If Software Could Talk...

... why changes are made.

Audris Mockus

audris@research.bell-labs.com

http://www.bell-labs.com/~audris

Motivation

- Software databases
  - always there
    - large
    - uniform over time
- Empirically based understanding - lacking
  - GQM experiments - difficult
  - not usable as project tracking tools
  - can not be widely deployed

Approach

- Any project has VCS data
  - Derive/augment data to:
    - explain variability in quality, effort, interval
  - Validate
    - survey, other products
  - Apply the results for
    - process understanding (org., decay, infrast.)
    - problem localization and tracking

Outline

- Switching software change data
- Classification of changes
  - automatic algorithm
    - experimental verification
- Size and difficulty of changes
- Other applications
  - org. theory, code decay, expertise localization, effort estimation

Software projects

- two decades of development
- distributed/real-time software
  - 8x more complex than application software (SEI)
- scale:
  - 100 million lines of code
  - 100 thousand pages documentation
  - 20 supported versions
- sophisticated development process
- thousands of software engineers

How Code Evolves

- By adding and deleting line blocks

before: after:

```
  // initialize
int i=0;
while (i++)
   read (x);
```

```
int i=0;
while (i++ < N)
read(x);
```

- one line deleted
  - two lines added
  - two lines unchanged

Why study changes

- Reflect relationships between
  - requirements and design
  - technology and implementation
  - personnel (organization)
  - time (evolution of the system)
- Practical
  - always documented by VCS
  - results have wide applicability

Any VCS Records:

- Change - added and deleted lines
  - Who - login, organization
  - When - date and time
  - Description - line of text
  - Available data:
    - ~100M lines, ~3M changes, ~5K logins, 20Gb
      - ~30 products (select one)

No VCS Records for:

- Why
  - fix, new, improve, ...
- How difficult
  - effort, interval, complexity
- Will it cause fault in the future
  - estimate fault potential

Why Change?

- Primary reasons for maintenance activities
  - corrective: fix faults
  - adaptive: add features
- How those reasons relate to:
  - interval, effort, quality
  - developer, size
  - location, time

How to extract

- Use change description line
  - extract frequent keywords
  - classify keywords (fix, new, add, etc.)
    - discover new types
      - perfective - code cleanup
      - inspection - code inspection suggestions
    - verify on sample abstracts
  - keyword -> purpose of the change
  - iterate

Keywords

Adaptive:	**Corrective:**
add, new, create, initial coding, modify, update	fix, bug, error, problem, incorrect, must, needs
Perfective:	**Inspection:**
cleanup, remove, clear, unneeded, flex name	code review, inspection, rework, walkthrough

Proportions

- Why:
  - add new functionality - 45%
  - fix faults (bug) - 34%
  - cleanup/restructure - 4%
  - code inspection - 5%
  - unclassified - 12%

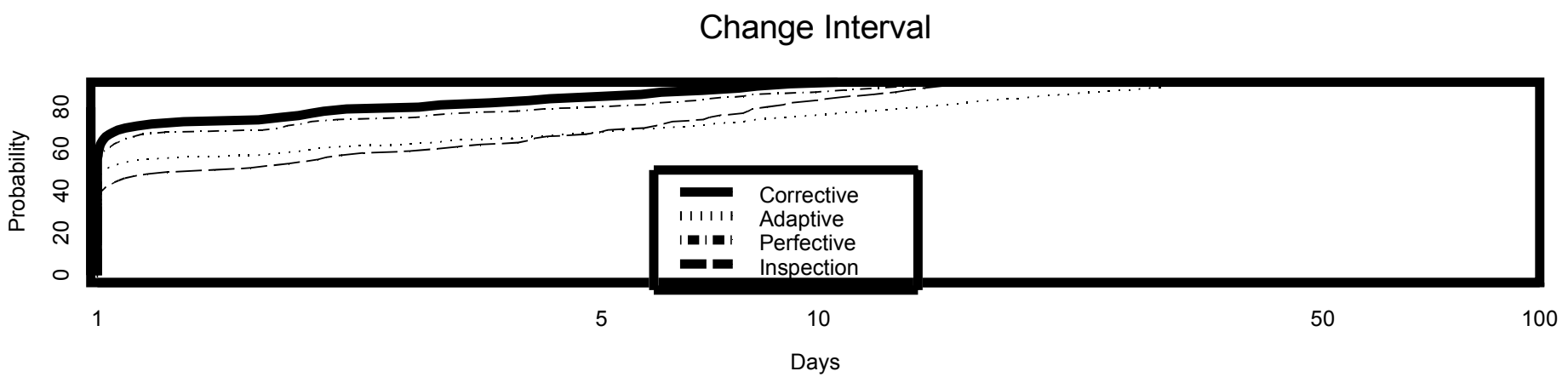
Is it right?

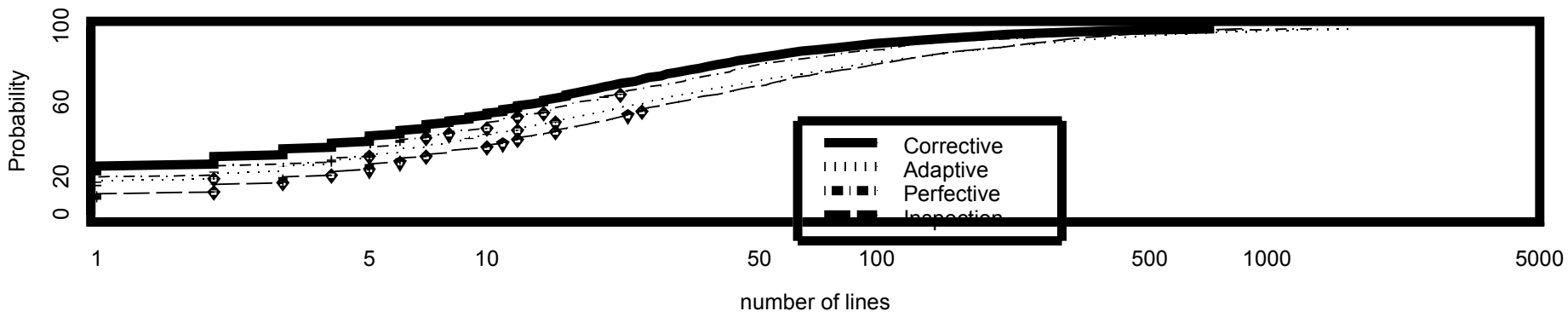
- Survey:
  - 2+5 developers (>9 years experience)
  - 20+150 changes (< 2 years old)
    - ~ equal numbers for different types
      - small percent of all changes developers did
- Questions
  - Type: bug, new, cleanup, inspection
  - Difficulty: Easy, Medium, Hard

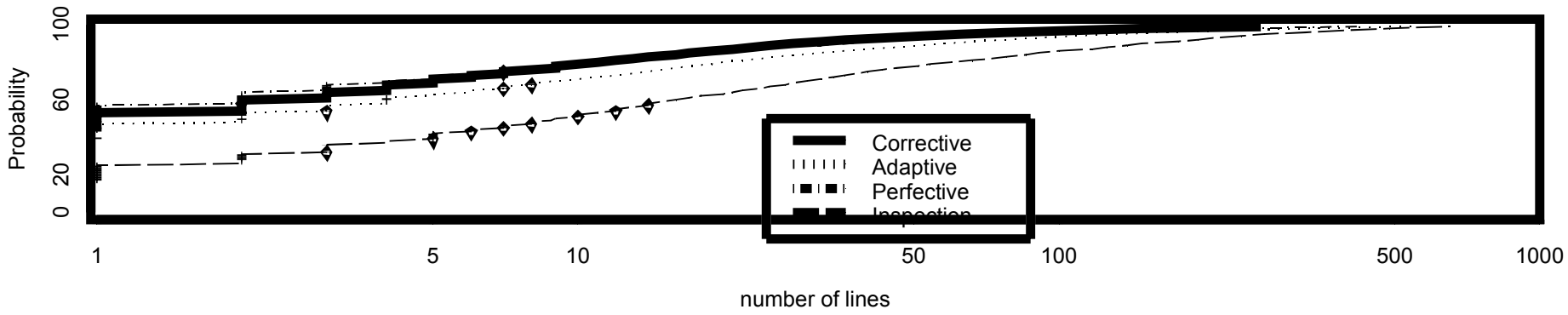
Results

- Unclassified changes are mostly bug fixes
  - Almost perfect match
    - Inspection changes are easiest to detect

Dev.\Auto	Corrective	Adaptive	Perfective	Inspection
Corrective	**35**	**10**	**5**	**1**
Adaptive	**11**	**23**	**3**	**4**
Perfective	**10**	**8**	**27**	**9**
Inspection	**1**	**0**	**0**	**21**


Change Interval
Probability
0
20
40
60
80
Corrective
Adaptive
Perfective
Inspection
1
5
10
50
100
Days

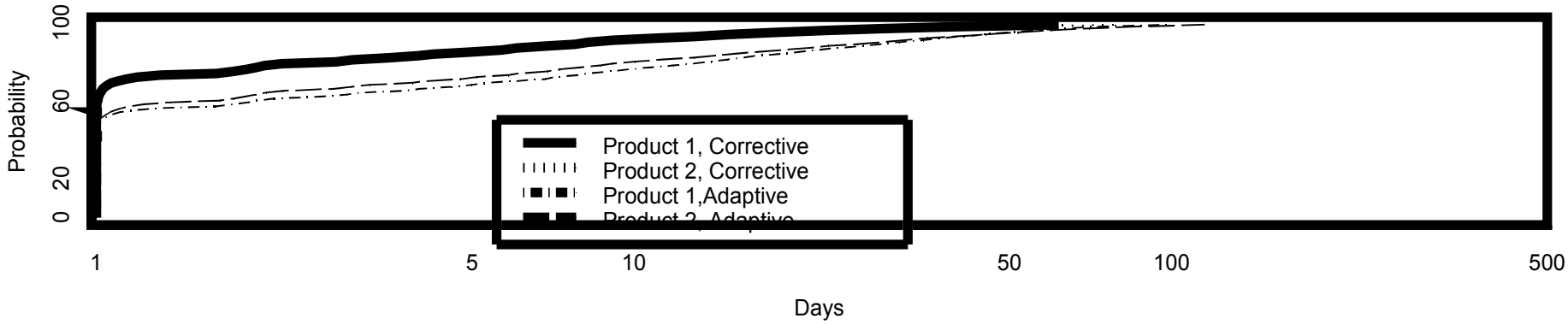

Lines Added
Probability
0
20
60
100
Corrective
Adaptive
Perfective
Inspection
1
5
10
50
100
500
1000
5000
number of lines


Lines Deleted
Probability
0
20
60
100
Corrective
Adaptive
Perfective
Inspection
1
5
10
50
100
500
1000
number of lines

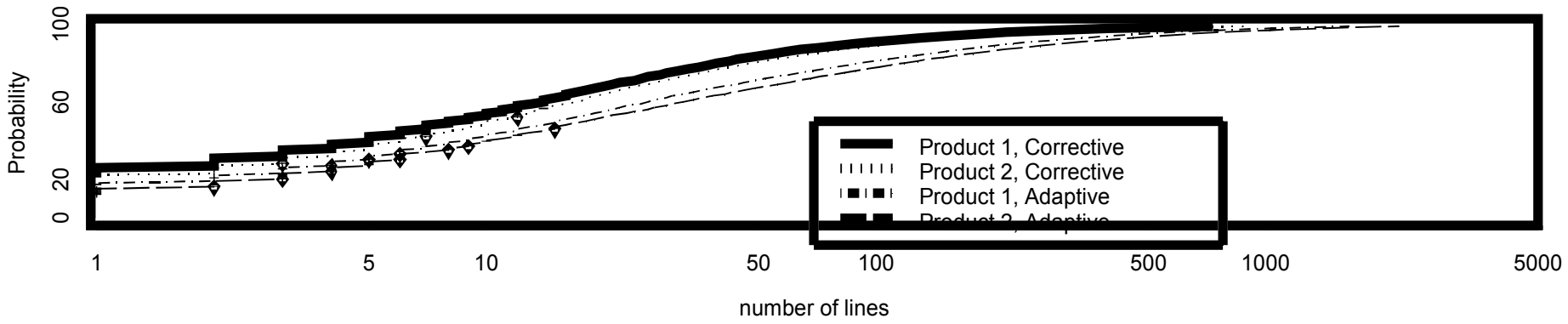
Will it work?

- Other Product
  - 2 X size and five years older
    - different functionality
    - different organization
- Tool
  - the same classification (no manual input)
- Results
  - very similar purpose profiles

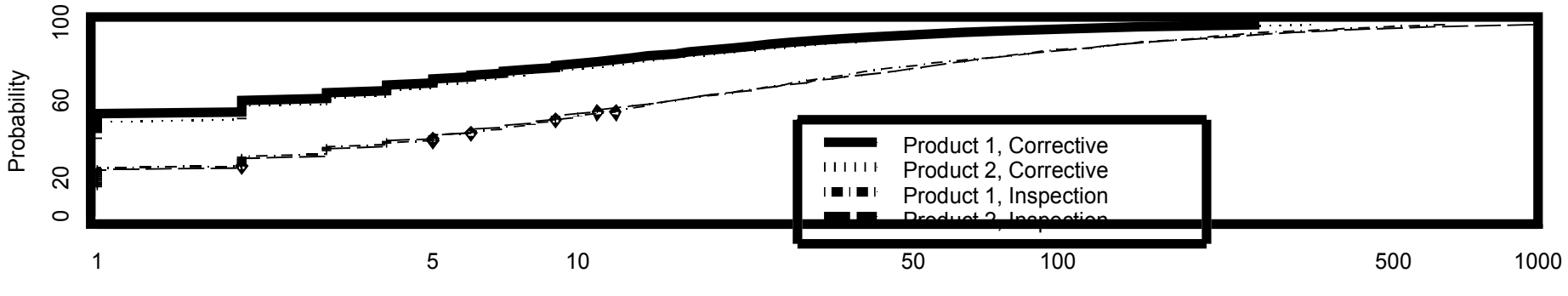
Change Interval



Lines Added



Lines Deleted



Why changes are difficult?

$$Difficulty \sim Fix + Developer + Size + \log(Interval)$$

- Fixes are hard - no matter size
  - Interval barely important adjusted for size
  - Type and size are more important than developer

Factor	DF	SS	p-val	effect
Fix	1	12	0	+
Size	1	7	0	+
Developer	6	10	0	
log(interval)	1	1.4	.027	+
Cleanup	1	.6	.12	+
Residuals	159	45		

Is change difficult?

- Difficult
  - more than 2 files touched, many delta, fault fix
- Frequently repeated, predominant
  - more 100 times, at least 30% of the time
- Are different parts equally difficult?
- Are changes becoming harder over time?
- Where to reengineer the code?

Summary

- Algorithm to extract purpose
  - automatic
    - validated by survey, on other product
- 4 types of changes discovered
  - different size, interval profiles
- Relationships
  - difficult - type
  - size, interval type

Summary

- Can VCS be used to find out:
  - why change is made
    - why change is difficult
- Obtain essential properties of changes
  - Data source available for all SW projects
  - Non-intrusive data collection
  - Methodology to describe software projects
- Potential to predict the impact of:
  - organizational (team size)
  - process (code inspections)
  - technology (compilers, computer languages)

Future & Current Work

- Refine classification
  - detail - type of fix - overflow, deadlock,...
  - domain - HW/SW, phase
- Utilize other databases
  - financial support system - effort
  - customer tracking - serious faults
- VCS enhancement tools
  - problem localization, project status

Fault Potential

- Do past changes predict future faults
  - predict proportion of faults
    - in a two year period
      - for 88 modules
  - numbers, sizes, age of changes
- Best predictor:
  - past number of faults
  - but NOT: complexity, connectivity, #authors

Can developers know:

- Which subsystems/modules are hard?
- What types of changes are frequent?
- Who writes the most code?

- Access platform goals:
  - standard Netscape interface
  - no software/data to install
  - point and click

Link: Developer activity

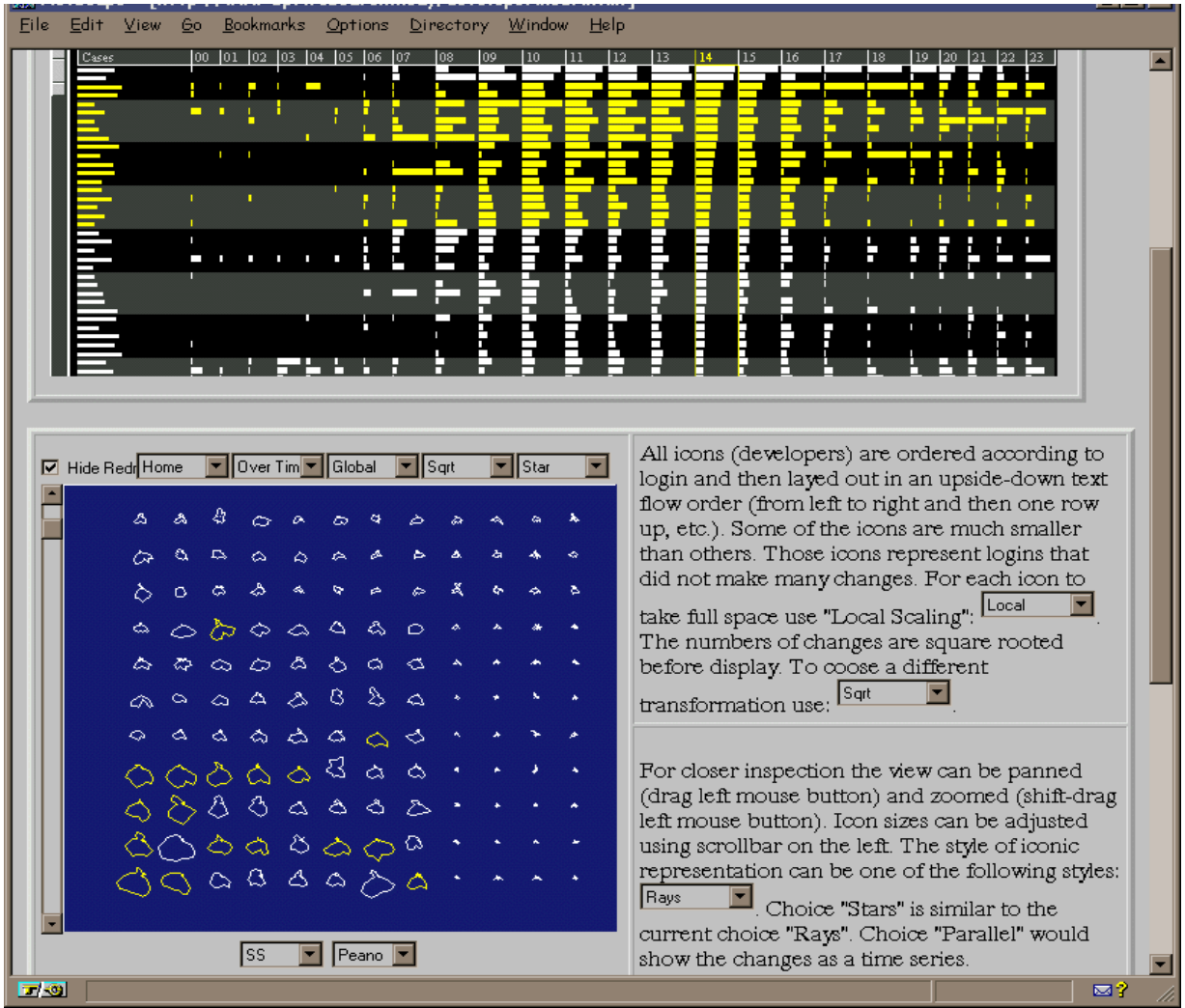
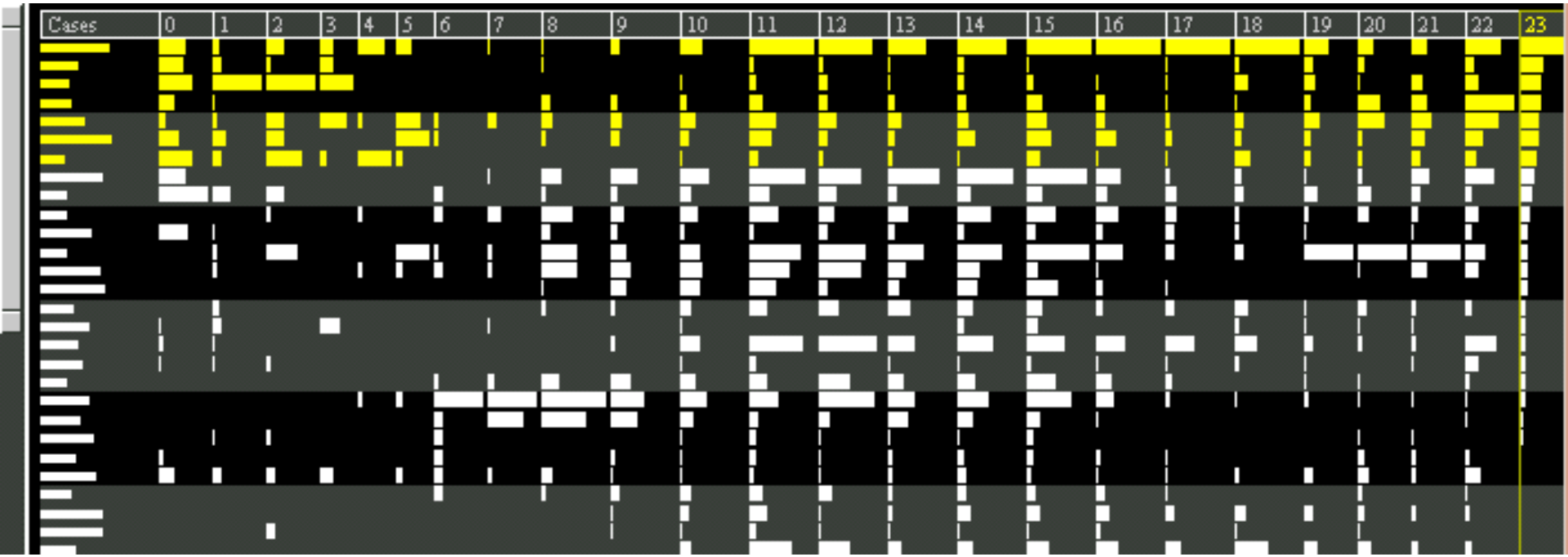
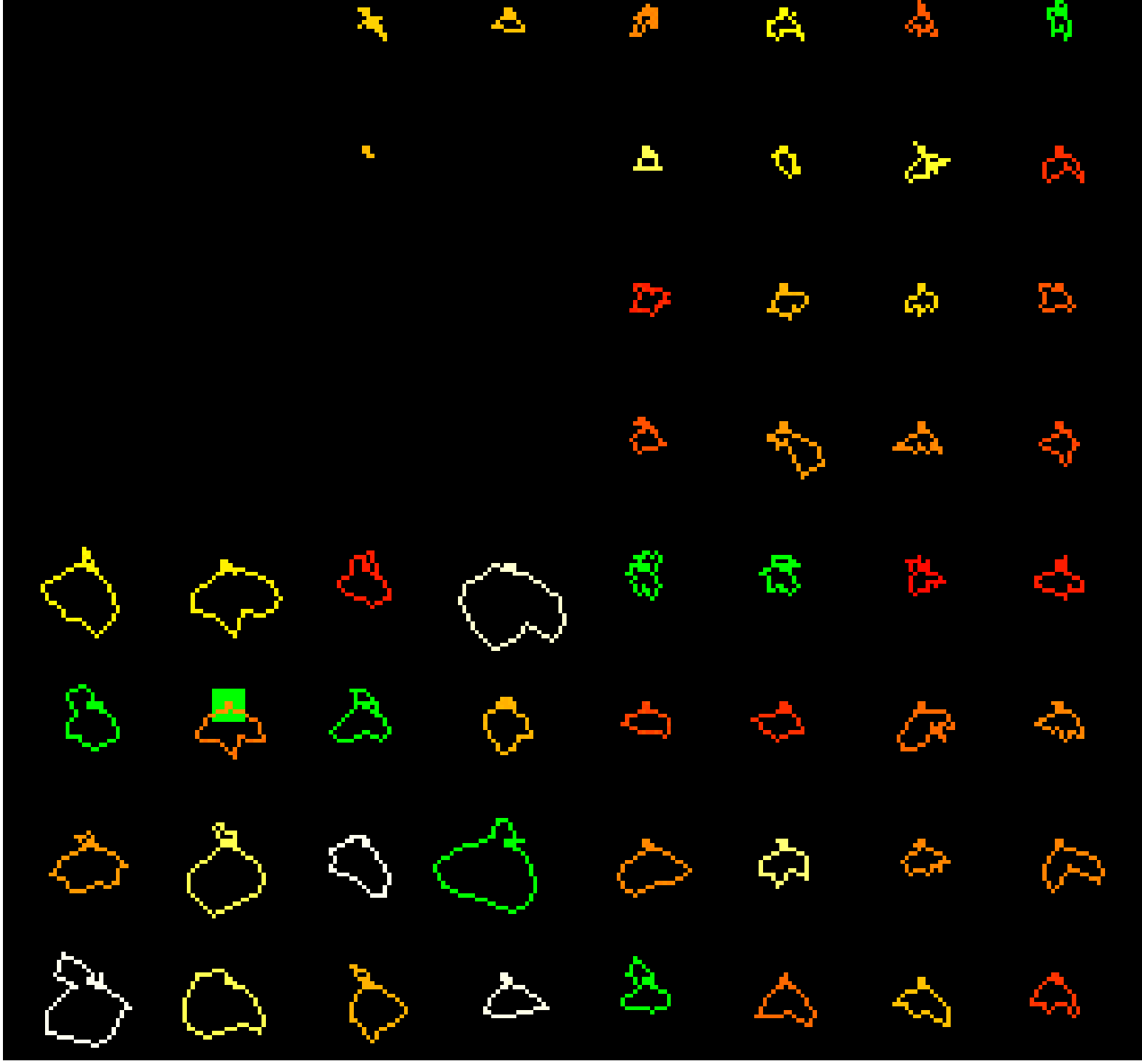


Table view





Cases	0	b	c	n
sae	462	1651	85	454
dipietro	817	1583	218	701
kjs	635	1225	521	184
scooby	433	1144	275	151
cuervo	1602	1090	452	772
kjs28	573	1016	295	376
mcy	534	977	3	251
rgs	496	729	216	159
claa	366	669	52	236
lisamp	197	640	78	67
tom	577	628	63	1169
louie	494	596	48	108
ar1	219	572	20	57
kenchan	106	508	32	42
chap	366	495	79	127

Cases	a	b	c	n
tom	577	628	63	1169
ccc	102	127	90	1040
emilyr	427	194	15	968
cuervo	1602	1090	452	772
dipietro	817	1583	218	701
atang	153	1	0	580
liuzzo	376	252	111	559
bhp	1101	469	137	554
jedavis	97	117	5	499
stanza	273	116	361	475
sae	462	1651	85	454
sunu	1618	150	94	449
cvb1	333	403	89	448
kern1	393	130	47	437
tuch	541	159	298	432

Cases	0	b	c	n
dlo	373	281	625	226
doris	1375	447	579	388
kjs	635	1225	521	184
cuervo	1602	1090	452	772
bmary	865	376	445	30
senior	723	149	386	243
stanza	273	116	361	475
rosson	383	140	317	19
rnts	54	84	312	320
tuch	541	159	298	432
kjs28	573	1016	295	376
scooby	433	1144	275	151
fenn	301	207	269	194
shp	165	406	228	180
dipietro	817	1583	218	701