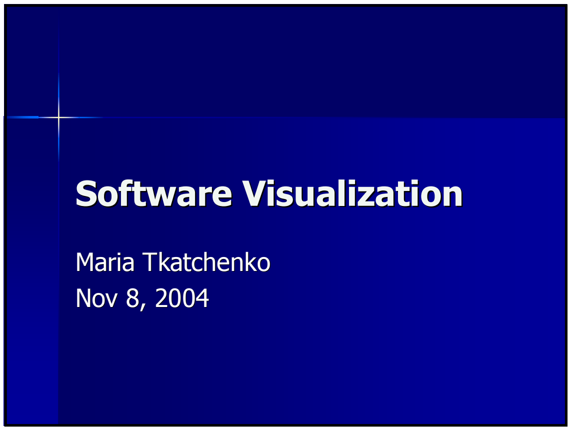
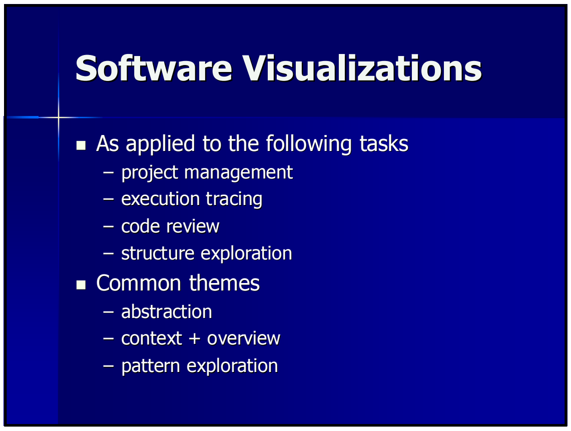


Software Visualization

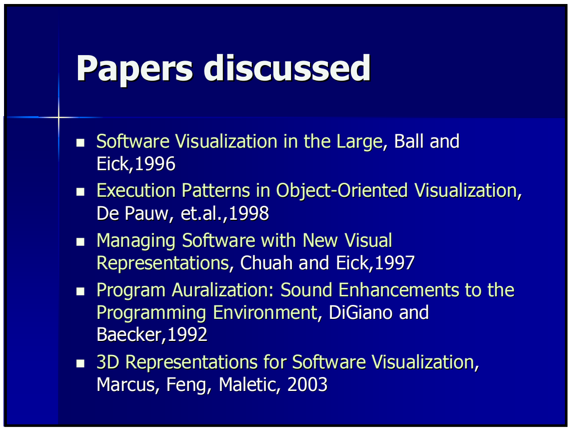
Maria Tkatchenko

Nov 8, 2004



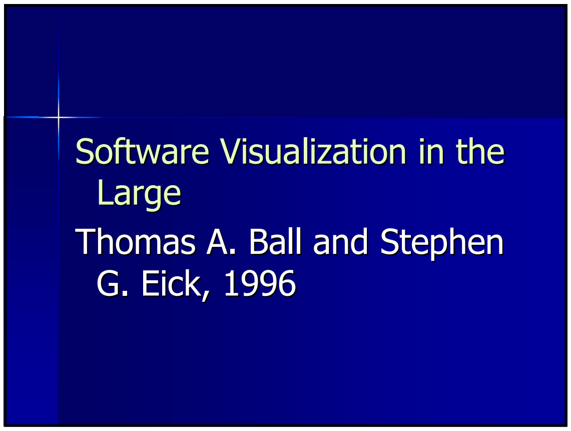
Software Visualizations

- As applied to the following tasks
  - project management
  - execution tracing
  - code review
  - structure exploration
- Common themes
  - abstraction
  - context + overview
  - pattern exploration



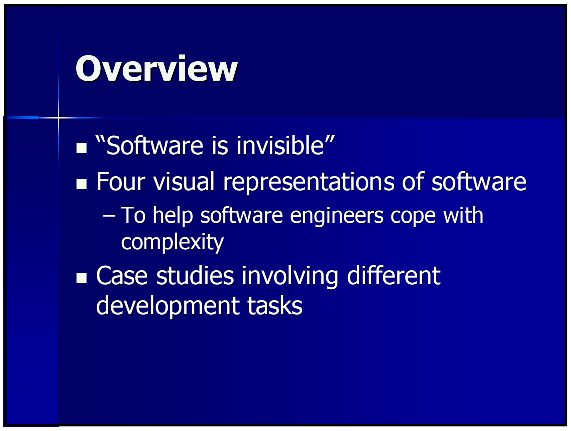
Papers discussed

- Software Visualization in the Large, Ball and Eick,1996
- Execution Patterns in Object-Oriented Visualization, De Pauw, et.al.,1998
- Managing Software with New Visual Representations, Chuah and Eick,1997
- Program Auralization: Sound Enhancements to the Programming Environment, DiGiano and Baecker,1992
- 3D Representations for Software Visualization, Marcus, Feng, Maletic, 2003



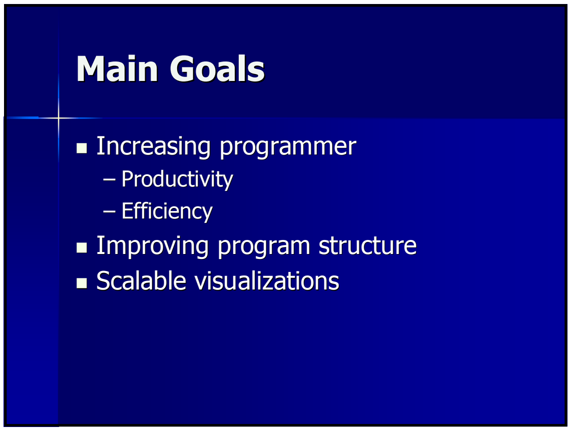
Software Visualization in the Large

Thomas A. Ball and Stephen G. Eick, 1996



Overview

- "Software is invisible"
- Four visual representations of software
  - To help software engineers cope with complexity
- Case studies involving different development tasks



Main Goals

- Increasing programmer
  - Productivity
  - Efficiency
- Improving program structure
- Scalable visualizations

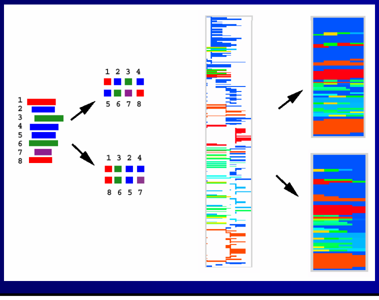
Visualizing software

- Structure
- Run-time behavior
- Code itself

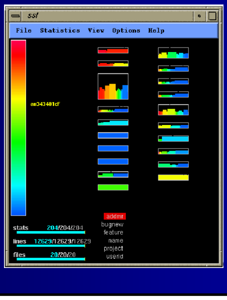
Visual representations

- Line representation
- Pixel representation
  - Show line as a pixel
- File summary representation
  - File as a rectangle, inner time-series
- Hierarchical representation
  - Zoomable tree-map

Pixel representation



File summary representation



stats 204/204/204
lines 12629/12629/12629
files 20/20/20

Critique (1)

- "Hiding system complexity… contributes to low programmer productivity"
  - Untrue of object-oriented
  - Good design, interfaces, documentation, etc. recover this
- IDEs and special purpose tools now deal with the issue identified

Critique (2)

- Need for textual visualization of a large system?
- Aim may be to condense too much information
- Good way to visualize non-functional properties of text if metadata available

Execution Patterns in Object-Oriented Visualization
Wim De Pauw, David Lorenz, John Vlissides, and Mark Wegman.,1998

Overview

- Visualizing execution traces of object-oriented programs
- Explore at different levels of abstraction
- Classification of behavior into patterns
- Goals of tools:
  - Explore structure of execution
  - Find areas to optimize

Current execution tracing

- Textual
  - Too much detail in output
  - Hard to control
- OO visualization systems
  - Microscopic – sequence of message sends
  - Macroscopic - cumulative
- Very difficult to scale

Execution pattern view

- Observe any part of the programs execution at various levels of detail
  - Detail on demand
- Detect and present generalized patterns of execution
  - Pattern subsumes many parts of the trace
- Figure from paper

Interaction diagram vs. execution pattern

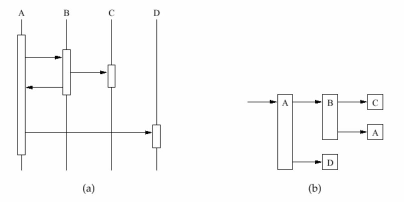


Figure 1: Simple interaction diagram (a) and its corresponding execution pattern (b)

Useful features

- Collapsing/expanding subtrees
- More clear notation for interaction diagrams
- Easy change in level of abstraction and view
- Detection and collapsing of repetitions
- Tree operations
  - Flattening
  - Overlaying

Flattening

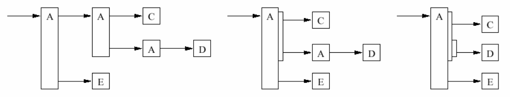


Figure 14: Using flattening to visualize association

Pattern detection

- Not only reduces clutter, but makes things explicit
- Similar vs. identical
  - Automatic pattern detection important
  - The slight differences often not that important to the programmer
- Pattern matching
  - Automatic
  - Tools for programmer to express similarity

Patterns

- Identity
- Class Identity ***
- Message Structure
- Depth-Limiting
- Repetition ***
- Polymorphism ***
- Associativity
- Commutativity ***

Experimental results

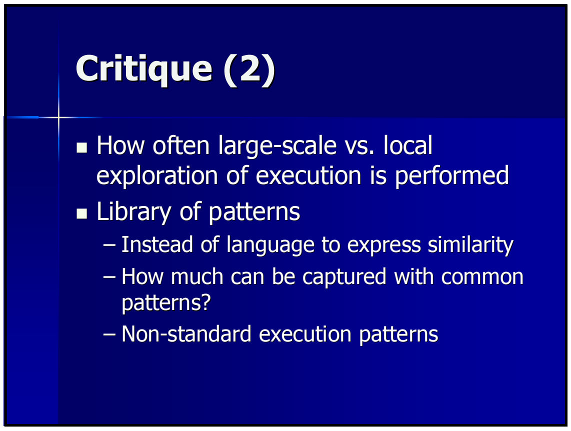
- Uncover unexpected behavior
- Help understand unfamiliar code
- Improve performance

Contributions

- Intuitive and scalable metaphor
- Generalization of similar execution patterns
- Execution patterns allow to characterize system complexity

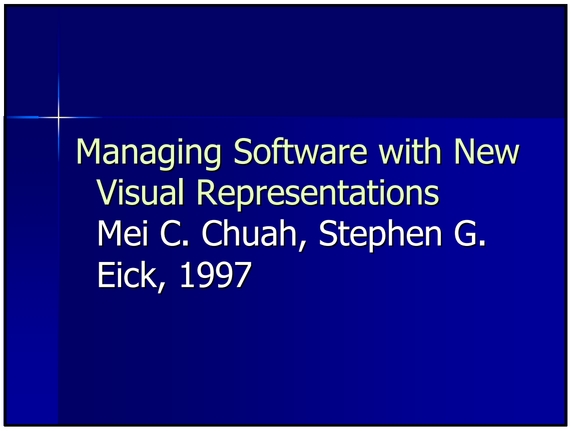
Critique (1)

- Collapsing of repetition is a great idea
  - Use for design as well as analysis
- Good use of the OO programming principles and metaphor
- Learning curve for distinguishing patterns and classes

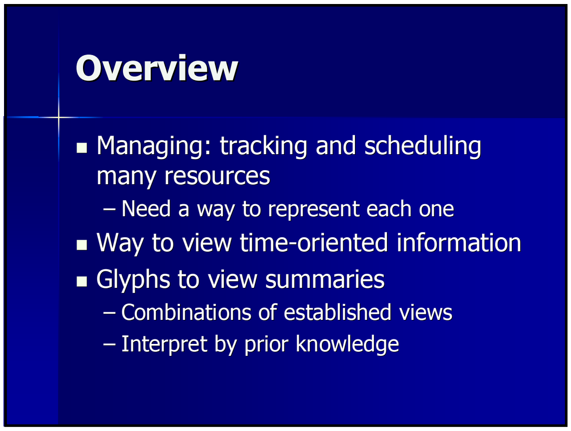


Critique (2)

- How often large-scale vs. local exploration of execution is performed
- Library of patterns
  - Instead of language to express similarity
  - How much can be captured with common patterns?
  - Non-standard execution patterns

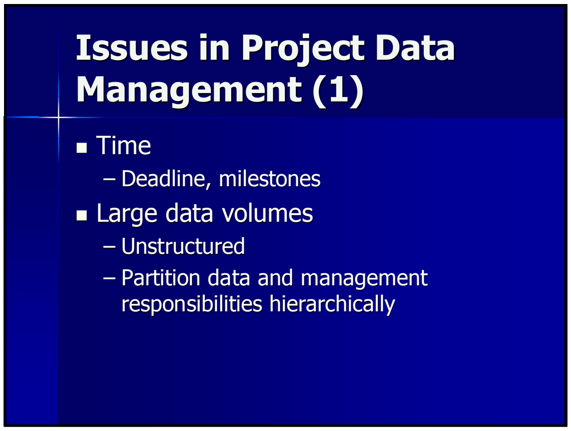


Managing Software with New Visual Representations
Mei C. Chuah, Stephen G. Eick, 1997



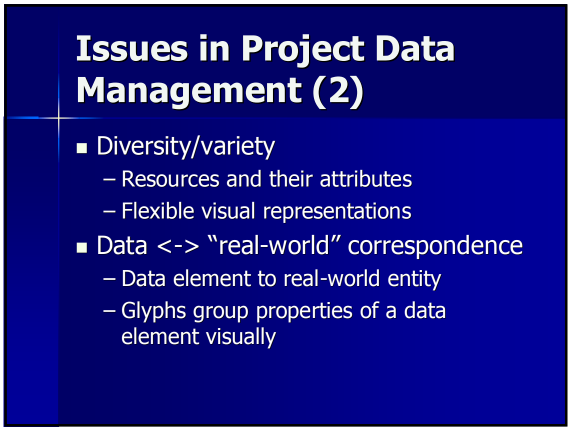
Overview

- Managing: tracking and scheduling many resources
  - Need a way to represent each one
- Way to view time-oriented information
- Glyphs to view summaries
  - Combinations of established views
  - Interpret by prior knowledge



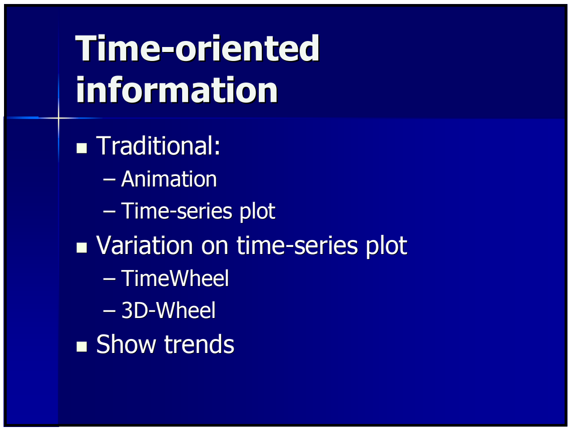
Issues in Project Data Management (1)

- Time
  - Deadline, milestones
- Large data volumes
  - Unstructured
  - Partition data and management responsibilities hierarchically



Issues in Project Data Management (2)

- Diversity/variety
  - Resources and their attributes
  - Flexible visual representations
- Data <-> “real-world” correspondence
  - Data element to real-world entity
  - Glyphs group properties of a data element visually



Time-oriented information

- Traditional:
  - Animation
  - Time-series plot
- Variation on time-series plot
  - TimeWheel
  - 3D-Wheel
- Show trends

TimeWheel (1)

- Each object attribute a time-series
- Individual time-series laid out around a circle
- Preattentively pick out objects
- Small multiples show:
  - General trend
  - divergences

TimeWheel (2)

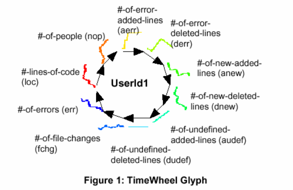

#-of-error-added-lines (aerr)
#-of-error-deleted-lines (derr)
#-of-people (nop)
#-of-new-added-lines (anew)
#-lines-of-code (loc)
UserId1
#-of-new-deleted-lines (dnew)
#-of-errors (err)
#-of-undefined-added-lines (audef)
#-of-file-changes (fchg)
#-of-undefined-deleted-lines (dudef)

Figure 1: TimeWheel Glyph

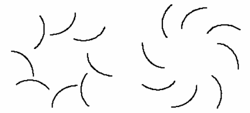


Figure 2: Left - increasing trend timeWheel (prickly fruit); Right- decreasing trend timeWheel (hairy fruit)

TimeWheel (4)

- Advantages over linear:
  - Reduce number of eye movements
  - Less susceptible to local patterns
  - No ordering implication from reading
  - Higher information density

3D-Wheel (1)

- Same as TimeWheel, use height to encode time
- Dominant time trend through shape
- Common problem of occlusion
  - Hard to identify divergences from trend

3D-Wheel (2)

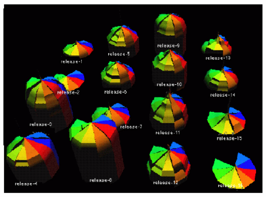


Figure 9: 3D wheel interface of the 16 software releases shown in Figure 3

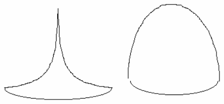
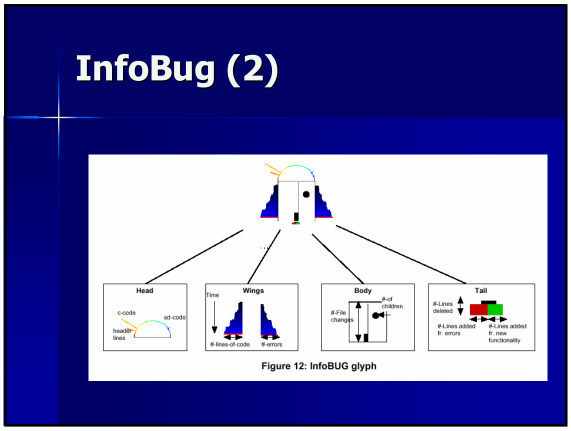


Figure 8: Left - increasing trend (sharp apex); Right - decreasing/tapering trend (balloon)

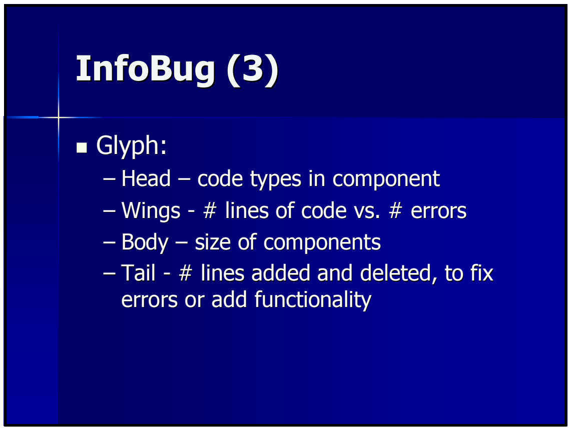
InfoBug (1)

- Interactive
- Use animation to show at different times within the project
- Small footprint
- Preattentive patterns



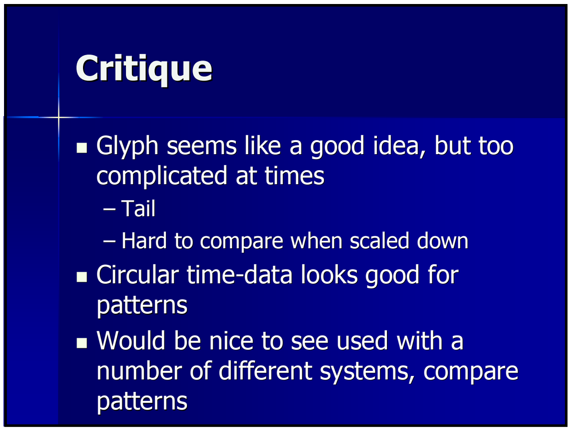
InfoBug (2)

Figure 12: InfoBUG glyph



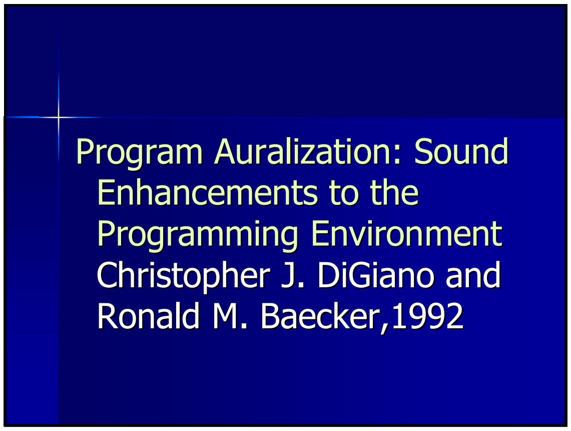
InfoBug (3)

- Glyph:
  - Head – code types in component
  - Wings - # lines of code vs. # errors
  - Body – size of components
  - Tail - # lines added and deleted, to fix errors or add functionality

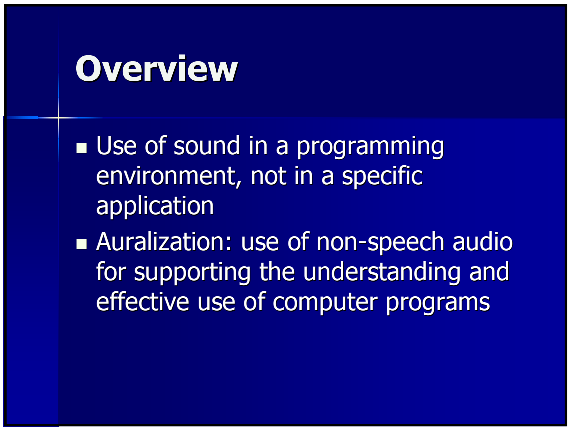


Critique

- Glyph seems like a good idea, but too complicated at times
  - Tail
  - Hard to compare when scaled down
- Circular time-data looks good for patterns
- Would be nice to see used with a number of different systems, compare patterns

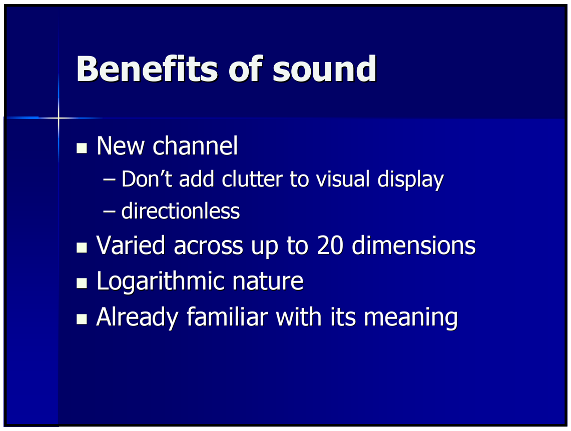


Program Auralization: Sound Enhancements to the Programming Environment
Christopher J. DiGiano and Ronald M. Baecker,1992



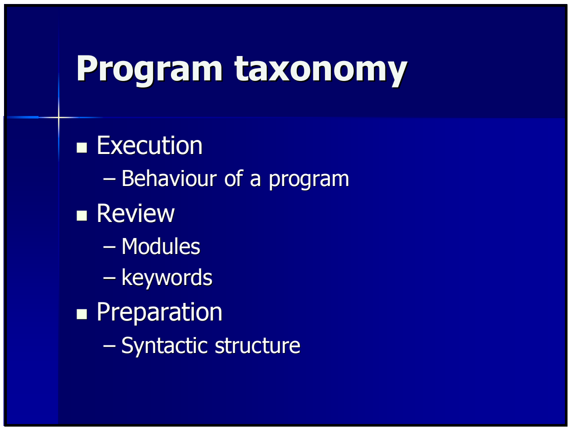
Overview

- Use of sound in a programming environment, not in a specific application
- Auralization: use of non-speech audio for supporting the understanding and effective use of computer programs



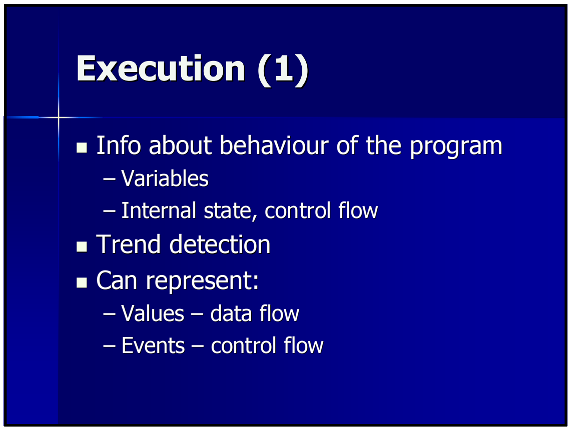
Benefits of sound

- New channel
  - Don’t add clutter to visual display
  - directionless
- Varied across up to 20 dimensions
- Logarithmic nature
- Already familiar with its meaning



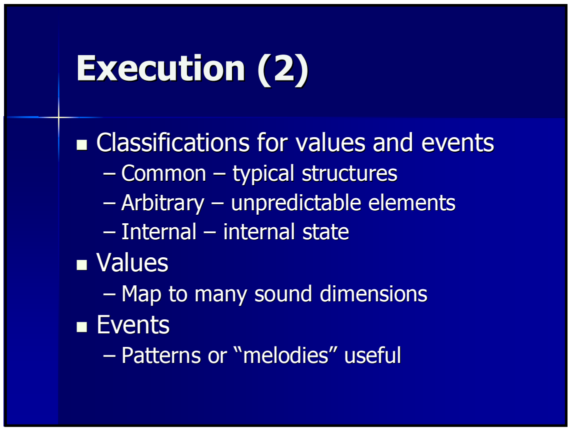
Program taxonomy

- Execution
  - Behaviour of a program
- Review
  - Modules
  - keywords
- Preparation
  - Syntactic structure



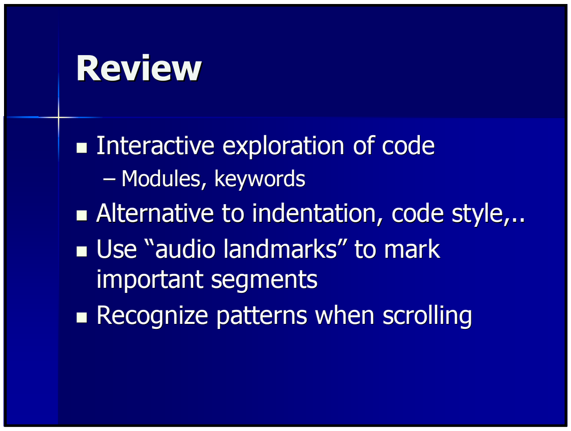
Execution (1)

- Info about behaviour of the program
  - Variables
  - Internal state, control flow
- Trend detection
- Can represent:
  - Values – data flow
  - Events – control flow



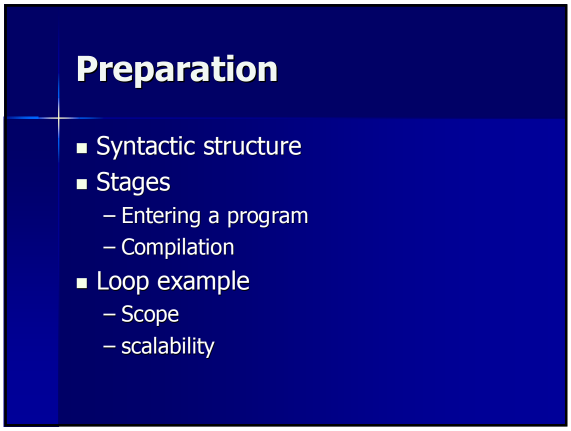
Execution (2)

- Classifications for values and events
  - Common – typical structures
  - Arbitrary – unpredictable elements
  - Internal – internal state
- Values
  - Map to many sound dimensions
- Events
  - Patterns or "melodies" useful



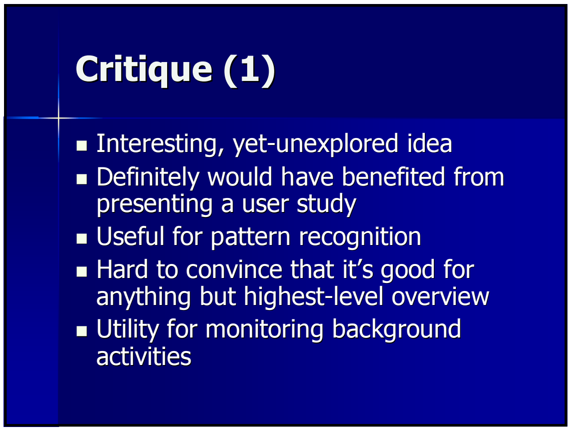
Review

- Interactive exploration of code
  - Modules, keywords
- Alternative to indentation, code style,..
- Use "audio landmarks" to mark important segments
- Recognize patterns when scrolling



Preparation

- Syntactic structure
- Stages
  - Entering a program
  - Compilation
- Loop example
  - Scope
  - scalability



Critique (1)

- Interesting, yet-unexplored idea
- Definitely would have benefited from presenting a user study
- Useful for pattern recognition
- Hard to convince that it's good for anything but highest-level overview
- Utility for monitoring background activities

Critique (2)

- Enhancement to visual, couldn’t replace
- No scalability
- Couldn’t follow execution real-time
- Workspace issues
- Real-life examples?

3D Representations for Software Visualization

Andrian Marcus, Louis Feng, Jonathan I. Maletic, 2003

Overview

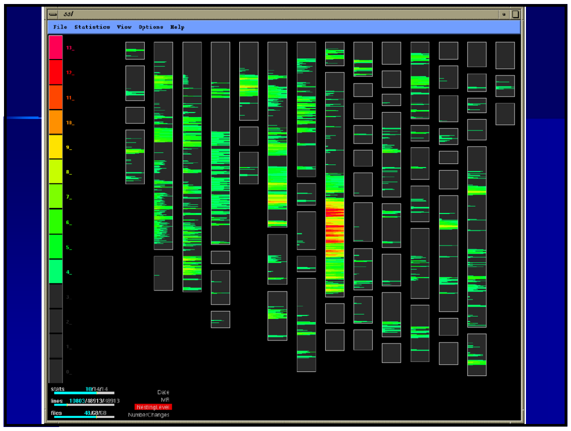
- Tool using 3D, texture, .. to represent multiple attributes in one view
- Visualization of large-scale software to assist in comprehension and analysis
- Categorize info to display important info more efficiently
- Visualization front-end, independent of source of data

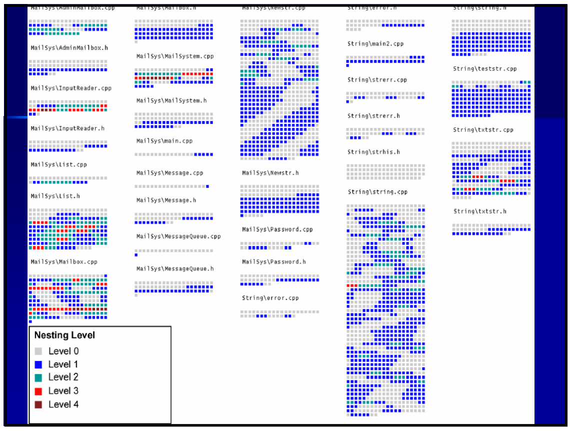
Dimensions of interest

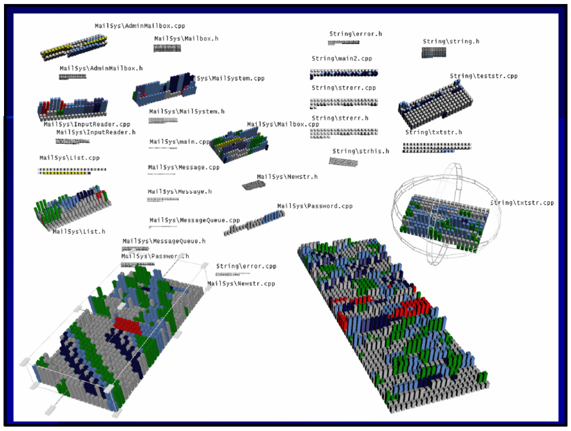
- Tasks – why visualize
- Audience – users
- Target – data source
- Representation – how to show data
- Medium – where displayed

Features (1)

- Separate visualization from data collection
- Manipulation on a per-element basis
- Users can develop own visualization metaphors based on tasks
- Function similar to another tool done 7 years prior







Features (2)

- Visual front-end – can be used with output of many analysis tools
- Certain elements only suitable for certain data types

Support for user needs

- Overview ***
- Zoom
- Filter
- Details-on-demand
- Relate ***
- History
- Extract

Critique

- Propose to develop a stereoscopic display – not practical?
- Visual elements only suitable for certain data – guidance to users?
- Core components designed as an application framework
  - Extend with new mappings and visual elements

Conclusions

- Applying visualization to various aspects of software engineering
- Various channels - visual, audio
- Building on existing ideas
- User studies and community acceptance?