Modeling Software Evolution

Audris Mockus
Bell Laboratories
Software Production Research
http://www.bell-labs.com/~audris

Overview

- Main issue in software production
  - expensive to maintain
- Solutions
  - understand how software evolves
  - use software data to aid decision making
    - track customer faults
    - what/when/how much to redesign/rewrite
- Issues
  - creative process, no repetition, not random
  - structured data (organization, code, process, usage)
  - non-uniform data across and within projects

Business Needs

- Satisfy customer
  - get new functionality fast
  - minimize faults and fix them fast
- Make developers more productive
  - enhance decision support tools
  - minimize data collection overhead
- Apply methodology to multiple projects
  - use generally available information
- Bottom line
  - make software easy to change

Large-Scale Software

- 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

Becomes hard to change - DECAYS

Software Data are Rich

- Interrelated hierarchies of:
  - **change:** delta ∈ MR ∈ IMR ∈ Feature
  - **organization:** developer ∈ group ∈ dept., ...
  - **structure:** file ∈ module ∈ subsystem | global declarations
  - **semantics:** data | call processing | network protocol | ...
  - **execution:** data flow | concurrency | call graph, ...
- Time structure
  - **phase:** requirements→design→implementation→testing→support
  - **calendar**: hour, day, week, month, year
  - **business**: market events and management decisions

How Code Evolves

- By adding and deleting line blocks

before: after:

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

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

What is Recorded?

- Change itself (added and deleted lines)
- Who made the change (login, organization)
- When the change was made - date and time
- Description of the change (text)
- Size:
  - ~100M lines, ~3M changes, ~5K logins

What is NOT Recorded

- Why
  - estimate from textual description
- How difficult
  - estimate from spread, size, and time
- Will it cause fault in the future
  - estimate fault potential
- Are new changes the same as old changes
  - delta $\rightarrow$MR $\rightarrow$IMR

Why the change was made?

- Hyperlink to Purpose
  - Add new functionality (new) 35%
  - Fix faults (bug) 30%
  - Cleanup/restructure (clean) 20%
  - Code inspection (inspection) 15%
- estimation using keyword
  - e.g. new, add → new; fix, fault → bug
- validated using IMR data, developer survey
  - 85% classified, 70% match developer opinions
- time/size/interval profiles

Is definition of a change constant?

- Data recorded/estimated
  - delta → login, size, file, MR
  - MR → description, interval, difficulty, type, IMR
  - IMR → 89 recorded fields

- Are MRs affected by management policy?

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

Is the change difficult?

- Difficult
  - more than 2 files touched, many delta, fault fix
- Frequently repeated, predominant
  - more 100 times, at least 30% of the time

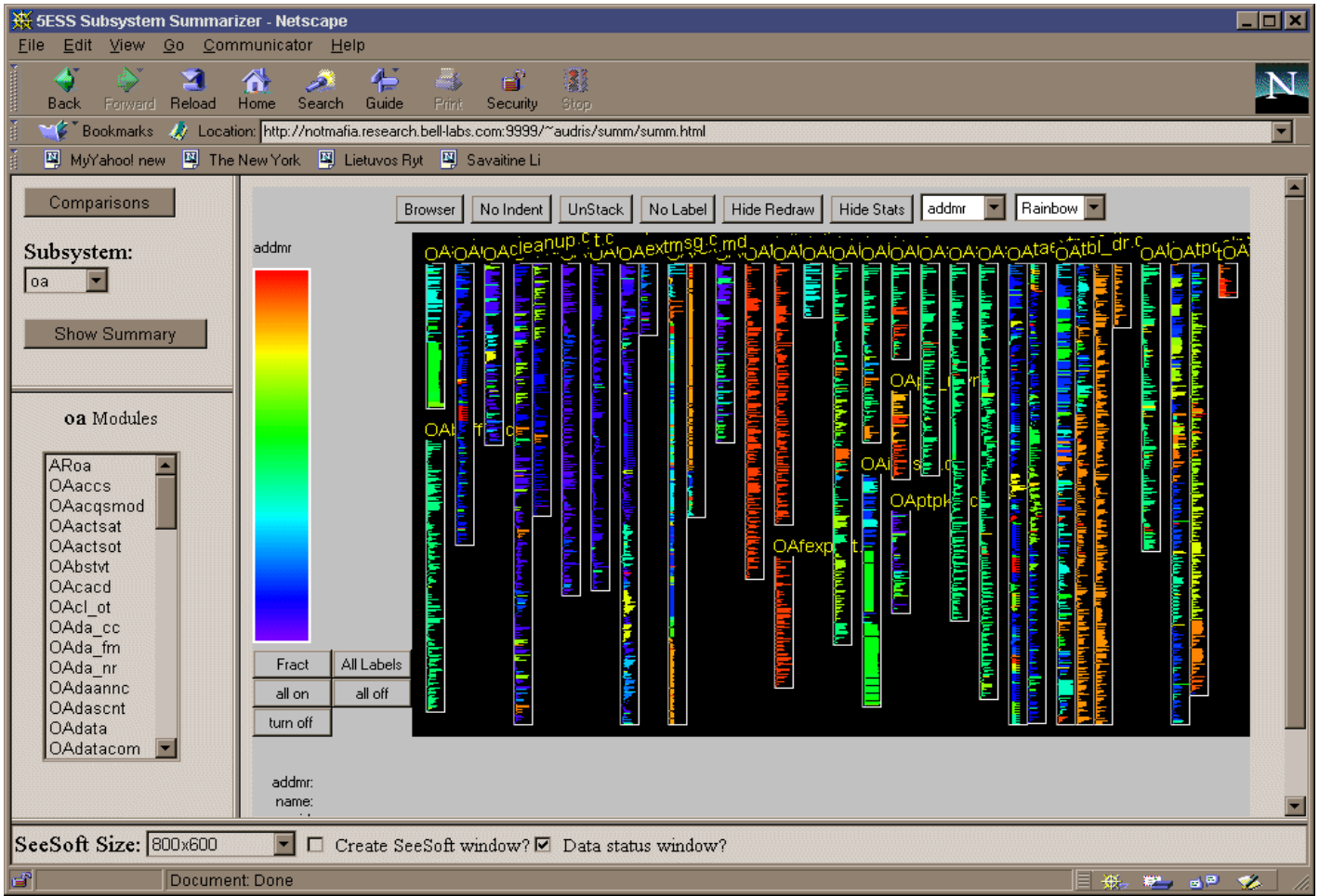
- Are different parts equally difficult?
- Are changes becoming harder over time?
- Where to reengineer the code?

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: Subsystem comparison, summa
SeeSoft drill-down to modules



Link: Developer activity

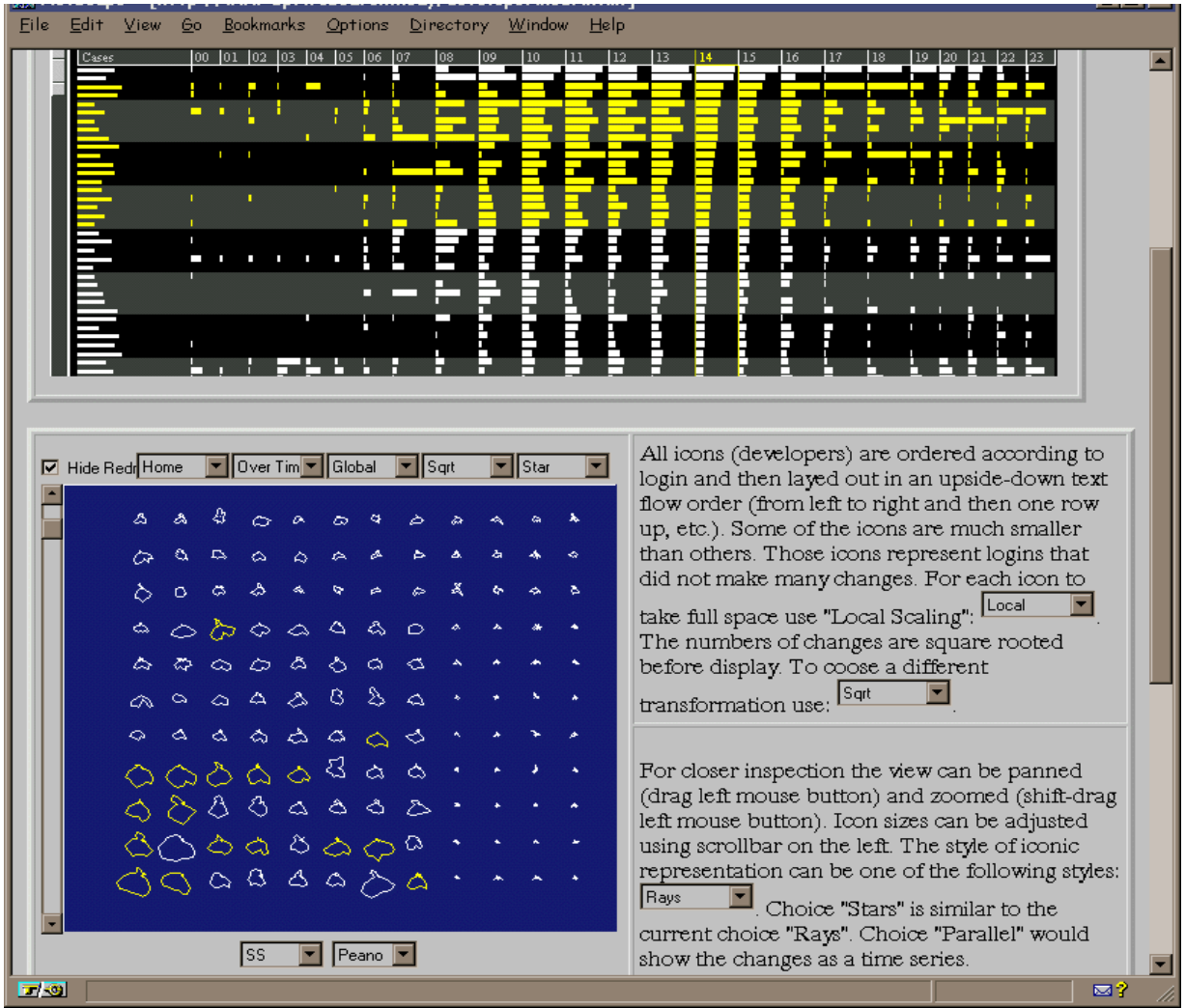
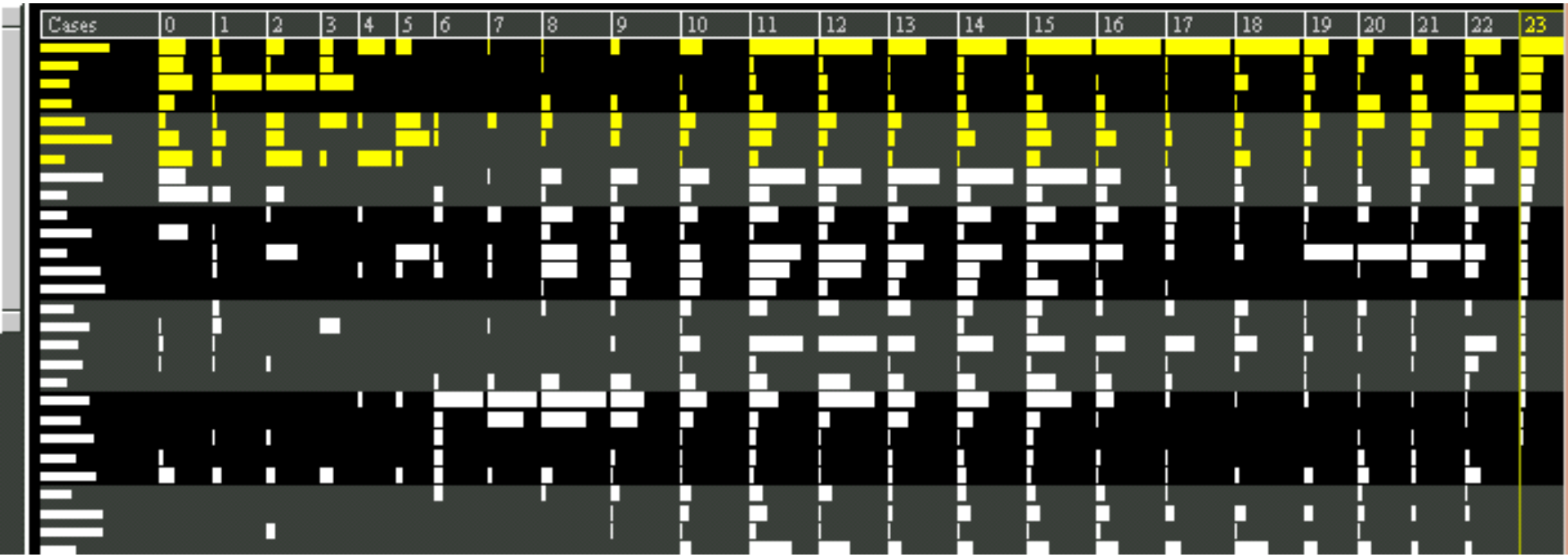
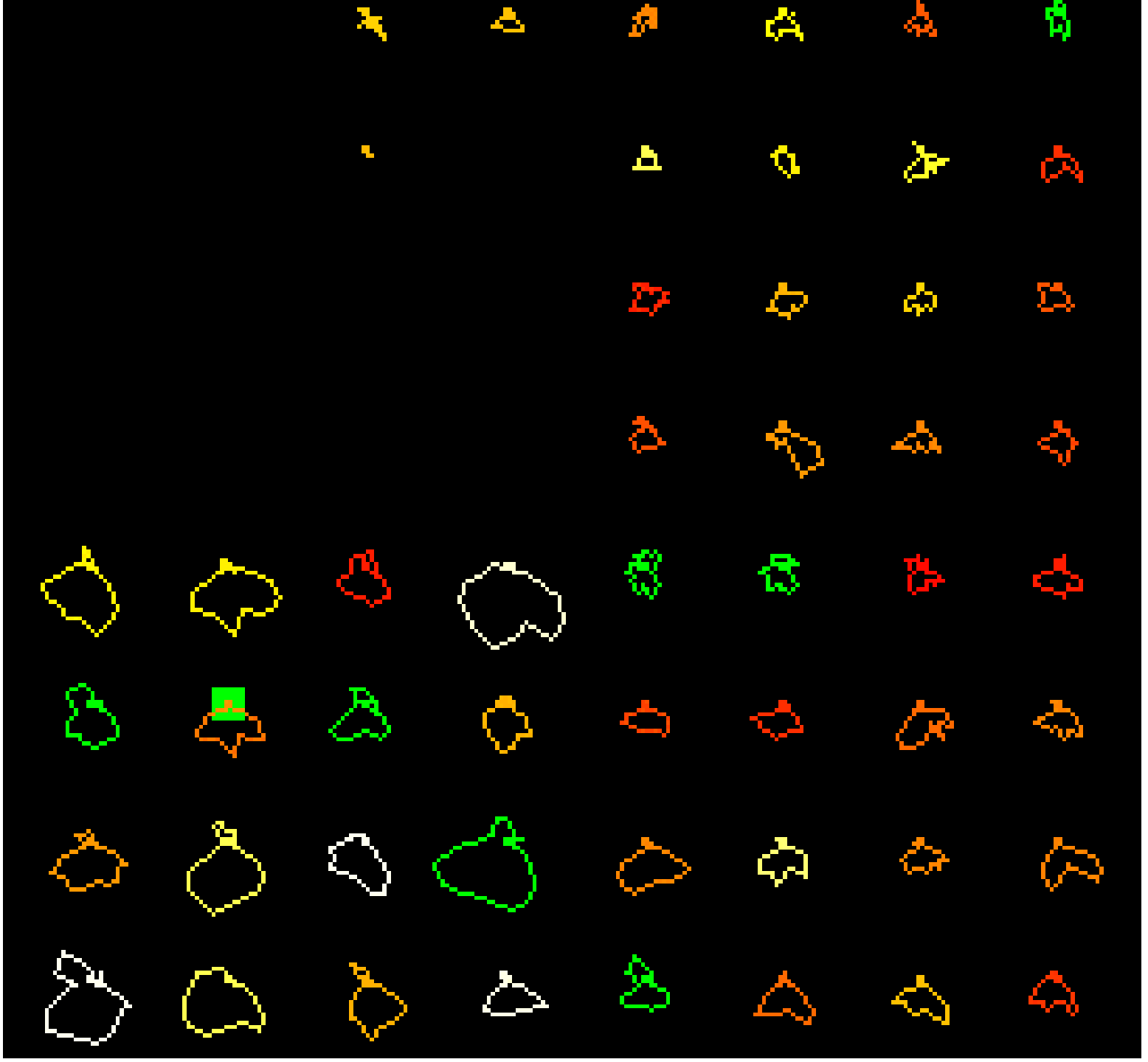
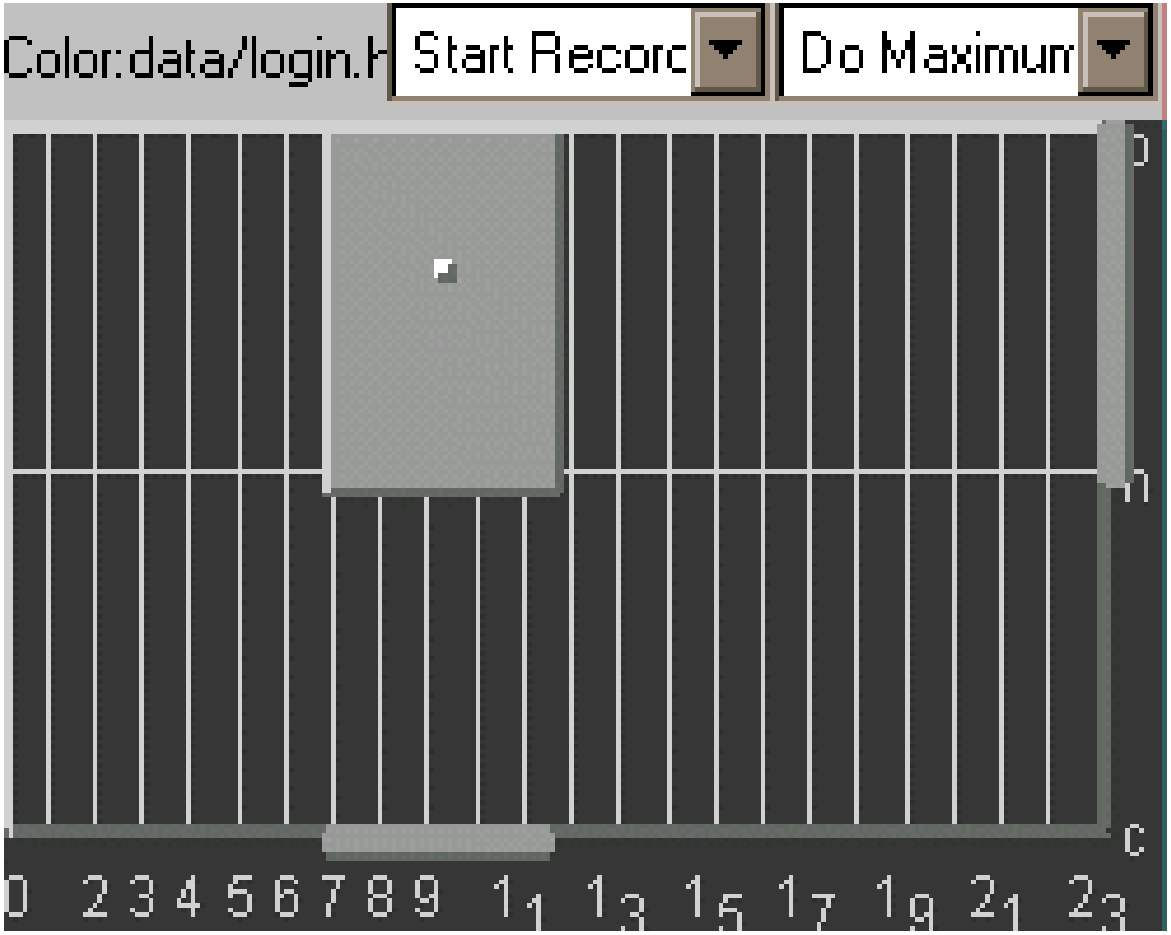


Table view





Aggregation eye



Summary

- Key problem - make software easier to change
  - 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
- Technology to distribute the results
- Potential to predict the impact of:
  - organizational (team size)
  - process (code inspections)
  - technology (compilers, computer languages)