The Structure of Large Systems

or

"What are all those people I'm paying too much doing anyway?"

by
James Neighbors

Purpose of this Work

* Source-to-source program transformations (1976)
  - transformations at wrong level of abstraction
  - procedural transformations a BIG problem
* Draco: software construction using components (1984)
  - domains
  - refinements
  - optimizations (previously transformations)
* Problems with Draco
  - what constitutes a domain?
  - complete refinement every time is unrealistic
* How do people cope with this in large systems?
  - classical automation analysis
  - not a statistical study

Viewpoint of the Work

* "Software Crisis" has not gone away

* Crisis is primarily in large systems not small systems
  - 60,000 lines not a problem
    * 4 – 5 developers
    * 2 – 3 maintainers
    * available technology works
    * hiring people may be a problem
  - 200,000 lines is a problem
    * 10 – 20 developers and maintainers
    * available technology is strained

* Building large systems is a social process

* Granularity of current reuse is too small
  - queues
  - sorts
  - list processing

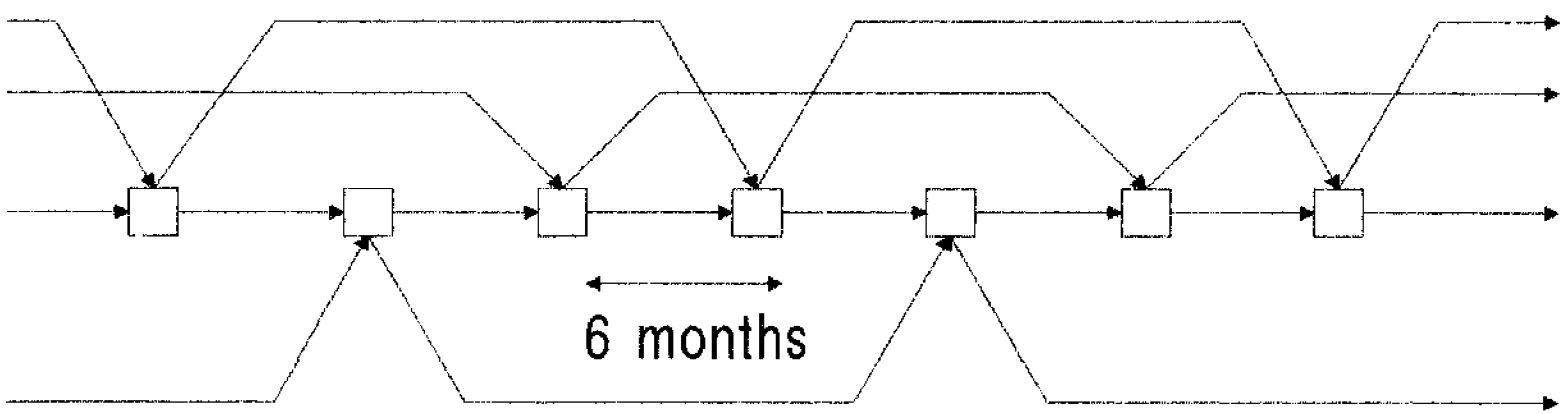
* Find out what granularity is used in current large systems

Method of Data Collection

* Systems studied:
  - CAD/CAM System, 7 years (FORTRAN 800,000 lines)
  - PBX/Digital Network, 3 years (PASCAL 600,000 lines)
  - four others similar size, less detail
  - no assembly systems
* Efficient automated tools are a requirement
  - parser generators (source code, linkage files, MAKE files)
  - file management system for large files
  - report generators
  - diagram generators
  - migrating to PCs from mainframes
* Why is this work tolerated in the organization?
  - Quality Control information is a side-effect
  - coding standards
  - stronger type checking
  - inter/intra module flow analysis
  - reverse engineering
    * tightly coupled modules (MIL analysis)
    * architectural design diagrams

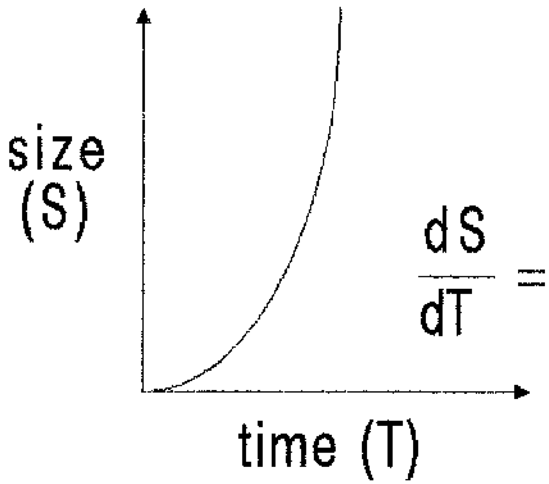
Life in the Big System

* big systems are 200,000 lines and up
* ongoing evolution: complete rebuild and restructing with major changes in function NOT "we bid it, we built it, it's over"
* about one programmer per 10,000 – 20,000 lines of source code
* 500,000 line system has a monthly burn rate of about $250,000.00 per month
* naming problems, lost code even with SCC, aliased data, common data areas
* at least "3 – fork" development
  - new release required every 6 months
  - 18 months required to make a substantial change

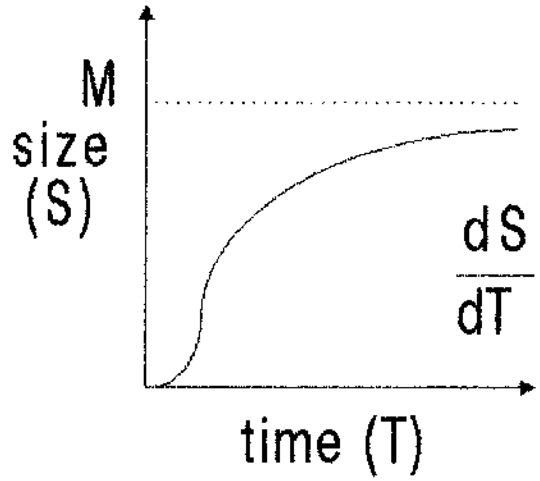


Logistic Growth w/Waste Accumulation

following *A Model for Growth and Decay of Biological and Social Systems*
Rein Taagepera, **The Study of Man**, Vol 1, 1972



$$\frac{dS}{dT} = kS$$



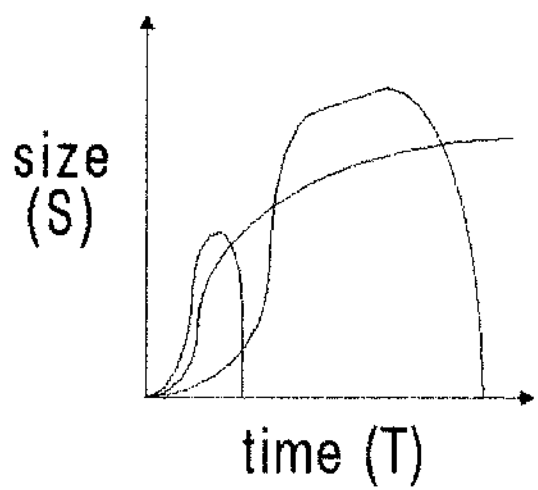
$$\frac{dS}{dT} = k(M-S)S$$

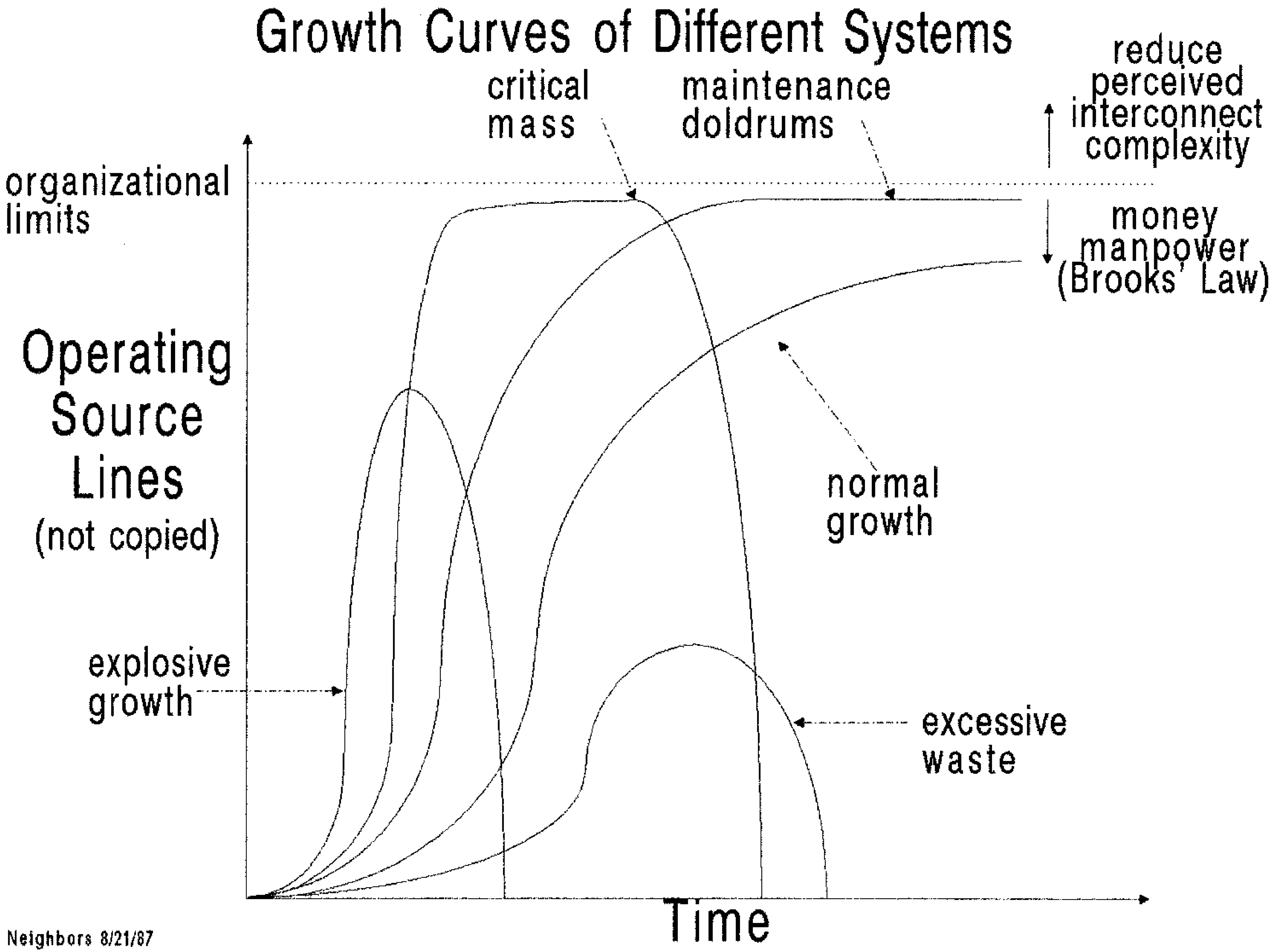
$$\frac{dS}{dT} = k(M-S)(S-h''W) \quad \text{where} \quad \frac{dW}{dT} = h'S \quad \text{thus} \quad W = h'\int S\,dT$$

h' = waste accumulation rate
h'' = waste virulence

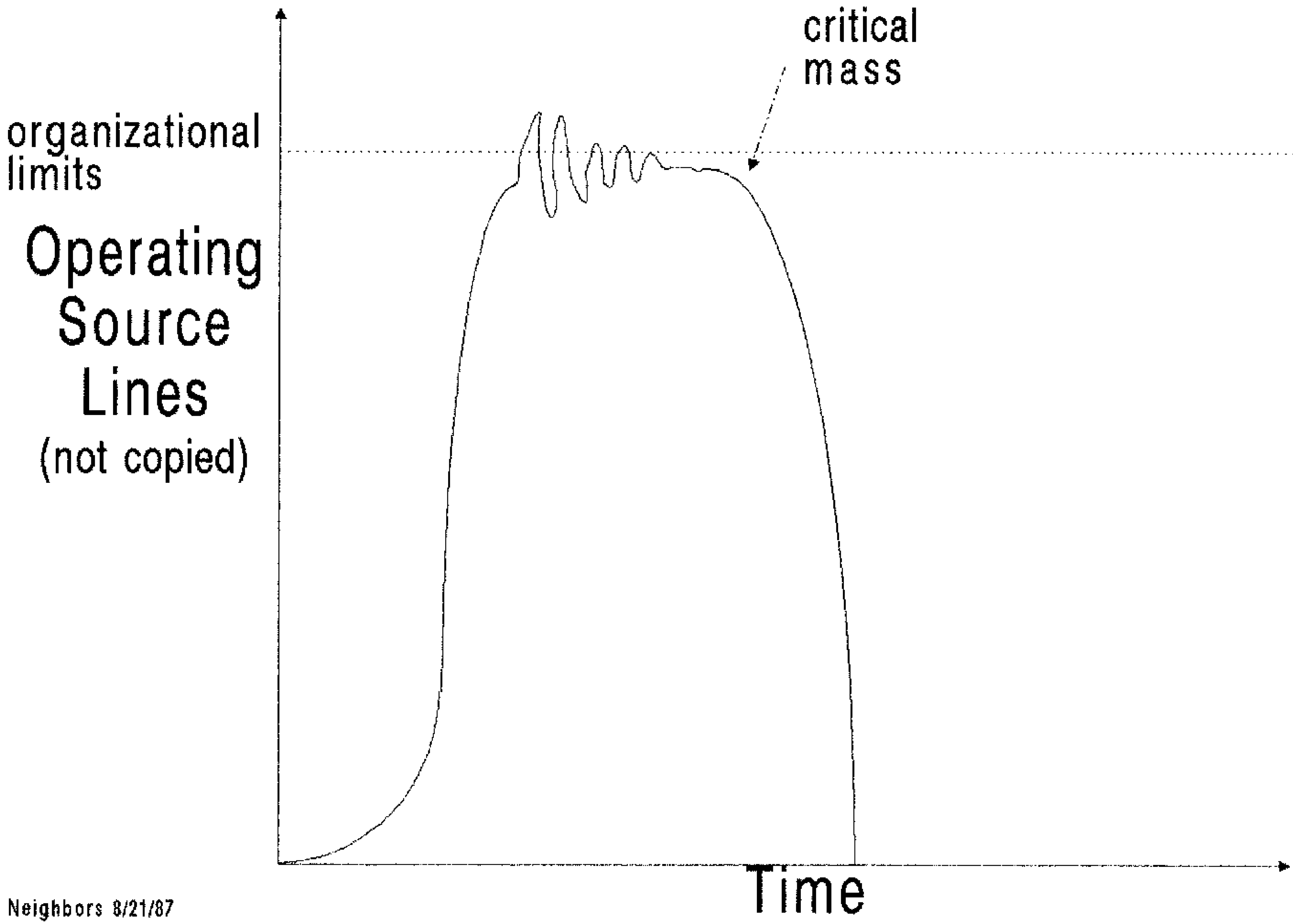
$$\frac{dS}{dT} = k(M-S)\left(S-h\int S\,dT\right)$$

where $h = h'h''$

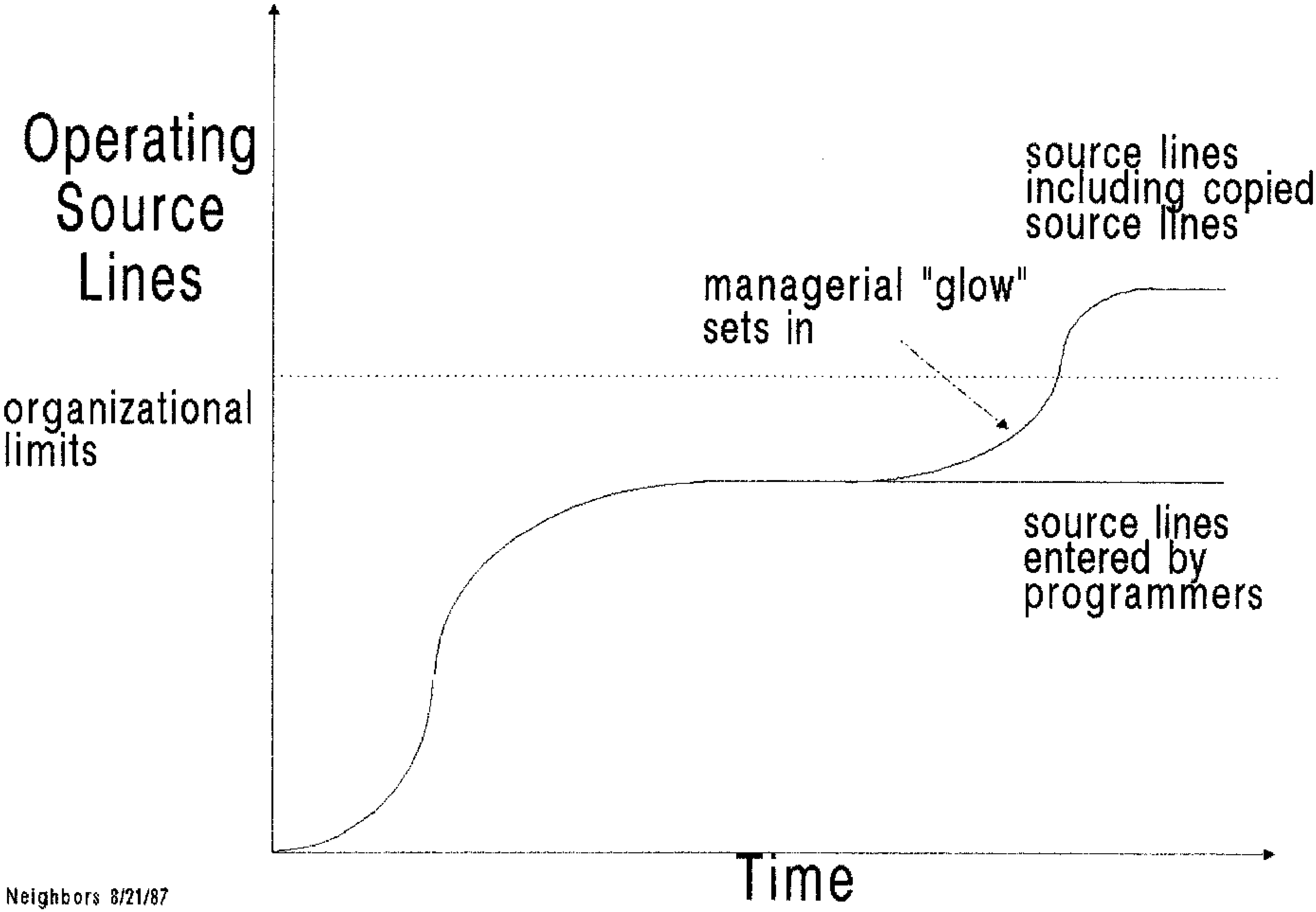



Growth Curves of Different Systems
critical mass
maintenance doldrums
reduce perceived interconnect complexity
organizational limits
money manpower (Brooks' Law)
Operating Source Lines (not copied)
normal growth
explosive growth
excessive waste
Time
Neighbors 8/21/87

Oscillations as Development Limits Out

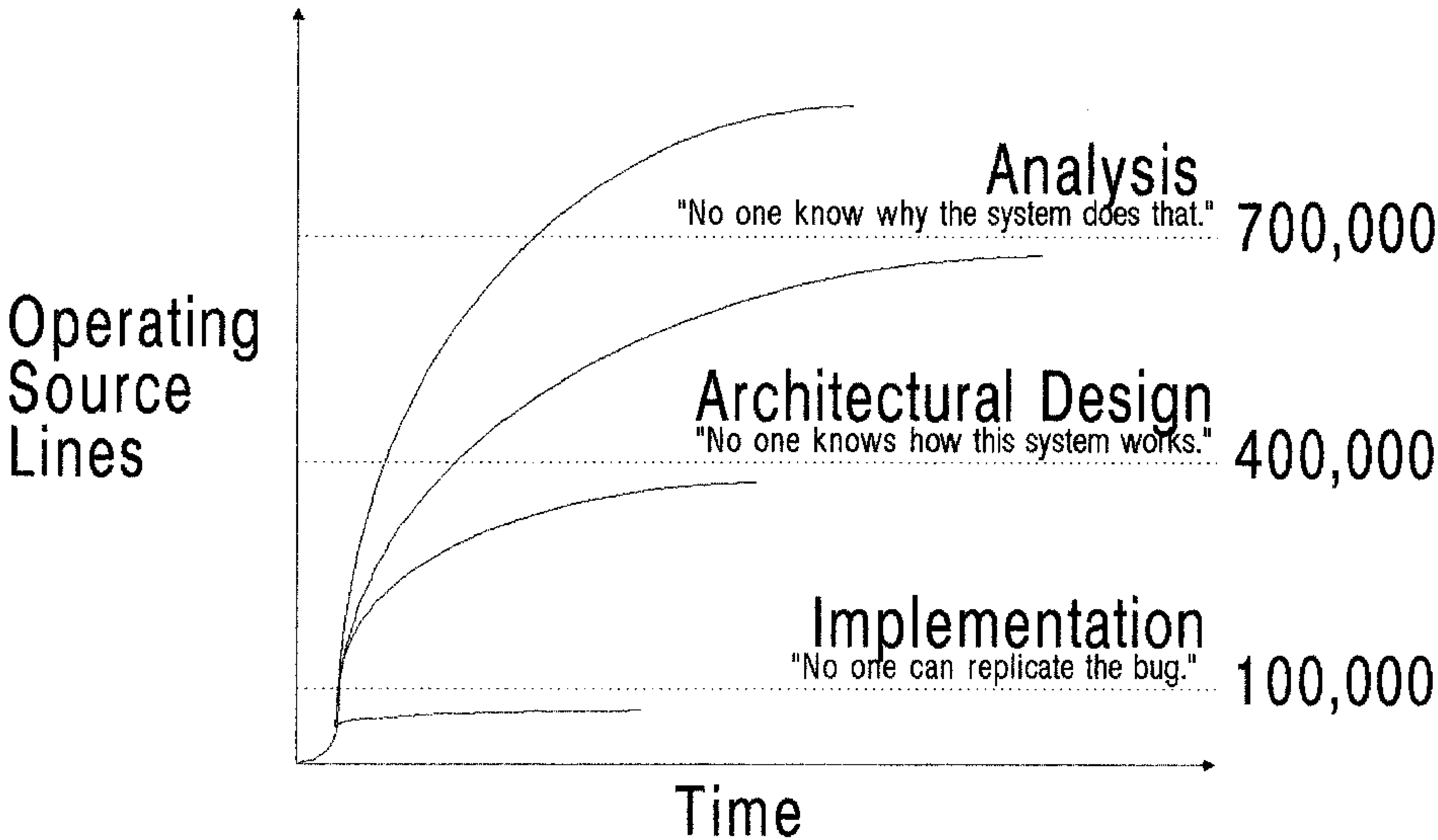


Code "Reuse" During Maintenance Doldrums

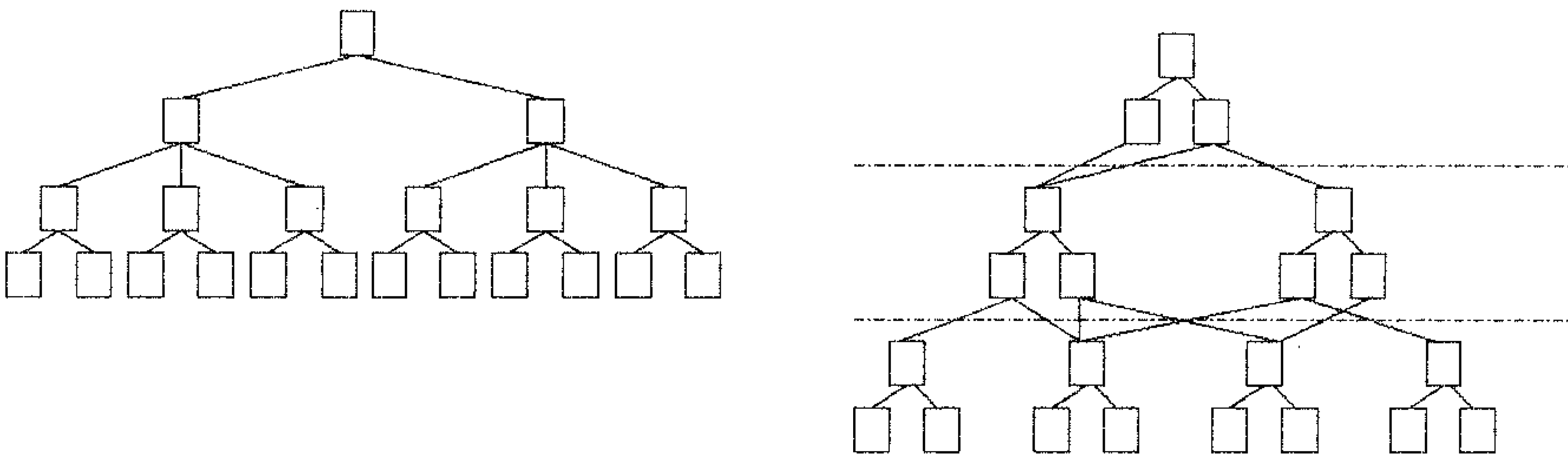


Neighbors 8/21/87

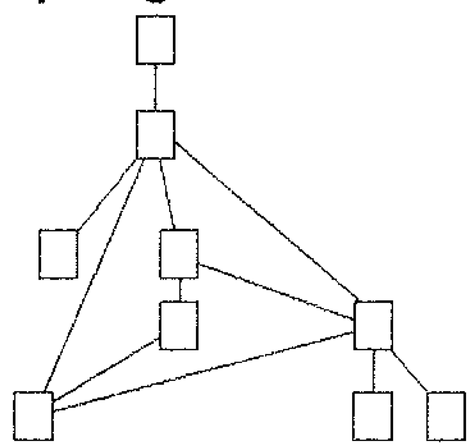
Sizes and Phases where Problems Occur



Architectural Design Structures
Functional Decomposition vs. Layers of Abstraction

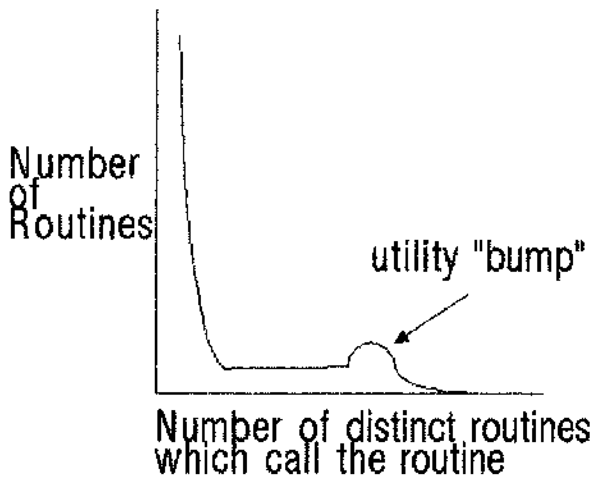


and, of course, real programs exhibit both structures ...

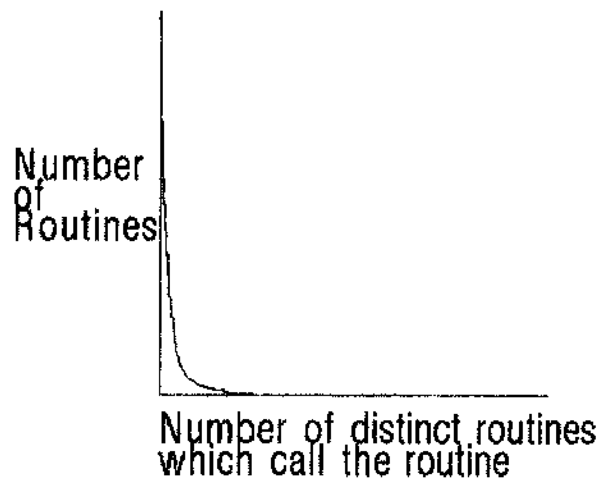


what do large systems do?

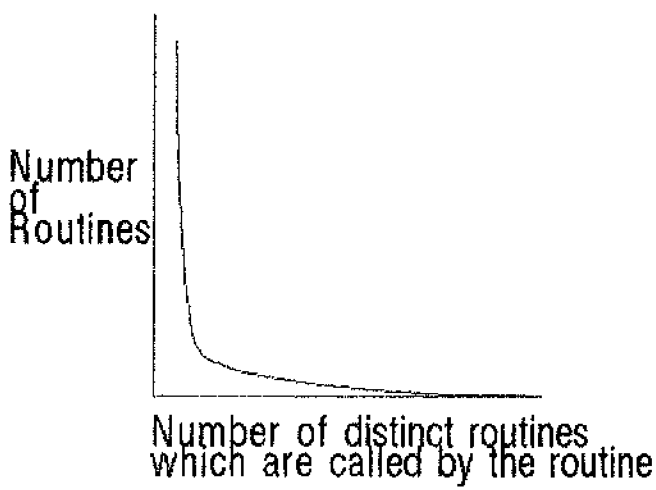
Control Flow Branching



small programs



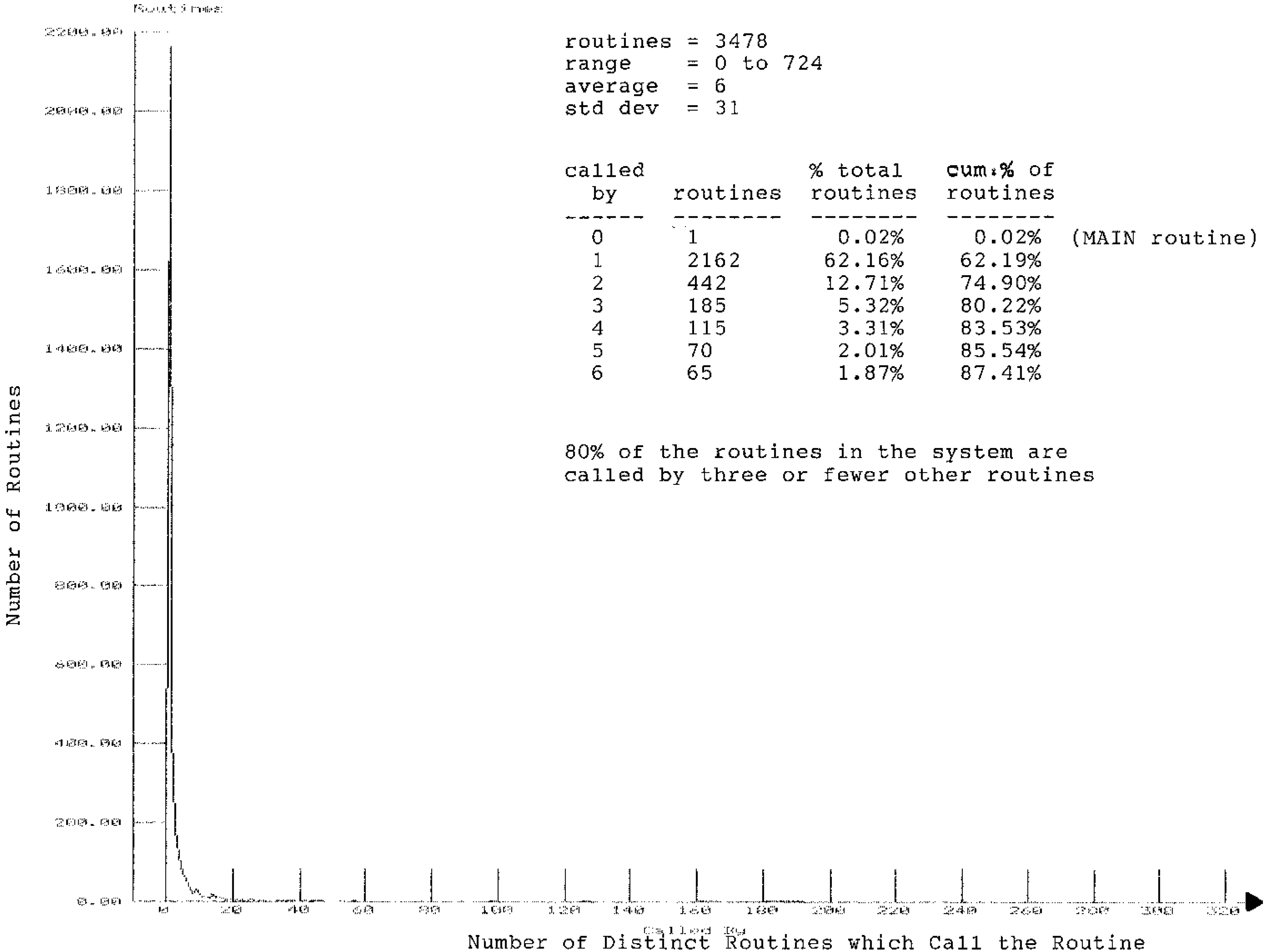
large systems



both programs and systems

NOTE: routine size mean is 220 source lines, range is 10 to 500 source lines

Routines Which are Called By N Other Routines

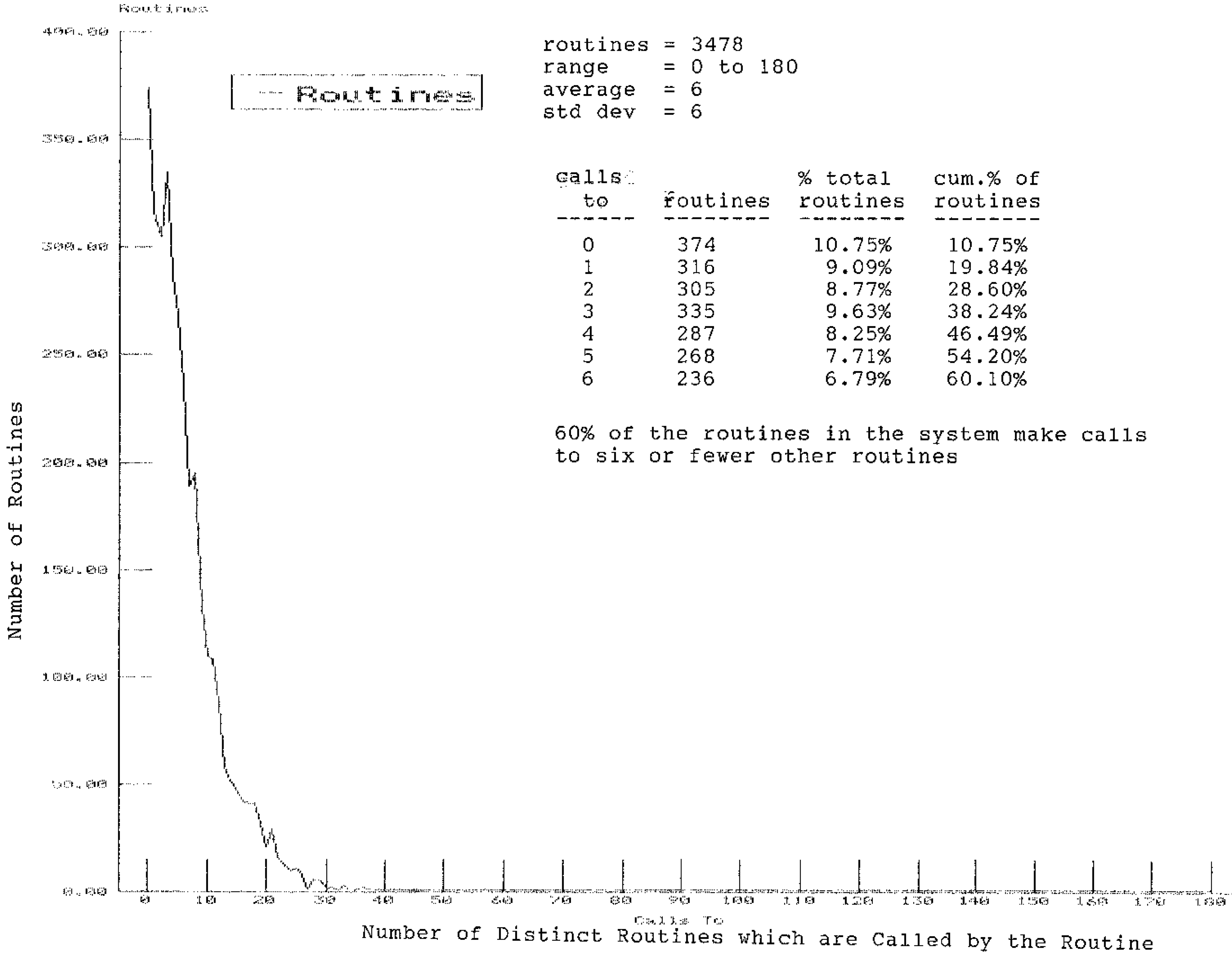


routines = 3478
range = 0 to 724
average = 6
std dev = 31

called by	routines	% total routines	cum.% of routines	
0	1	0.02%	0.02%	(MAIN routine)
1	2162	62.16%	62.19%	
2	442	12.71%	74.90%	
3	185	5.32%	80.22%	
4	115	3.31%	83.53%	
5	70	2.01%	85.54%	
6	65	1.87%	87.41%	

80% of the routines in the system are called by three or fewer other routines

Number of Routines which Make N Calls To Other Routines

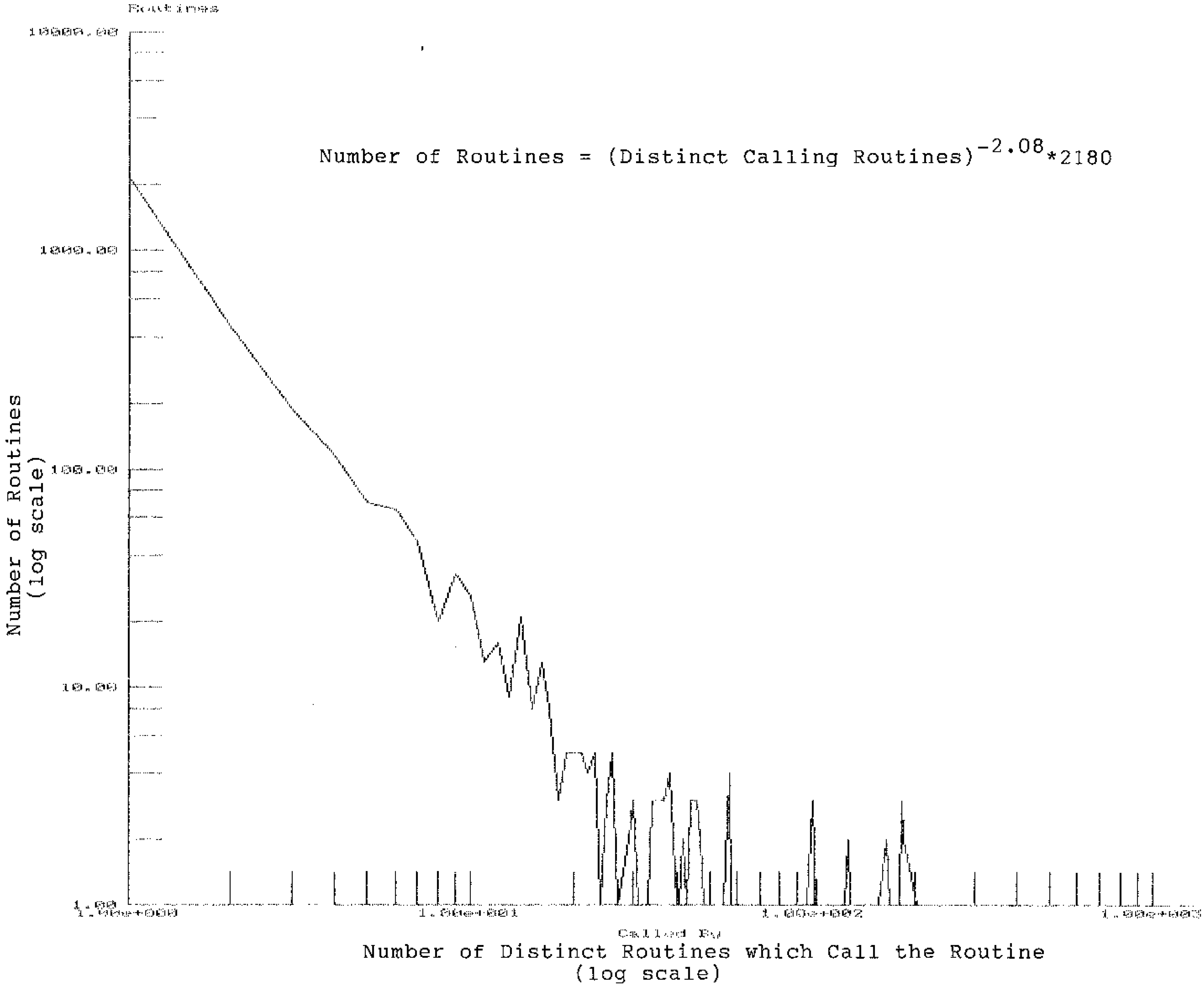



```
routines = 3478
range    = 0 to 180
average  = 6
std dev  = 6
```

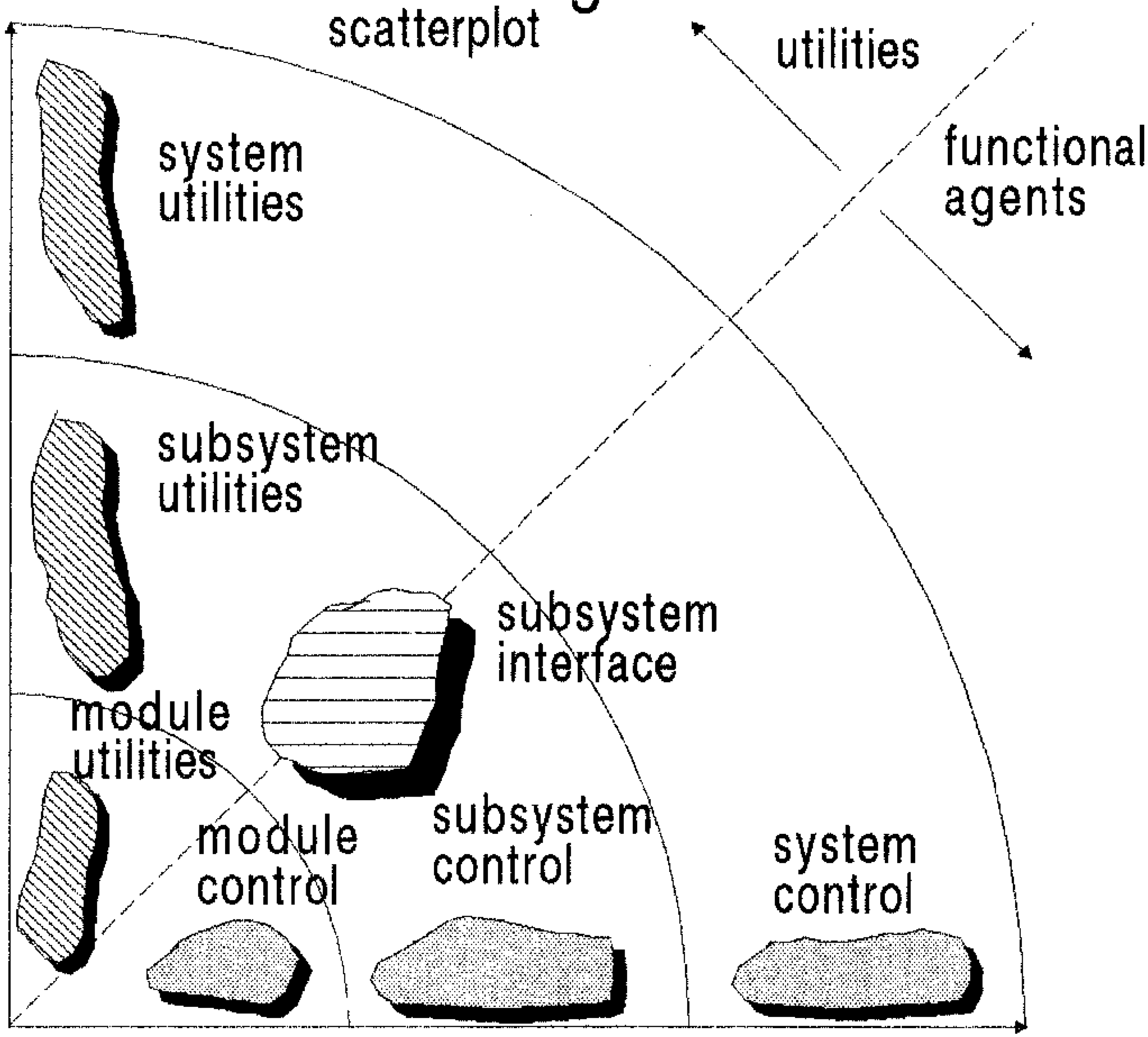
calls to	routines	% total routines	cum.% of routines
0	374	10.75%	10.75%
1	316	9.09%	19.84%
2	305	8.77%	28.60%
3	335	9.63%	38.24%
4	287	8.25%	46.49%
5	268	7.71%	54.20%
6	236	6.79%	60.10%

60% of the routines in the system make calls to six or fewer other routines

Routines Which are Called By N Other Routines



Routine Branching Ratios



Number of Distinct Routines which Call the Routine

Number of Distinct Routines which are Called by the Routine

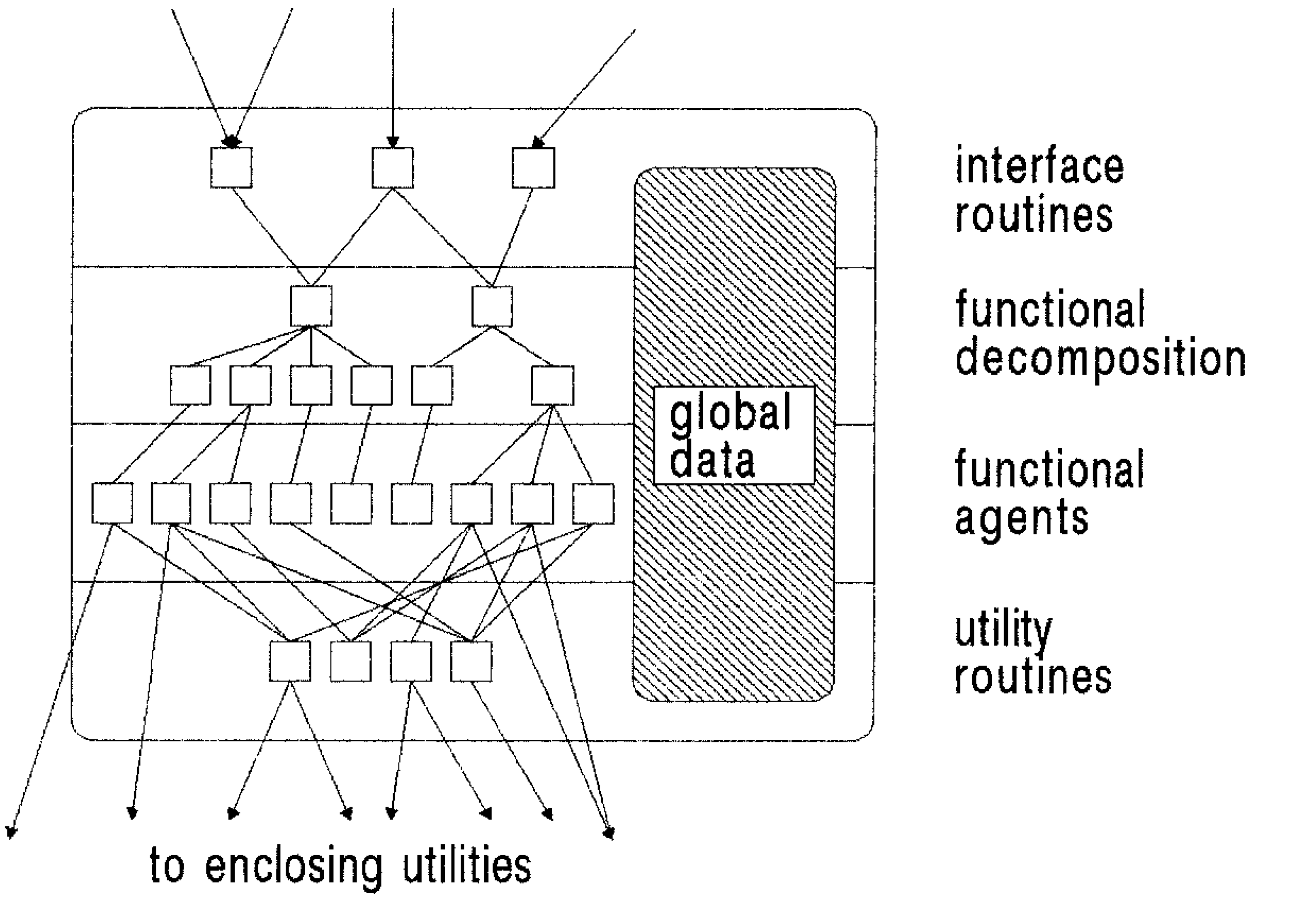
Note: don't take this too seriously, the branch ratio by itself is not strong enough to characterize a routine.

Neighbors 8/21/87

Subsystems

* subsystems are found by applying a policy to each routine:
  - branch ratio w/respect to rest of system
  - branch ratio w/respect to subsystem under construction
  - shared global data
* subsystems have a regular structure similar to systems
* subsystems contain 20–150 routines or 4,000–30,000 source lines
* subsystems are embedding and recursive
* subsystems are the key to reverse engineering and regaining control of a large system

System and Subsystem Structure



Neighbors 8/21/87

Conclusions

* ganularity of large systems are subsystems of 20 – 150 routines or 4,000 – 30,000 source lines

* size of subsystem is approximately the same number of source lines assigned per programmer only the mapping is usually not one programmer to one subsystem

* "discovery" of components in existing systems
  - very little chance w/no refinement history
  - Balzer's "information spreading"
  - subsystems are domain artifacts which have not spread or embedded

Draco Implications

* domain concept justified

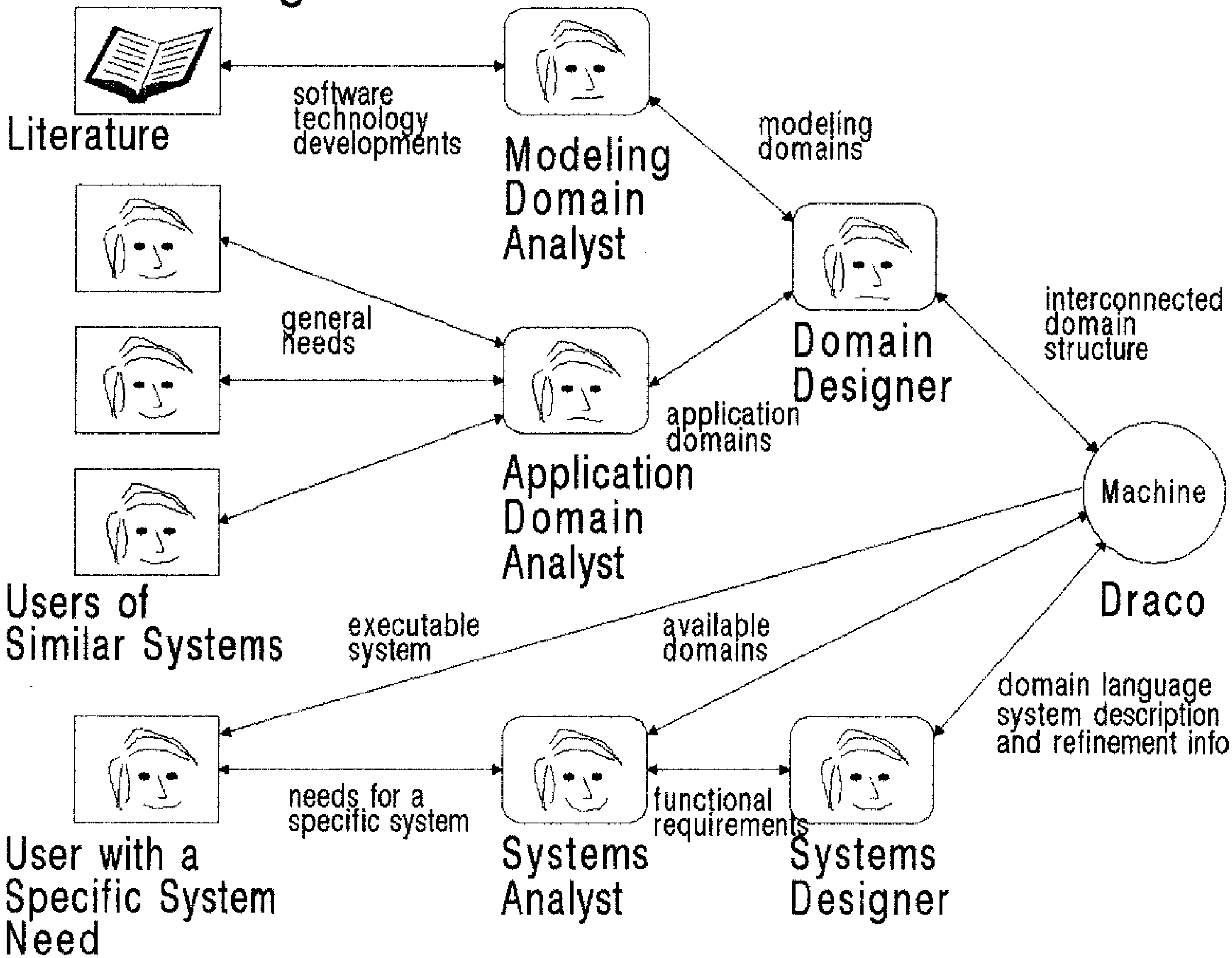
* a subsystem resulting from a Draco refinement of a problem should imply the use of a particular modeling domain. but the converse is not true

* partially refined subsystems should be the library unit managed by Draco – no more complete refinements

* varying architectural design a good idea

* bottom end of the domain hierarchy is not as important as I once thought – modeling domains are much more important

Organizational Context of Draco



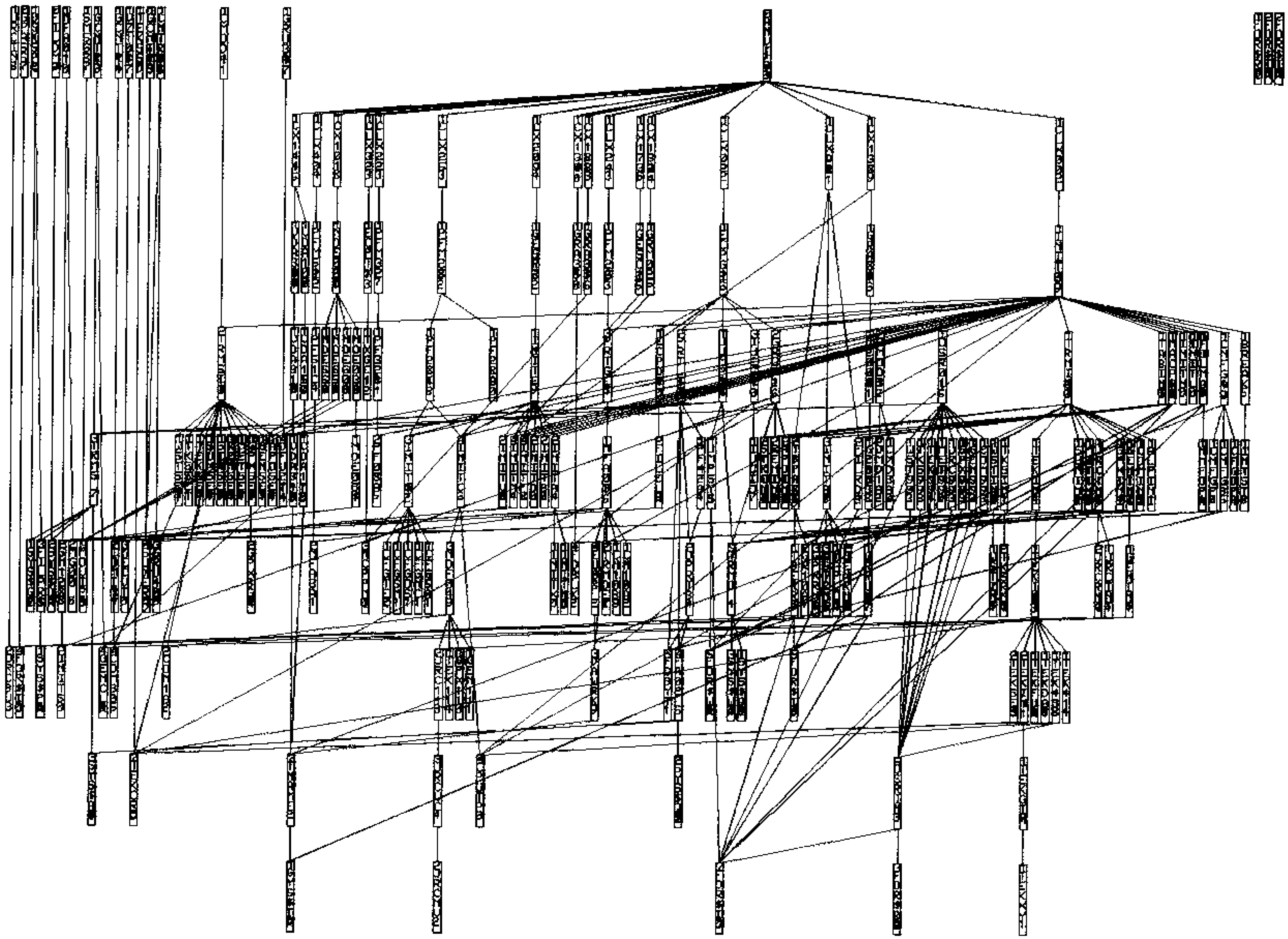
Future

* more work on subsystem extraction and restructuring

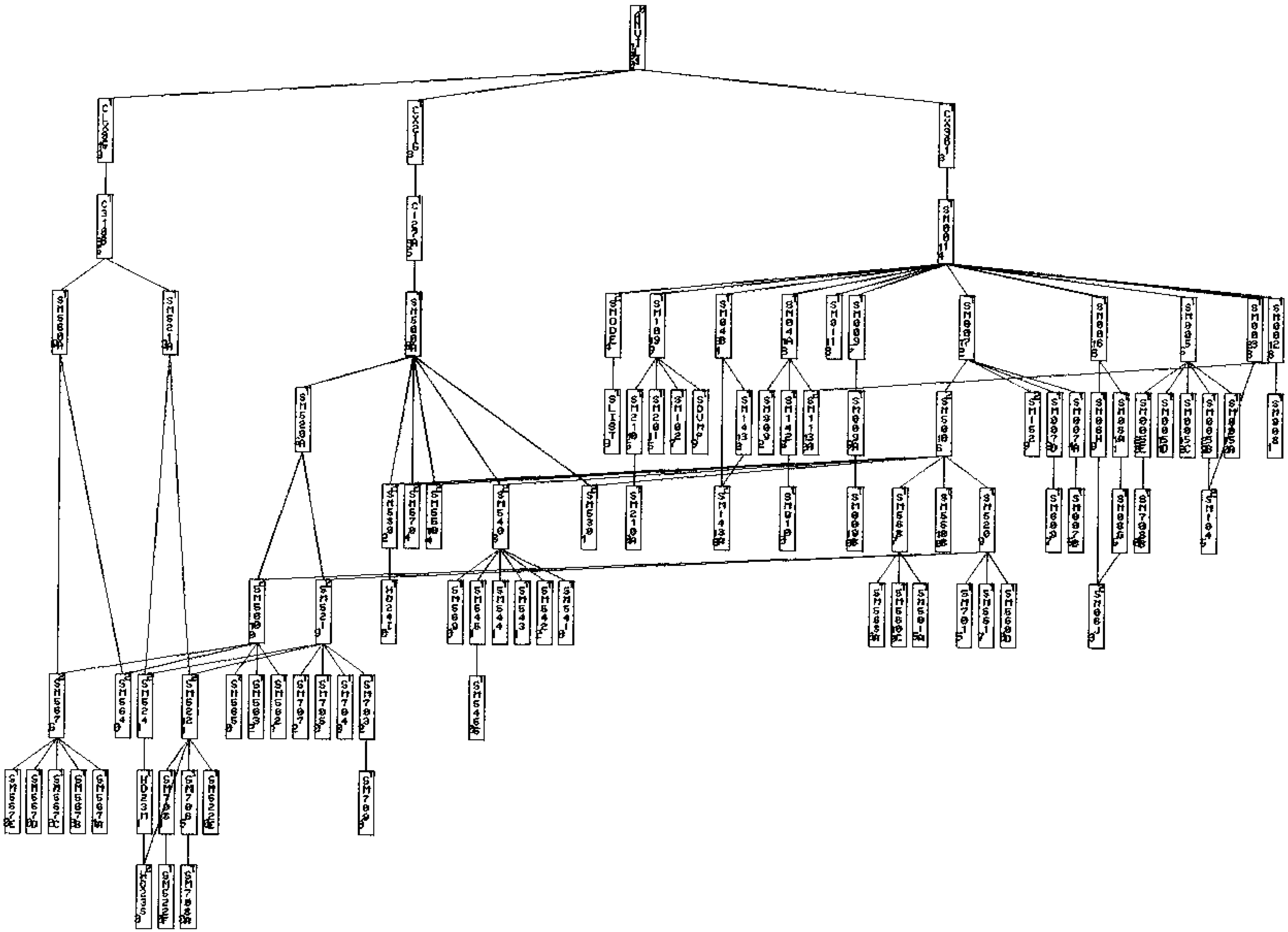
* "scan it yourself" kit – a nice application domain

* Draco rebuild in Ada targeting Ada
  system architecture in Biggerstaff's reusability book

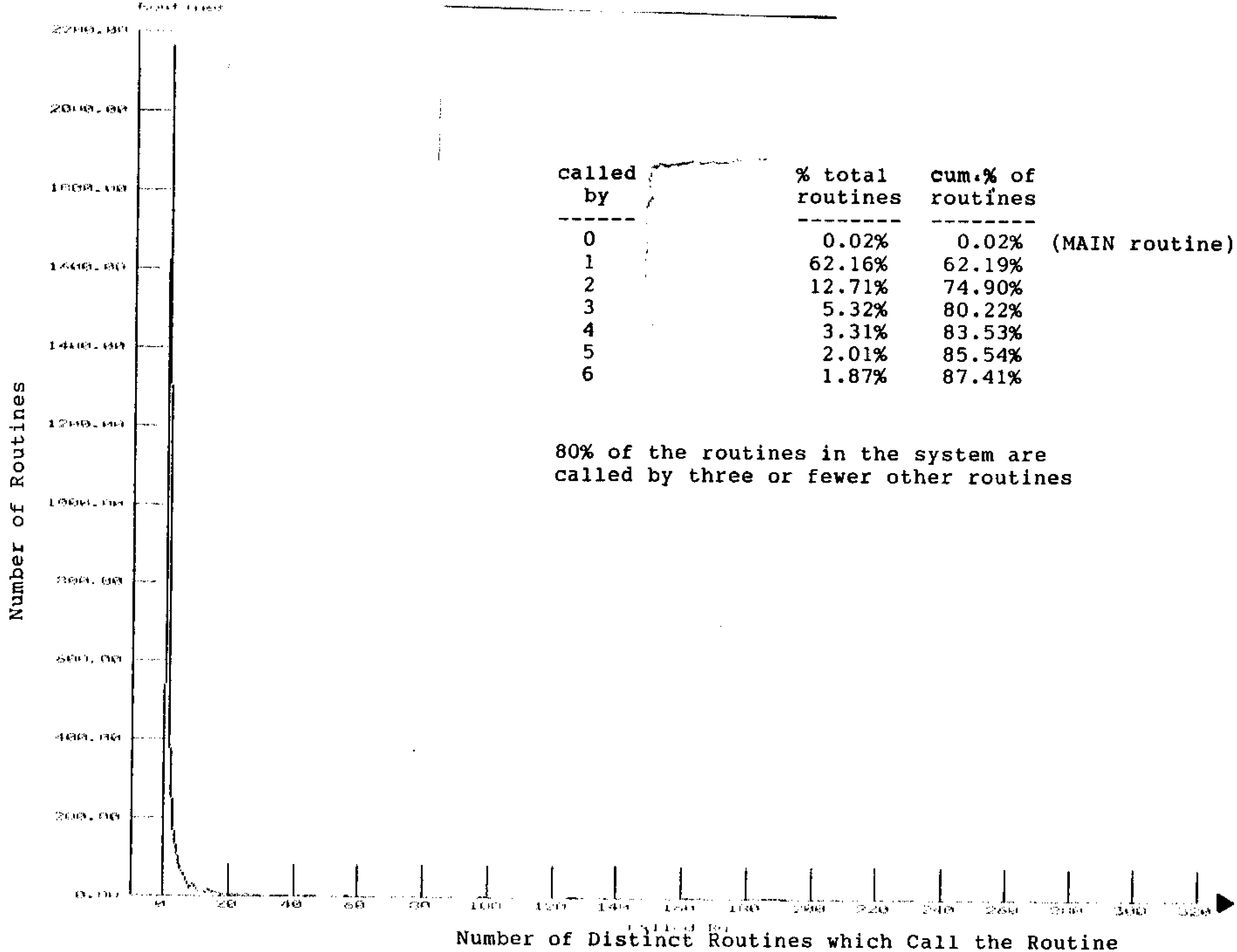
ENTER ROUTINE (IN SINGLE QUOTES), %DOWN, %UP



ENTER ROUTINE (IN SINGLE QUOTES), %DOWN, %UP



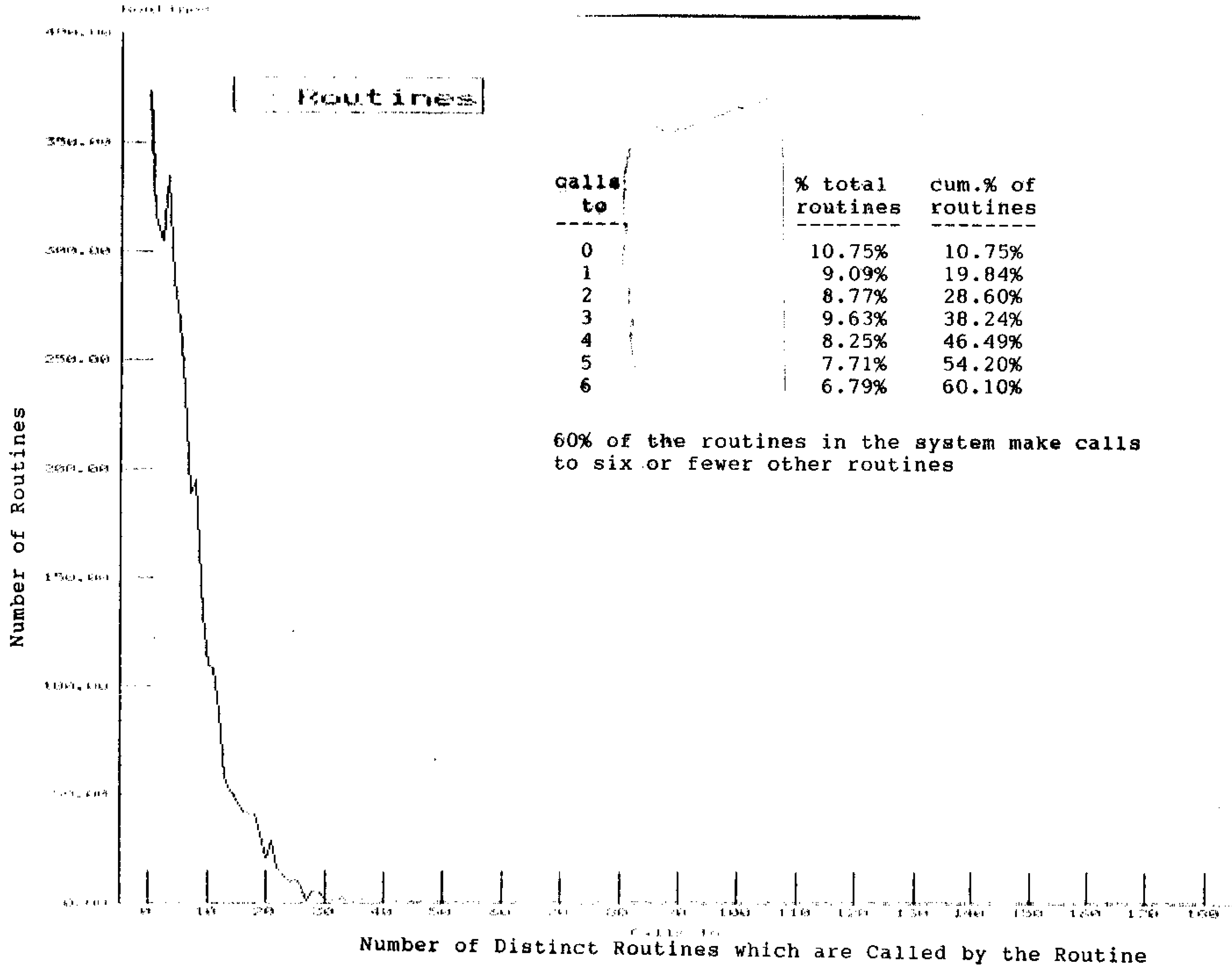
Routines Which are Called By N Other Routines



called by	% total routines	cum.% of routines	
0	0.02%	0.02%	(MAIN routine)
1	62.16%	62.19%	
2	12.71%	74.90%	
3	5.32%	80.22%	
4	3.31%	83.53%	
5	2.01%	85.54%	
6	1.87%	87.41%	

80% of the routines in the system are called by three or fewer other routines

Number of Routines which Make N Calls To Other Routines



calls to	% total routines	cum.% of routines
0	10.75%	10.75%
1	9.09%	19.84%
2	8.77%	28.60%
3	9.63%	38.24%
4	8.25%	46.49%
5	7.71%	54.20%
6	6.79%	60.10%

60% of the routines in the system make calls to six or fewer other routines

Routines Which are Called By N Other Routines