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

  - 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
Rehn Taagepera, The Study of Man, Vol 1, 1972

\[ \frac{dS}{dT} = kS \]

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

\[ h' = \text{waste accumulation rate} \]
\[ h'' = \text{waste virulence} \]

\[ \frac{dS}{dT} = k(M - S)(S - h \int S \, dT) \] where \[ h = h'h'' \]
Growth Curves of Different Systems

- critical mass
- maintenance doldrums
- reduce perceived interconnect complexity
- money manpower (Brooks' Law)

Operating Source Lines

- explosive growth
- normal growth
- excessive waste

Time

organizational limits
Oscillations as Development Limits Out

organizational limits

Operating Source Lines (not copied)

critical mass
Code "Reuse" During Maintenance Doldrums

Operating Source Lines

source lines including copied source lines

managerial "glow" sets in

source lines entered by programmers

organizational limits

Time

Neighbors 8/21/87
Sizes and Phases where Problems Occur

- **Analysis**: "No one knows why the system does that." 700,000
- **Architectural Design**: "No one knows how this system works." 400,000
- **Implementation**: "No one can replicate the bug." 100,000

Operating Source Lines vs. Time

Neighbors 8/21/87
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

Number of Routines

utility "bump"

Number of distinct routines which call the routine

small programs

Number of Routines

Number of distinct routines which call the routine

large systems

Number of Routines

Number of distinct routines which are called by the routine

both programs and systems

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

Neighbors 8/21/87
Routines Which are Called By N Other Routines

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

<table>
<thead>
<tr>
<th>called by</th>
<th>routines</th>
<th>% total</th>
<th>cum % of routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0.02%</td>
<td>0.02% (MAIN routine)</td>
</tr>
<tr>
<td>1</td>
<td>2162</td>
<td>62.16%</td>
<td>62.19%</td>
</tr>
<tr>
<td>2</td>
<td>442</td>
<td>12.71%</td>
<td>74.90%</td>
</tr>
<tr>
<td>3</td>
<td>185</td>
<td>5.32%</td>
<td>80.22%</td>
</tr>
<tr>
<td>4</td>
<td>115</td>
<td>3.31%</td>
<td>83.53%</td>
</tr>
<tr>
<td>5</td>
<td>70</td>
<td>2.01%</td>
<td>85.54%</td>
</tr>
<tr>
<td>6</td>
<td>65</td>
<td>1.87%</td>
<td>87.41%</td>
</tr>
</tbody>
</table>

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

Number of Routines = \((\text{Distinct Calling Routines})^{-2.08 \times 2180}\)
Note: don't take this too seriously, the branch ratio by itself is not strong enough to characterize a routine.

Number of Distinct Routines which are Called by the Routine

Number of Distinct Routines which Call the Routine

Routine Branching Ratios

system utilities

module utilities

subsystem utilities

interface

control

subsystem control

module control

system control

functional agents

scatterplot
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

interface routines
functional decomposition
functional agents
utility routines

to enclosing utilities

□ = routine, 20 – 150 in a subsystem average about 35

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

- Literature
  - software technology developments

- Modeling Domain Analyst
  - modeling domains

- Domain Designer
  - interconnected domain structure

- Application Domain Analyst
  - application domains

- Users of Similar Systems
  - executable system

- Users with a Specific System Need
  - needs for a specific system

- Systems Analyst
  - functional requirements

- Systems Designer
  - available domains

- Machine
  - domain language system description and refinement info

Neighbors 8/21/87
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

<table>
<thead>
<tr>
<th>called by</th>
<th>% total routines</th>
<th>cum.% of routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.02%</td>
<td>0.02%</td>
</tr>
<tr>
<td>1</td>
<td>62.16%</td>
<td>62.19%</td>
</tr>
<tr>
<td>2</td>
<td>12.71%</td>
<td>74.90%</td>
</tr>
<tr>
<td>3</td>
<td>5.32%</td>
<td>80.22%</td>
</tr>
<tr>
<td>4</td>
<td>3.31%</td>
<td>83.53%</td>
</tr>
<tr>
<td>5</td>
<td>2.01%</td>
<td>85.54%</td>
</tr>
<tr>
<td>6</td>
<td>1.87%</td>
<td>87.41%</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Calls to</th>
<th>% total routines</th>
<th>cum.% of routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10.75%</td>
<td>10.75%</td>
</tr>
<tr>
<td>1</td>
<td>9.09%</td>
<td>19.84%</td>
</tr>
<tr>
<td>2</td>
<td>8.77%</td>
<td>28.60%</td>
</tr>
<tr>
<td>3</td>
<td>9.63%</td>
<td>38.24%</td>
</tr>
<tr>
<td>4</td>
<td>8.25%</td>
<td>46.49%</td>
</tr>
<tr>
<td>5</td>
<td>7.71%</td>
<td>54.20%</td>
</tr>
<tr>
<td>6</td>
<td>6.79%</td>
<td>60.10%</td>
</tr>
</tbody>
</table>

60% of the routines in the system make calls to six or fewer other routines.
Routines Which are Called By N Other Routines

Number of Routines = (Distinct Calling Routines)\(^{-2.08}\) \times 2180