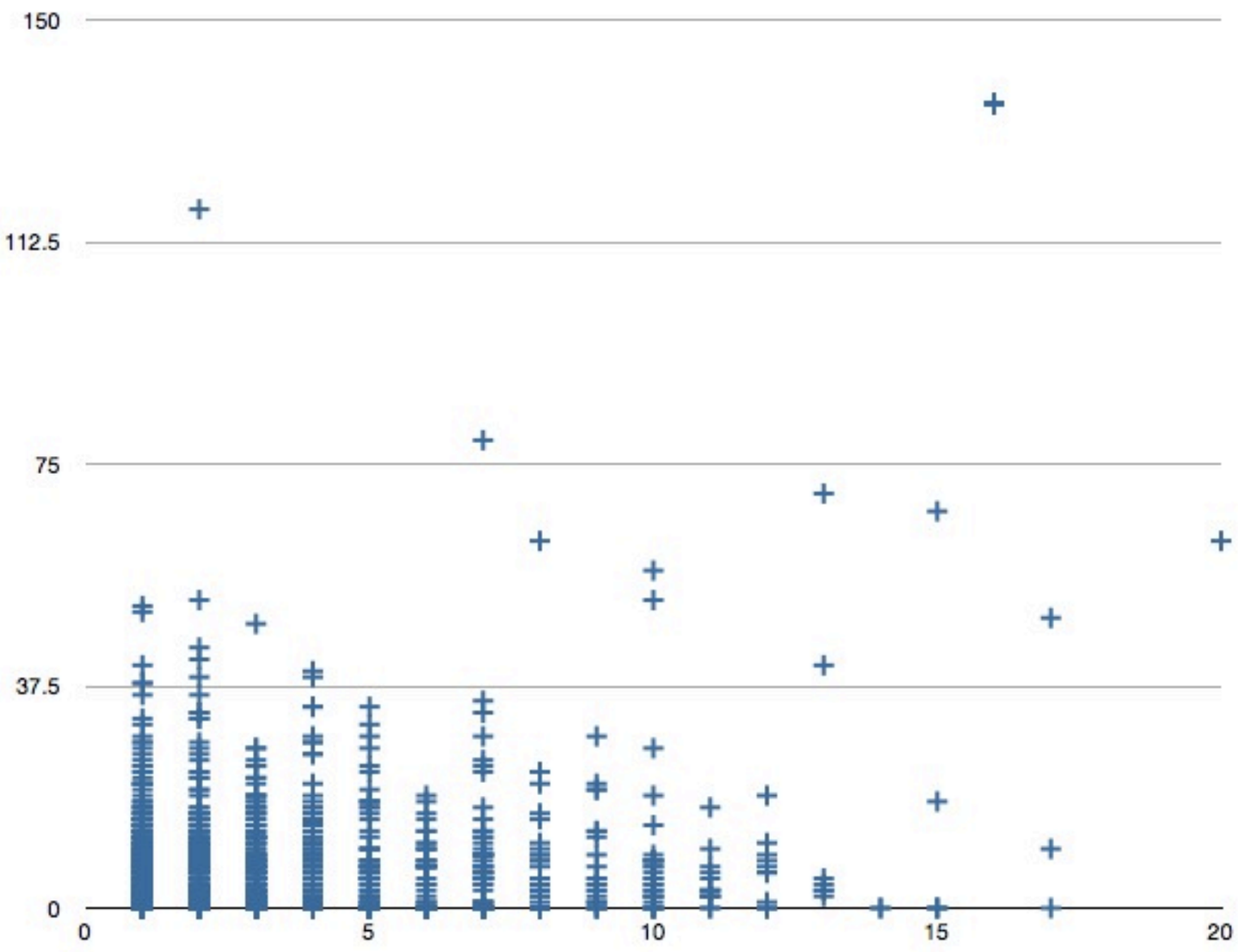
Average Lines of Code Per Commit By Week



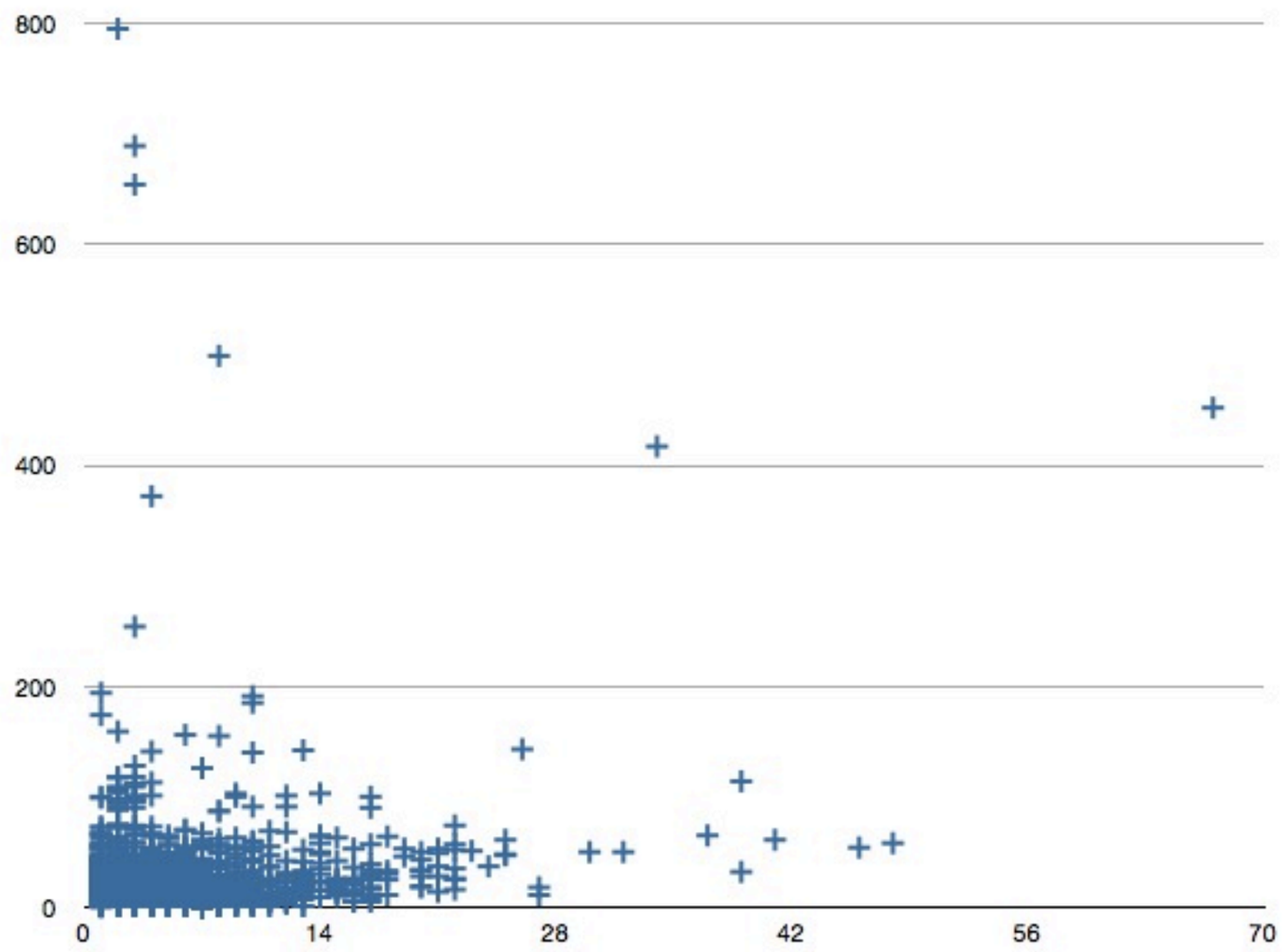
Complexity Tolerance (Developer A)



Complexity Tolerance (Developer B)



Ownership Effect (all methods)



Code Mining Issues

The Commit Problem

The Social Environment Problem

Blame

Dangerous Knowledge

Best Practice may be ‘Per Product Analysis’

Metrics

“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

“Laws” of Metrics

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

Things to Look For

Things to Look For

- Relationship between the presence of tests and refactoring

Things to Look For

- Reasons behind high churn in classes and methods (beyond the runaways)

Things to Look For

- Identification Patterns for Good Programming Episodes

Future Work

- Automated commits for full picture of development

Future Work

- Analysis of changes for developer improvement

Future Work

- Catalog of norms for good development

Future Work

- Integration with bug fix data