Software Visualization

CS 695 / SWE 699: Programming Tools

Fall 2023

GEORGE MASON UNIVERSITY

Today

- Part 1 Lecture(~45 mins)
  - 10 min break
- Part 2: Tech Talks (30 mins)
  - Two tech talks
- Part 3: In-Class Activity(1 hour)

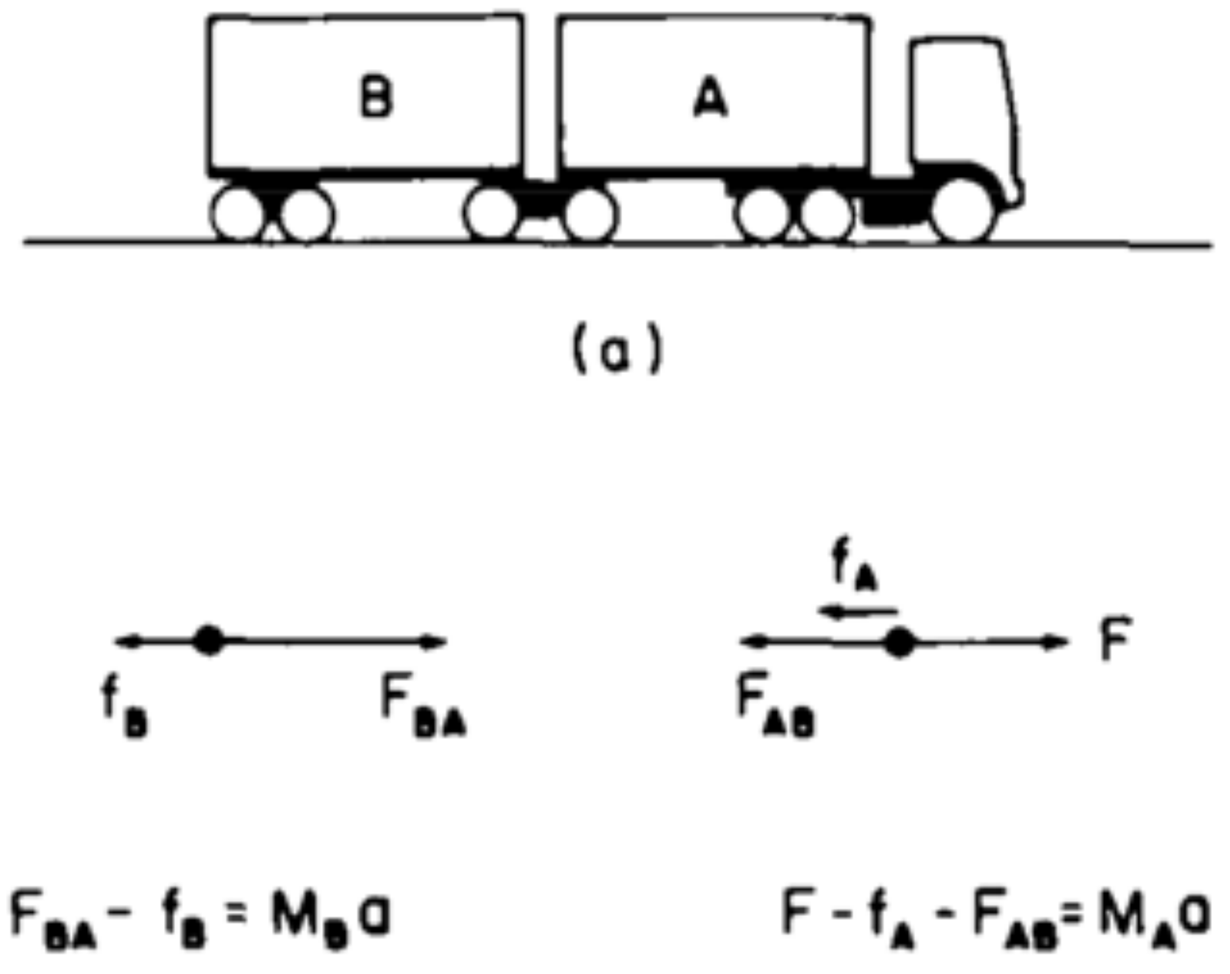
Logistics

- HW 4 due 11/29

Overview

- How can software visualization help?
- Different types of software visualization

Why a diagram is (sometimes) worth ten thousand words



- Diagrams can group together all information that is used together, thus avoiding large amounts of search for the elements needed to make a problem-solving inference.
- Diagrams typically use location to group information about a single element, avoiding the need to match symbolic labels.
- Diagrams automatically support a large number of perceptual inferences, which are extremely easy for humans

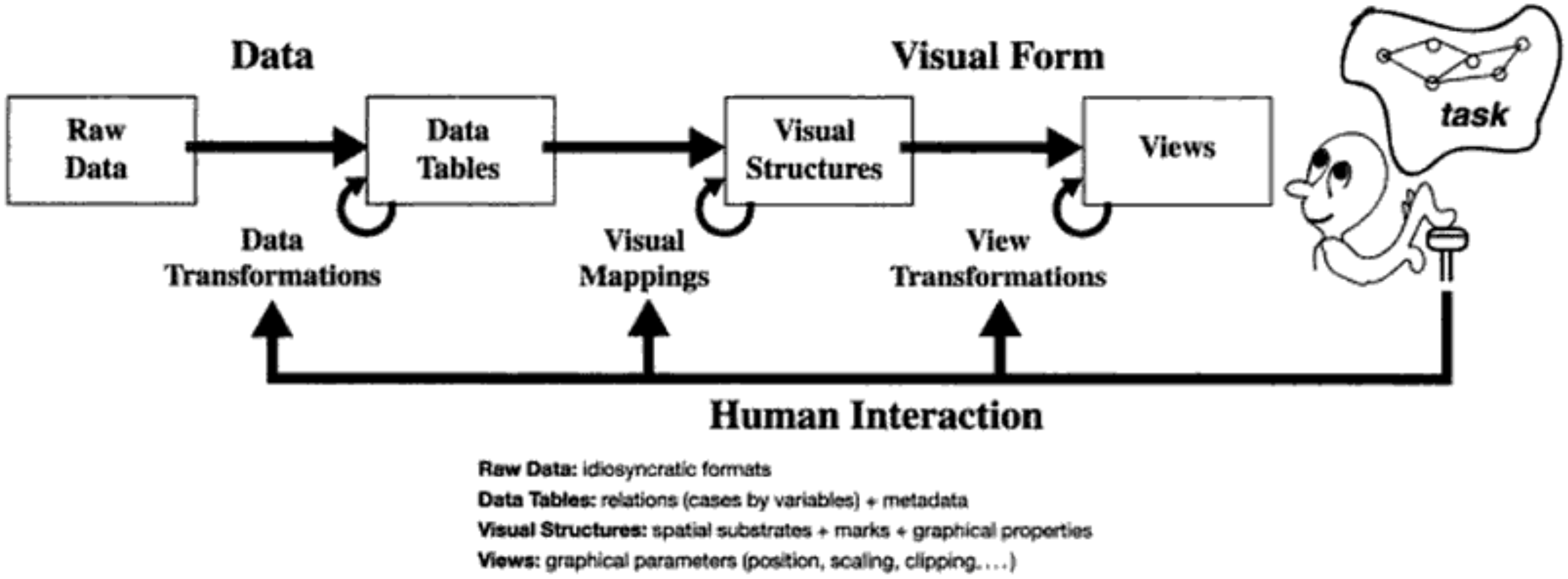
- Larkin & Simon, 1987, Cognitive Science 11, pp 65-99.

How information visualization amplifies cognition.

Increased Resources	
High-bandwidth hierarchical interaction	The human moving gaze system partitions limited channel capacity so that it combines high spatial resolution and wide aperture in sensing visual environments (Resnikoff, 1987).
Parallel perceptual processing	Some attributes of visualizations can be processed in parallel compared to text, which is aerial.
Offload work from cognitive to perceptual system	Some cognitive inferences done symbolically can be recoded into inferences done with simple perceptual operations (Larkin and Simon, 1987).
Expanded working memory	Visualizations can expand the working memory available for solving a problem (Norman, 1993).
Expanded storage of information	Visualizations can be used to store massive amounts of information in a quickly accessible form (e.g., maps).
Reduced Search	
Locality of processing	Visualizations group information used together, reducing search (Larkin and Simon, 1987).
High data density	Visualizations can often represent a large amount of data in a small space (Tufte, 1983).
Spatially indexed addressing	By grouping data about an object, visualizations can avoid symbolic labels (Larkin and Simon, 1987).
Enhanced Recognition of Patterns	
Recognition instead of recall	Recognizing information generated by a visualization is easier than recalling that information by the user.
Abstraction and aggregation	Visualizations simplify and organize information, supplying higher centers with aggregated forms of information through abstraction and selective omission (Card, Robertson, and Mackinlay, 1991; Resnikoff, 1987).
Visual schemata for organization	Visually organizing data by structural relationships (e.g., by time) enhances patterns.
Value, relationship, trend	Visualizations can be constructed to enhance patterns at all three levels (Bertin, 1977/1981).
Perceptual Inference	
Visual representations make some problems obvious	Visualizations can support a large number of perceptual inferences that are extremely easy for humans (Larkin and Simon, 1987).
Graphical computations	Visualizations can enable complex specialized graphical computations (Hutchins, 1996).
Perceptual Monitoring	Visualizations can allow for the monitoring of a large number of potential events if the display is organized so that these stand out by appearance or motion.
Manipulable Medium	Unlike static diagrams, visualizations can allow exploration of a space of parameter values and can amplify user operations.

S.K.Card, J.D.Mackinlay, B.Shneiderman, “Information Visualization”, Readings in Information Visualization: Using Vision to Think, Morgan Kaufman, Chapter 1.

Designing an information visualization



S.K.Card, J.D.Mackinlay, B.Shneiderman, “Information Visualization”, Readings in Information Visualization: Using Vision to Think, Morgan Kaufman, Chapter 1.

Tufte's principles of graphical excellence

- show the **data**
- induce the viewer to think about the substance rather than the methodology
- avoid distorting what the data have to say
- present **many** numbers in a small space
- make large data sets **coherent**
- encourage the eye to **compare** different pieces of data
- reveal data at several levels of detail, from overview to fine structure
- serve reasonable clear **purpose**: description, exploration, tabulation, decoration

Interactive visualizations

- Users often use iterative process of making **sense** of the data
  - Answers lead to new questions
- Interactivity helps user constantly change display of information to answer new questions
- Should offer visualization that offers best view of data moment to **moment** as desired view **changes**

How software visualizations may help

- Offer information that helps developers to answer questions
- Facilitate easier navigation between artifacts containing relevant information

Key questions for software visualization design

- Do you *really* need a visualization?
  - If you know the developer's question, can you answer it more simply *without* a visualization?

- **Anti-**pattern: show all the information, let user find patterns
  - In other domains (e.g., data analytics), visualization is a tool for data exploration and understanding dataset.
  - **Not true for SE:** developers want to complete tasks, finding patterns often not relevant

- How much context do you need?
  - More context —> more information to sort through
  - Less context —> more direct

Some popular forms of software visualizations

- Code
  - Iconographic representation of code text

- Algorithm & object structure visualizations
  - Depictions of data value changes over time
  - Runtime snapshots of object reference structure

- Module structure
  - Static views of module properties & dependencies (e.g., calls, references)

- Runtime software structure
  - Run time view of software elements that exist at runtime and messages sent (e.g., HTTP requests)

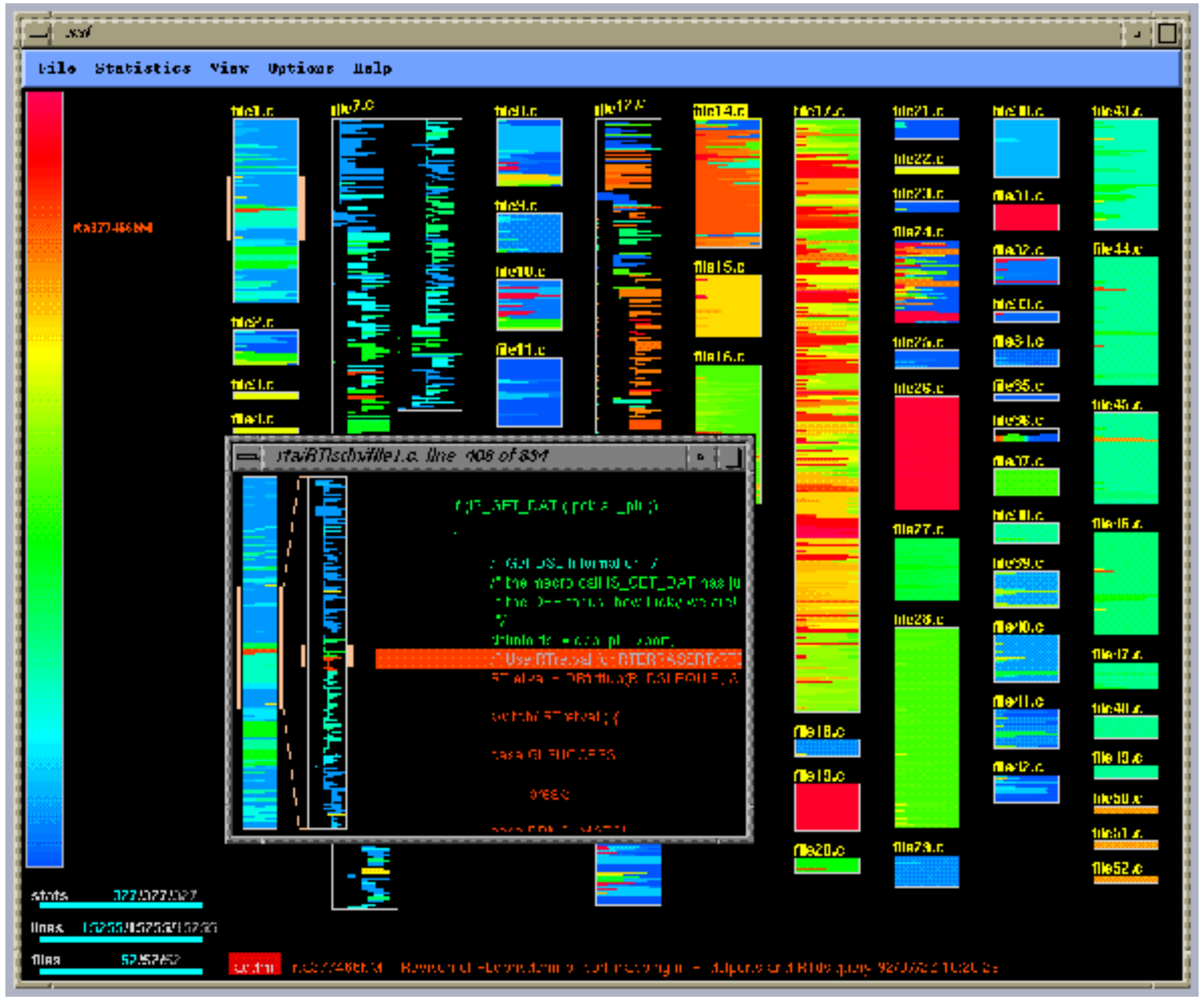
- Function calls
  - Dynamic and static depictions of function calls

Code visualizations

- Offer overview of source code
- Identify relevant sources lines matching some property
  - e.g., changed in a commit, passing a test, with a compiler warning
- Represent lines of iconagraphically
  - e.g., colored lines

SeeSoft

- AT&T Bell Labs [Eick, 1992]
- Visualization for performance
  - “Hot spots” in red
- Large volumes of code
  - Image is of 15,255 LOC
  - Up to 50,000 LOC
- Can indent like original source files
- Also, recently changed,
  - Version control systems
  - Static, dynamic analyses
- Interactive investigation



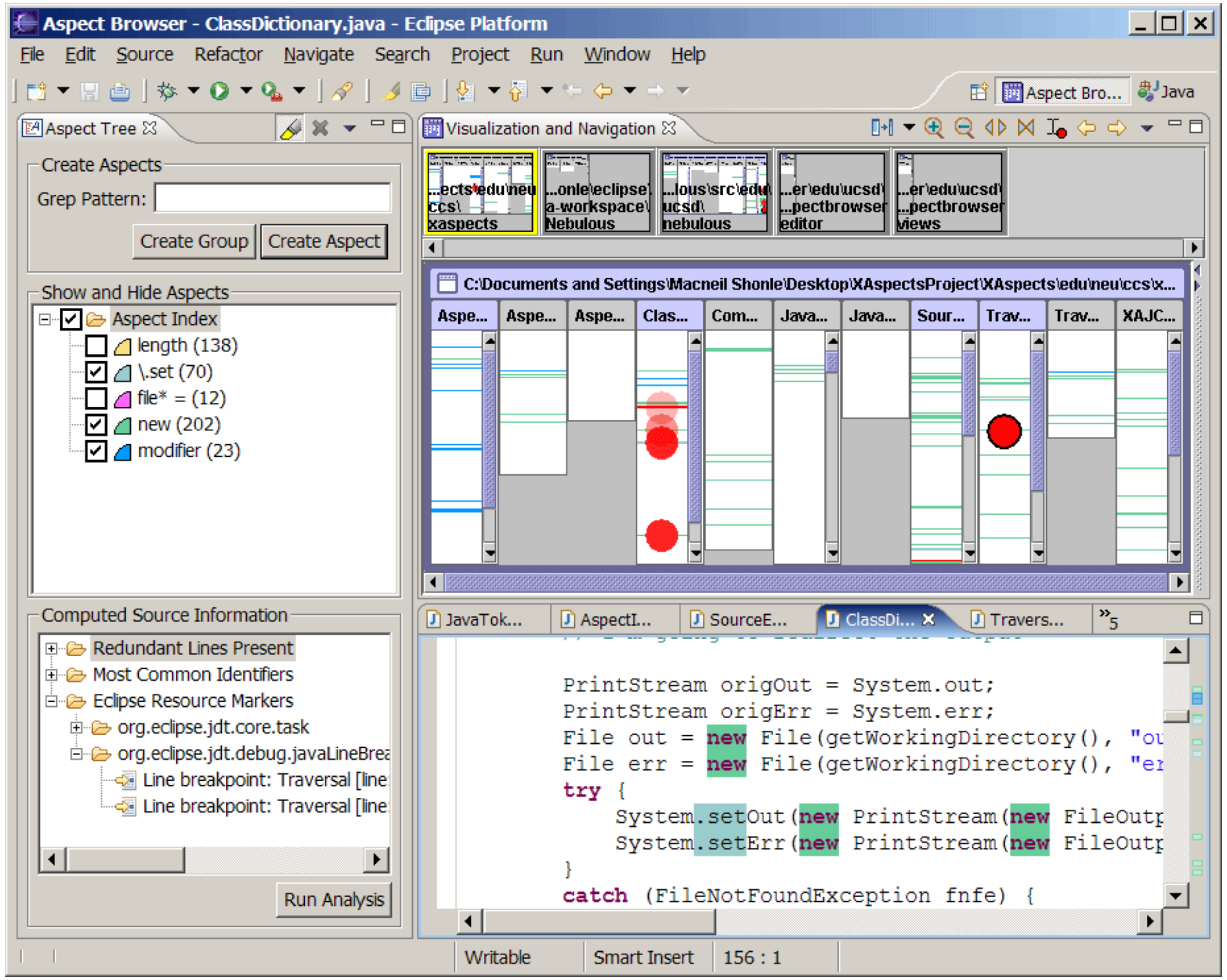
Stephen G. Eick, Joseph L. Steffen, and Eric E. Sumner, Jr.. 1992. Seesoft-A Tool for Visualizing Line Oriented Software Statistics. IEEE Trans. Softw. Eng. 18, 11 (November 1992), 957-968.

Tarantula

Color – code coverage
- Red – failed test case
- Green – past test case
- Yellow – hue is % of test cases passing

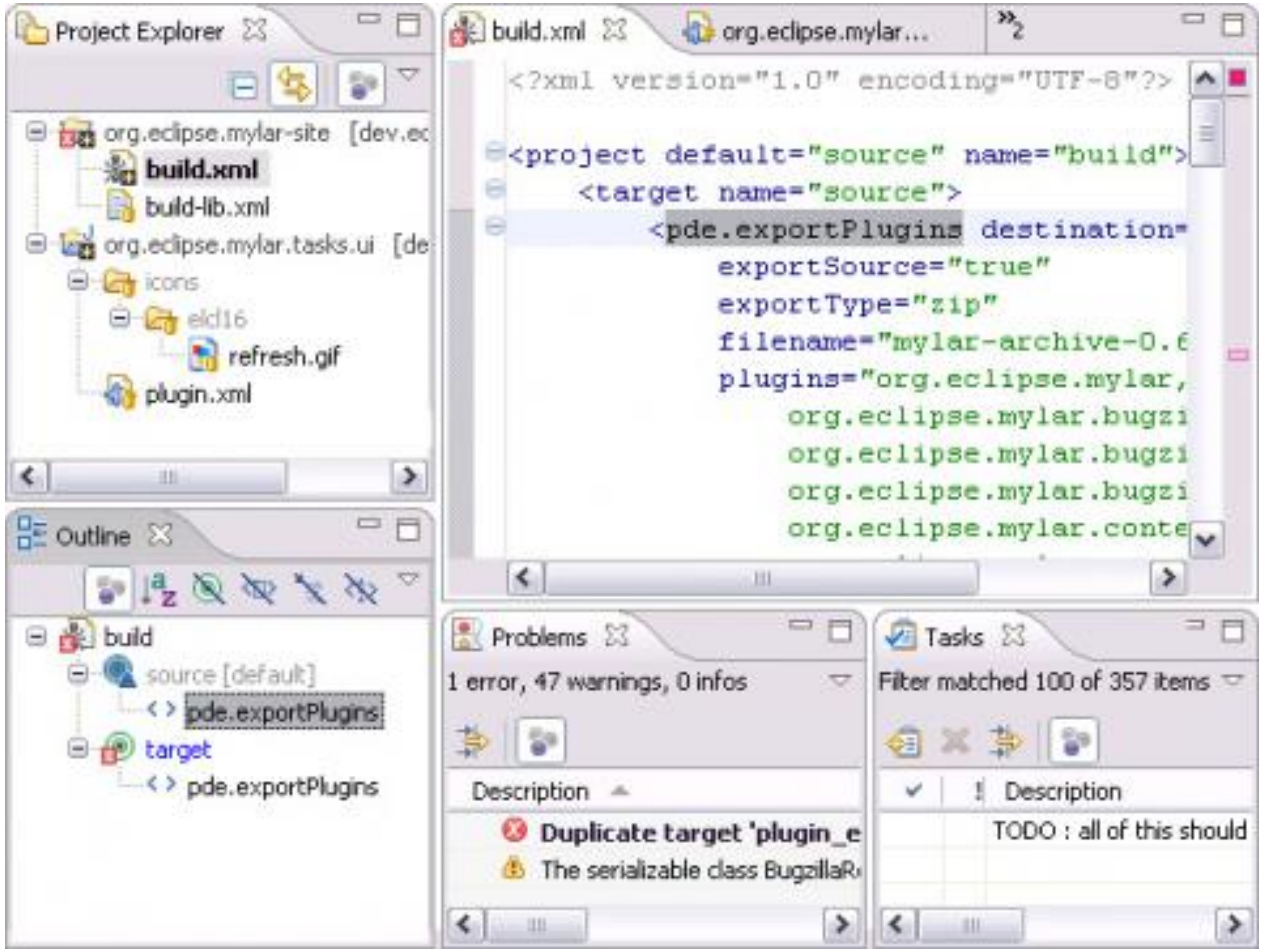
James A. Jones, Mary Jean Harrold, and John Stasko. 2002. Visualization of test information to assist fault localization. International Conference on Software Engineering (ICSE '02), 467-477.

AspectBrowser

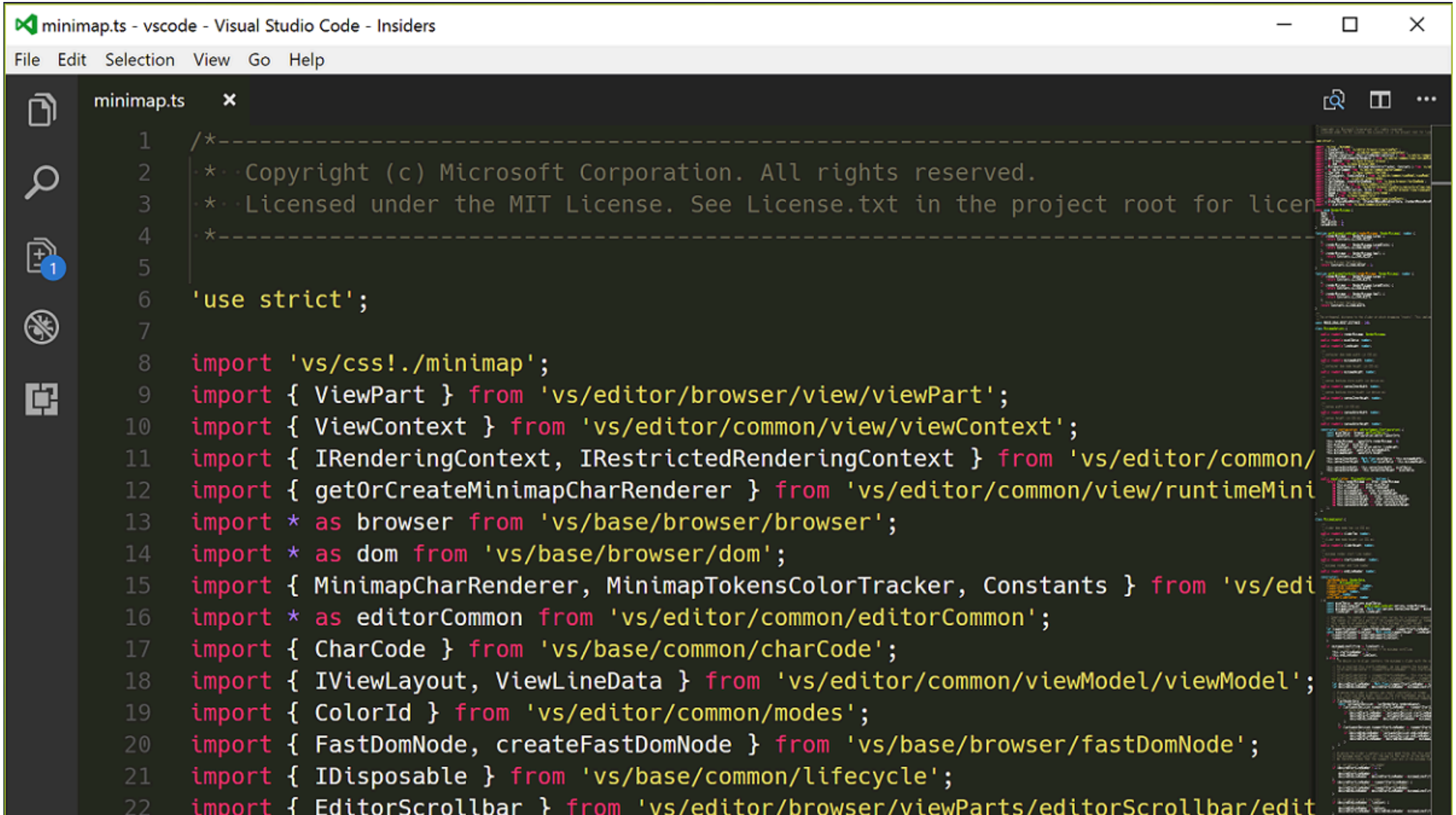


Macneil Shonle, Jonathan Neddenriep, and William Griswold. 2004. AspectBrowser for Eclipse: a case study in plug-in retargeting. In Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange (eclipse '04). ACM, New York, NY, USA, 78-82.

Industry Use: Eclipse Markers



Industry use: Visual Studio Code Minimap



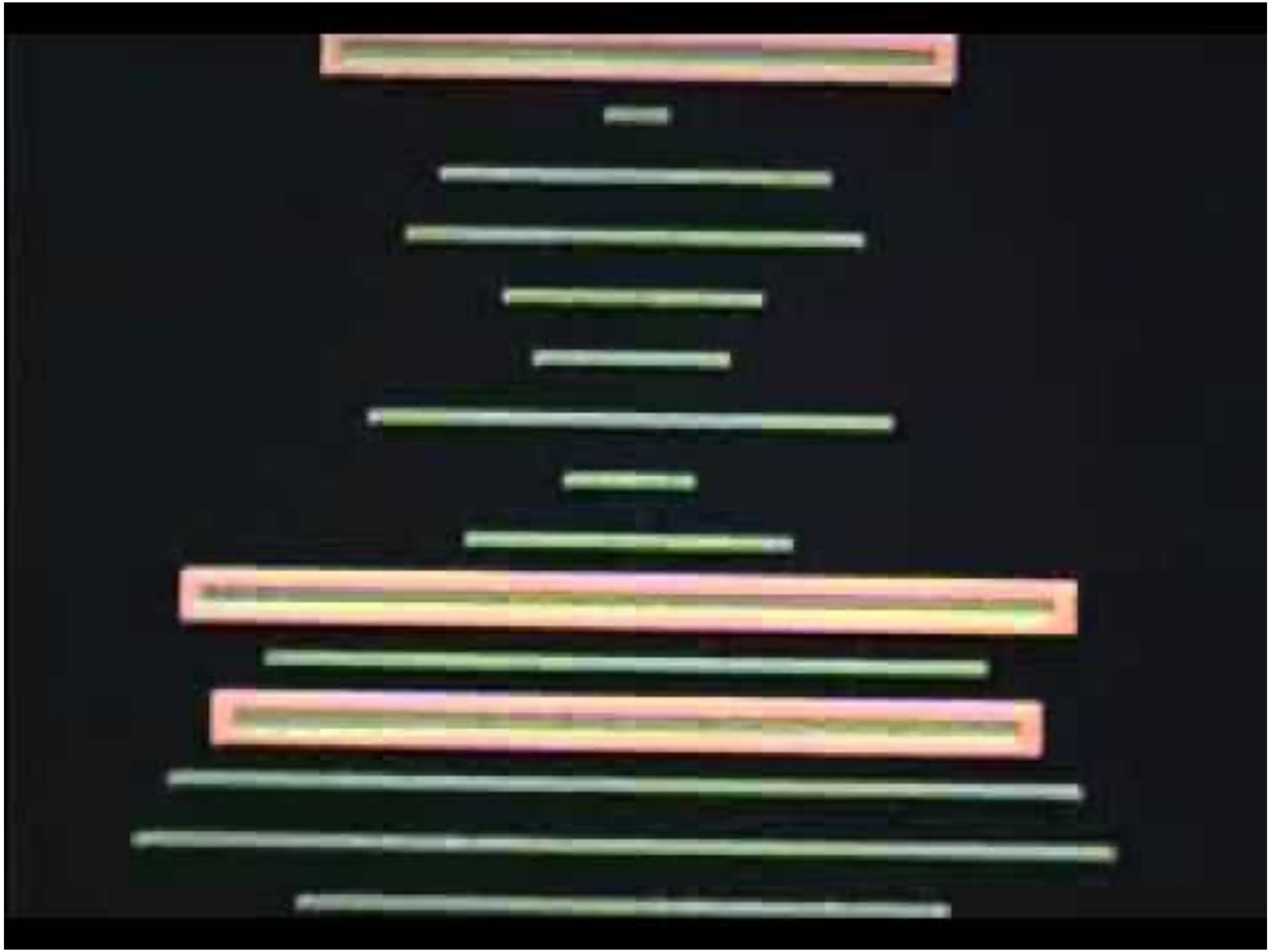
Algorithm & object structure visualizations

- Depict runtime state at a snapshot or over time
  - e.g., elements in a collection, numeric values

- Often focused on teaching basic algorithms (e.g., sorting algorithms, linked list insertion)

(Section adapted from Software Visualization, Lecture by Brad A. Myers, Spring 2011)

Sorting out Sorting



Incense

First to automatically create viz. of data structures

Produce pictures "like you might drawn them on a blackboard"

Goal: help with debugging

Figure 14.
ARRAY [1..4] OF POINTER with two POINTERs referring to the same value.

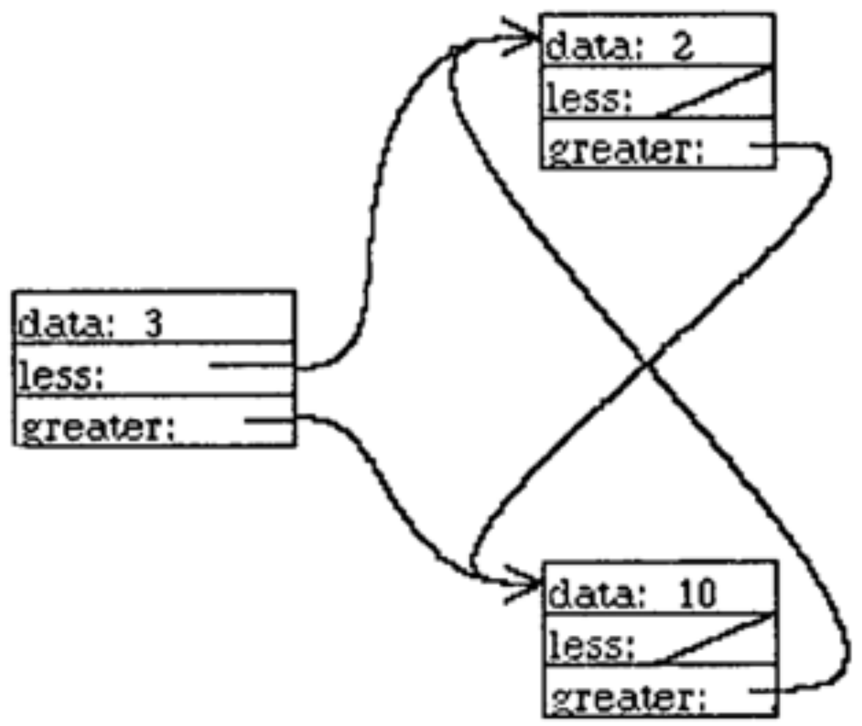


Figure 15.
This erroneous tree structure demonstrates that a pointer to previously displayed object does not generate a new copy. The second arrow is drawn to the first occurrence.

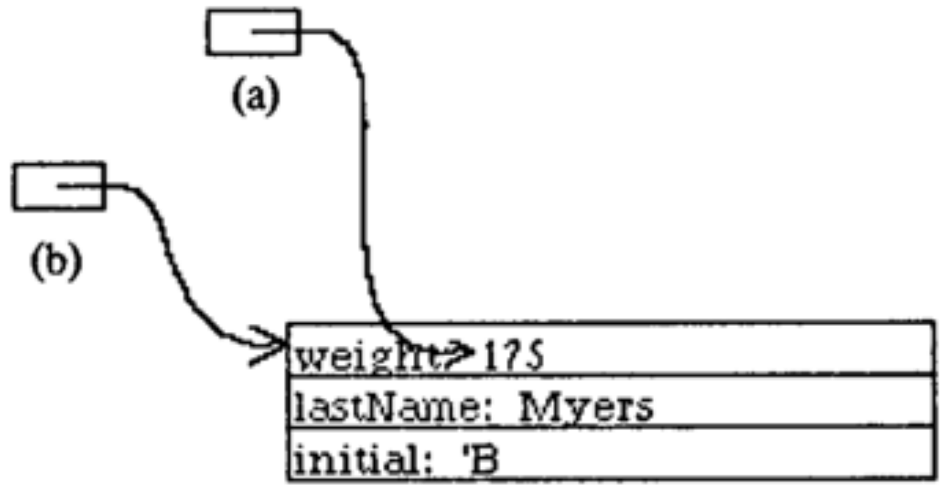


Figure 16.
Pointer to value inside a record (a) does not get confused with a pointer to the record itself (b).

Figure 17.
Incense display for
RECORD [int: INTEGER, p1: POINTER TO CARDINAL].

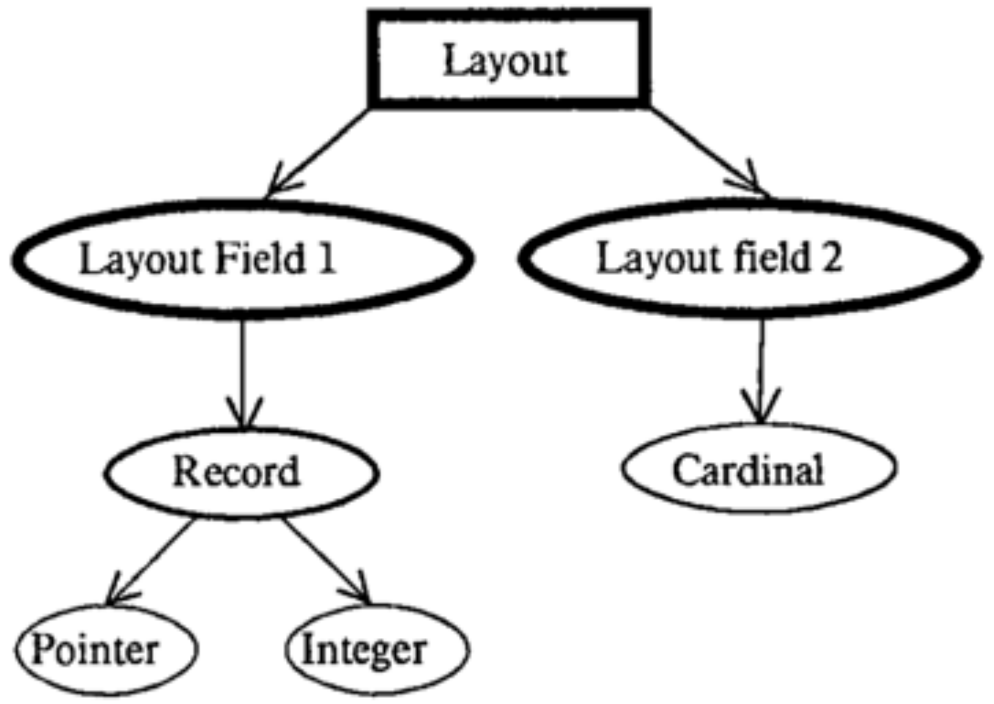


Figure 18.
Artist hierarchy that would be created for:
rec: RECORD [p1: POINTER TO CARDINAL, int: INTEGER];
(This figure was not created by Incense).

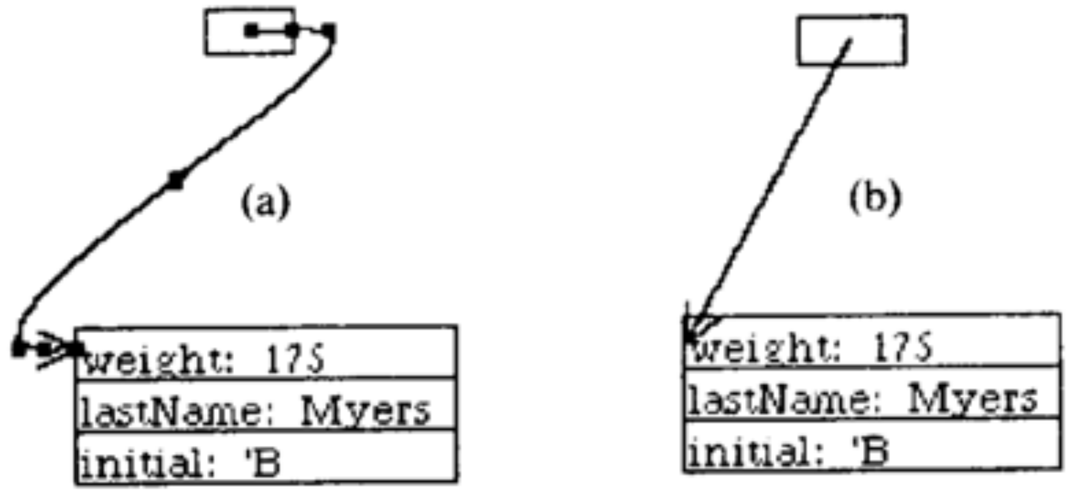
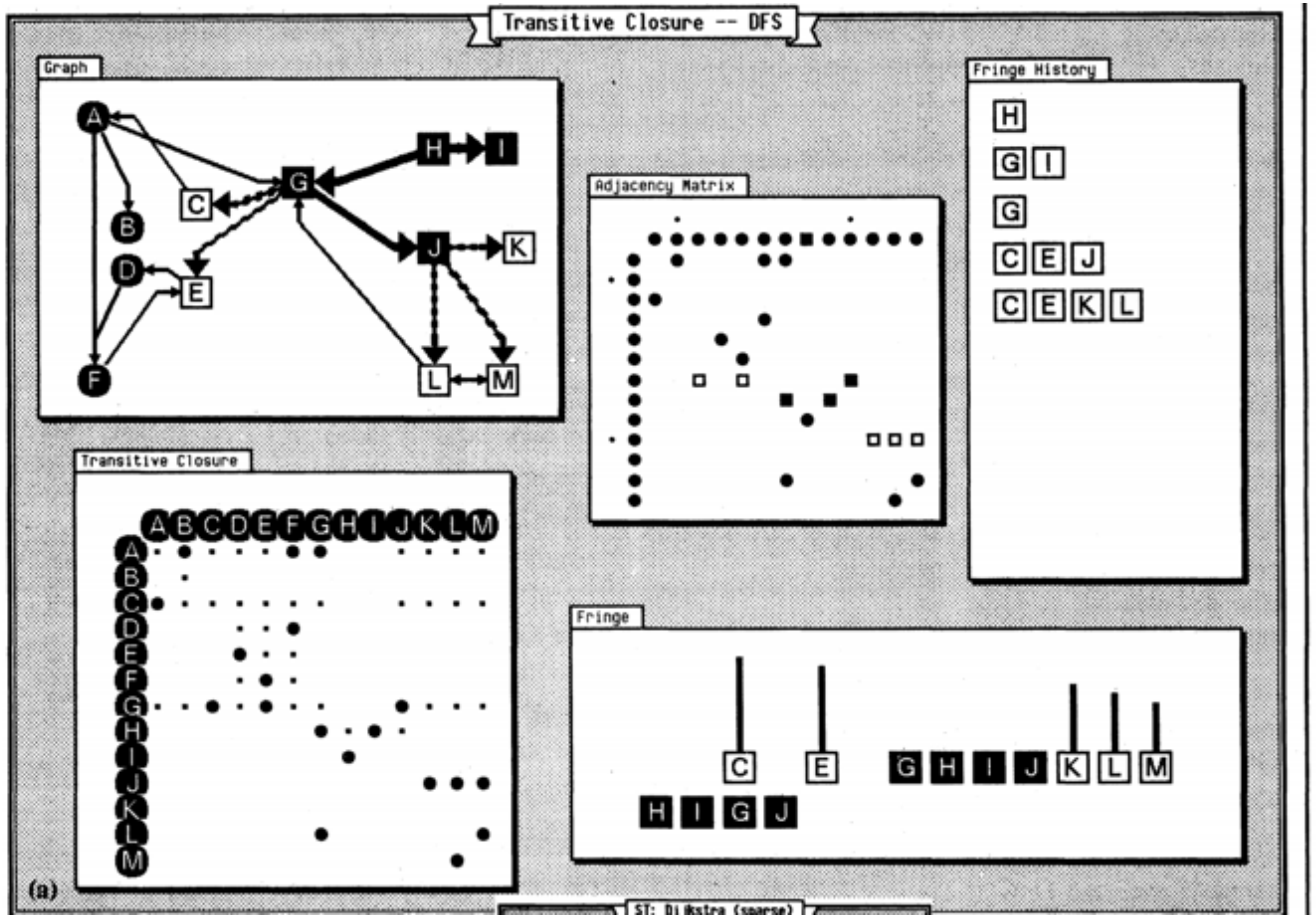


Figure 19.
Demonstration of the advantage of curved lines used in Incense (a) over straight lines (b). The control points used to specify the spline are shown as black squares in (a).

Brad A. Myers. 1983. INCENSE: A system for displaying data structures. Conference on Computer graphics and interactive techniques (SIGGRAPH '83),115-125.

Brown University Algorithm Simulator and Animator (BALSA)

- Major interactive integrated system
  - Extensively used for teaching at Brown Univ.
  - Lots of algorithms visualized
  - Architecture for attaching the graphics with code
    - Still required significant programming for each viz.
  - Marc followed up with Zeus ('91) at DEC SRC



Marc H. Brown and Robert Sedgewick. Techniques for Algorithm Animation. IEEE Software, 1985.

Transition-based Animation Generation (TANGO)

- John Stasko PhD thesis at Brown Univ. (1990)
- Smooth animations between states
  - Paths & transitions
- Make it easier to author algorithm visualizations
  - Events inserted into the code tied to animations

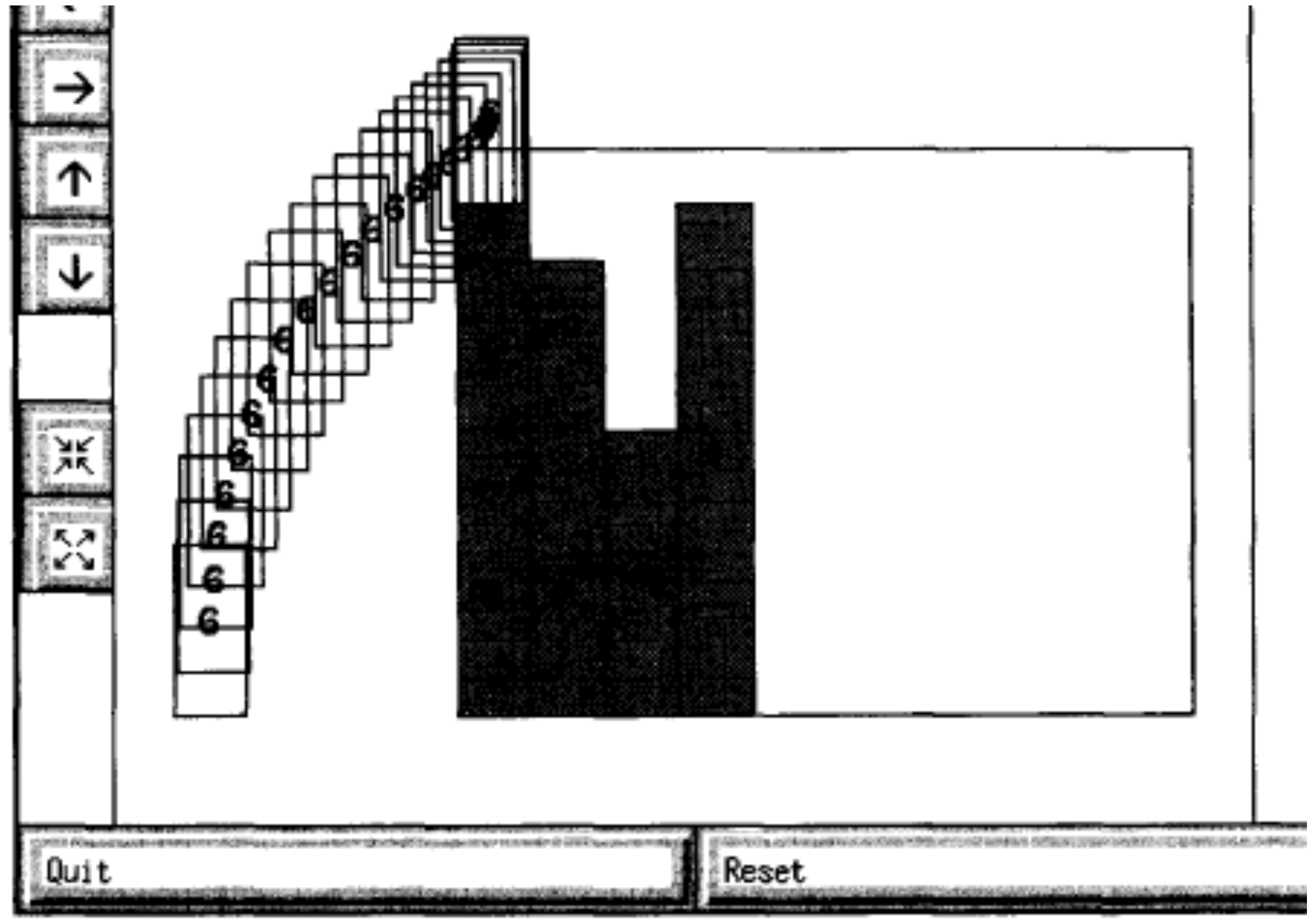


Figure 9. Superimposed sequence of frames from the bin-packing animation.

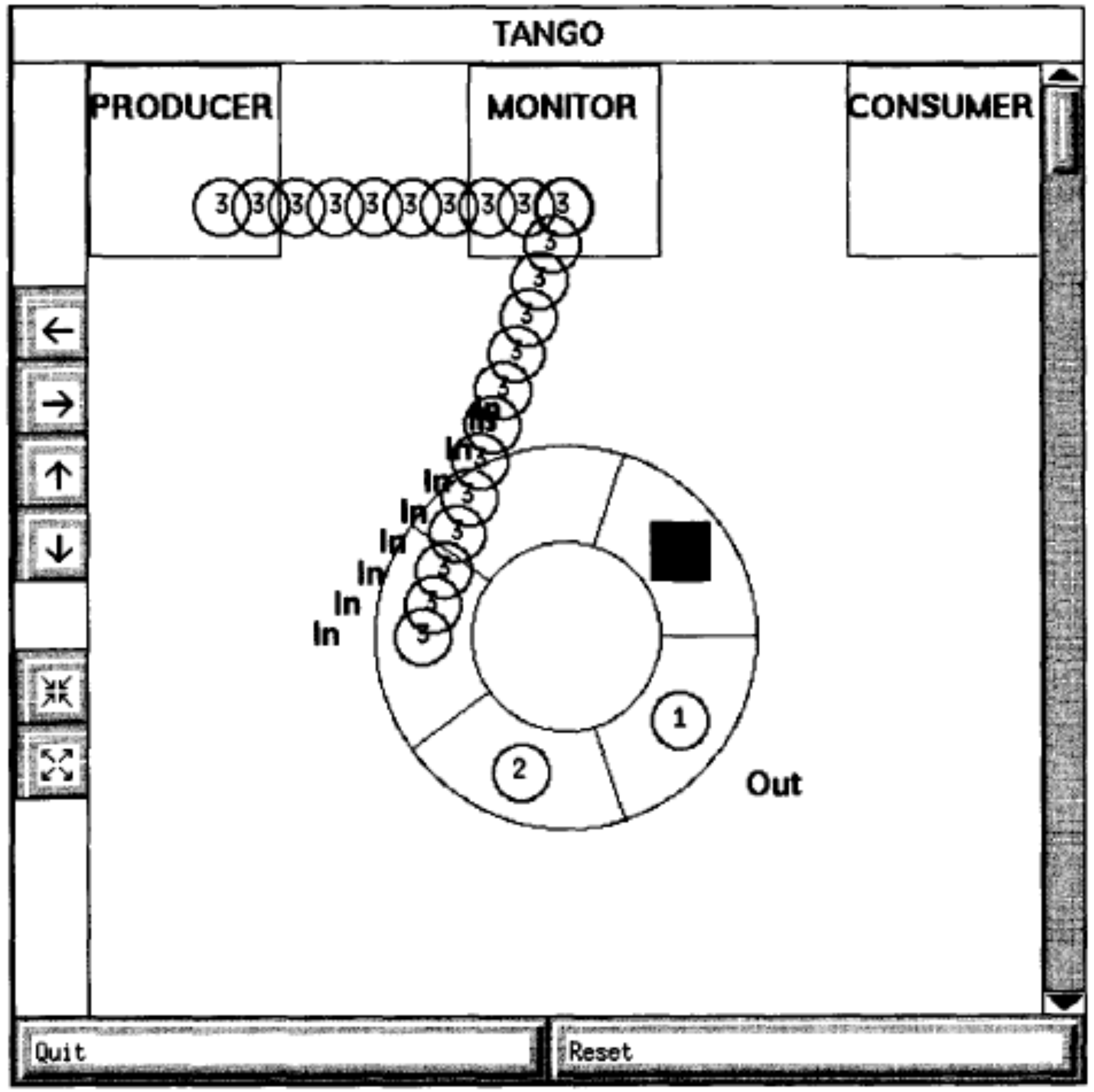
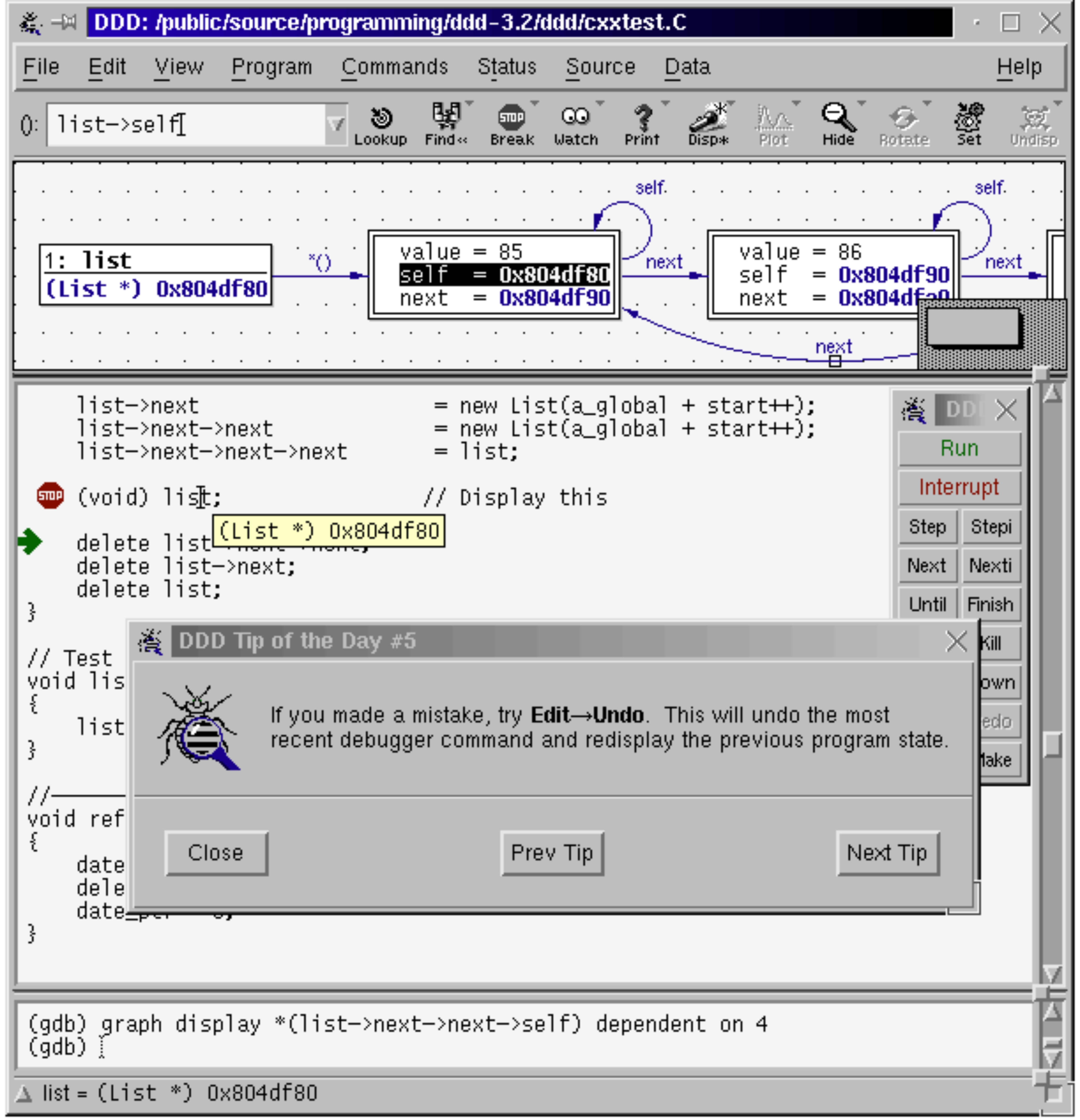


Figure 2. Tango animation of a producer-consumer ring buffer.

J. T. Stasko, "Tango: a framework and system for algorithm animation," in Computer, vol. 23, no. 9, pp. 27-39, Sept. 1990.

Data Display Debugger

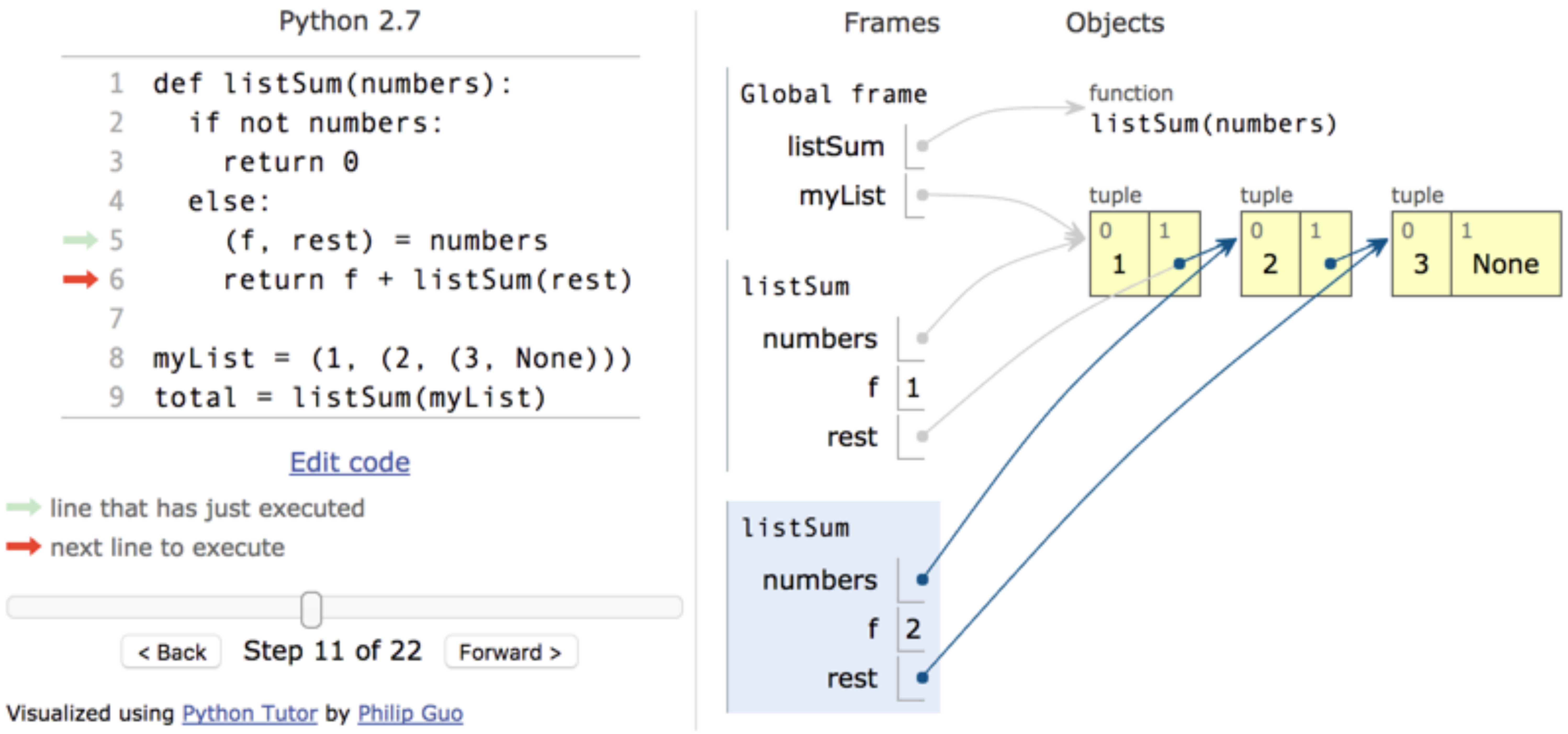


https://www.gnu.org/software/ddd/

Andreas Zeller and Dorothea Lütkehaus. 1996. DDD—a free graphical front-end for UNIX debuggers. SIGPLAN Not. 31, 1 (January 1996), 22-27.

PythonTutor

http://pythontutor.com/



Over 2.5 million people in over 180 countries have used Python Tutor to visualize over 20 million pieces of code

Philip J. Guo. Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education. In Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE), March 2013.

Module Views

- Depict static structure of modules (e.g., files, folders, packages)

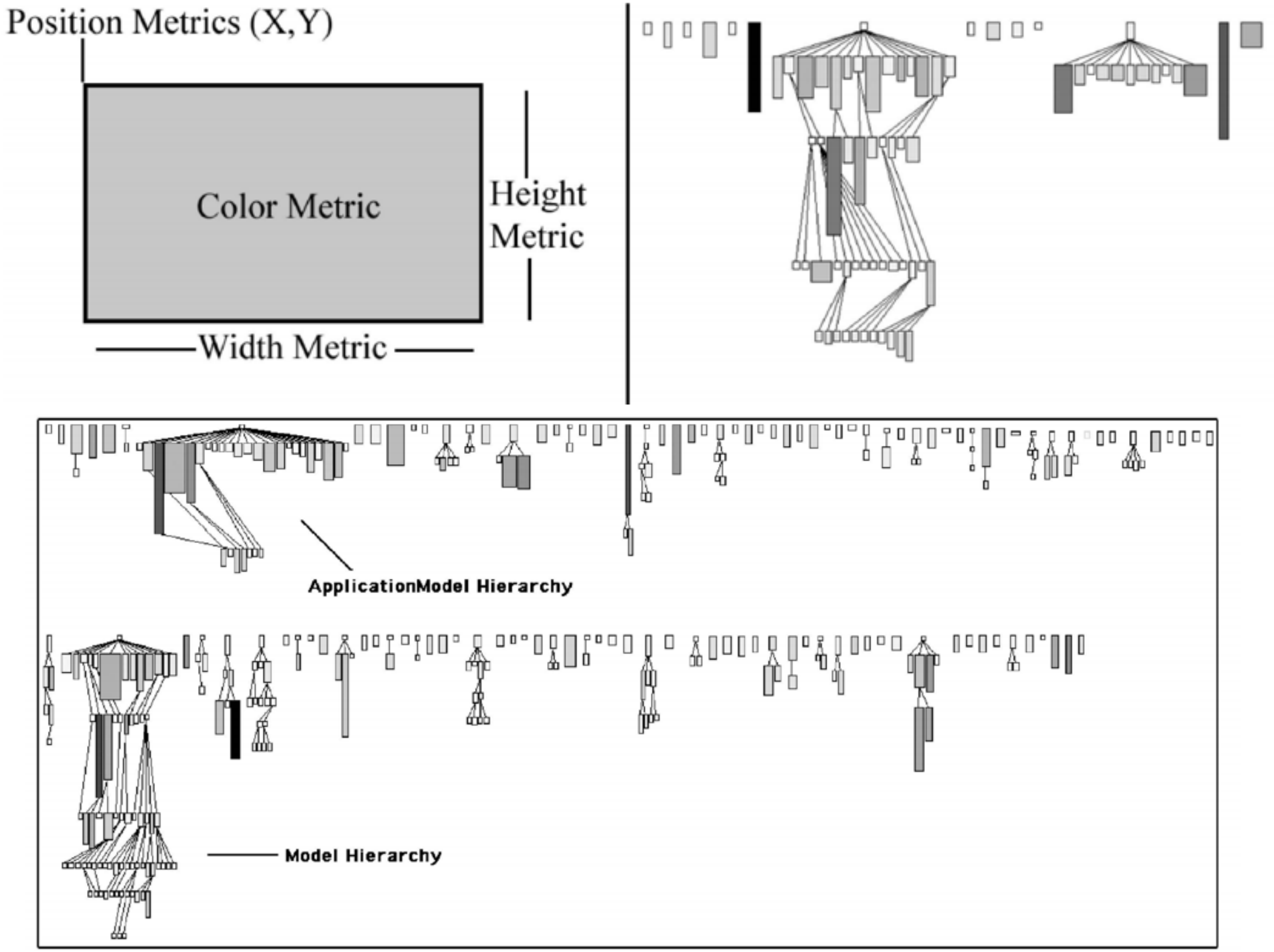
- Often depicts dependencies between modules

- Focus on reverse engineering tasks, refactoring tasks, other architecture related tasks

SHriMP

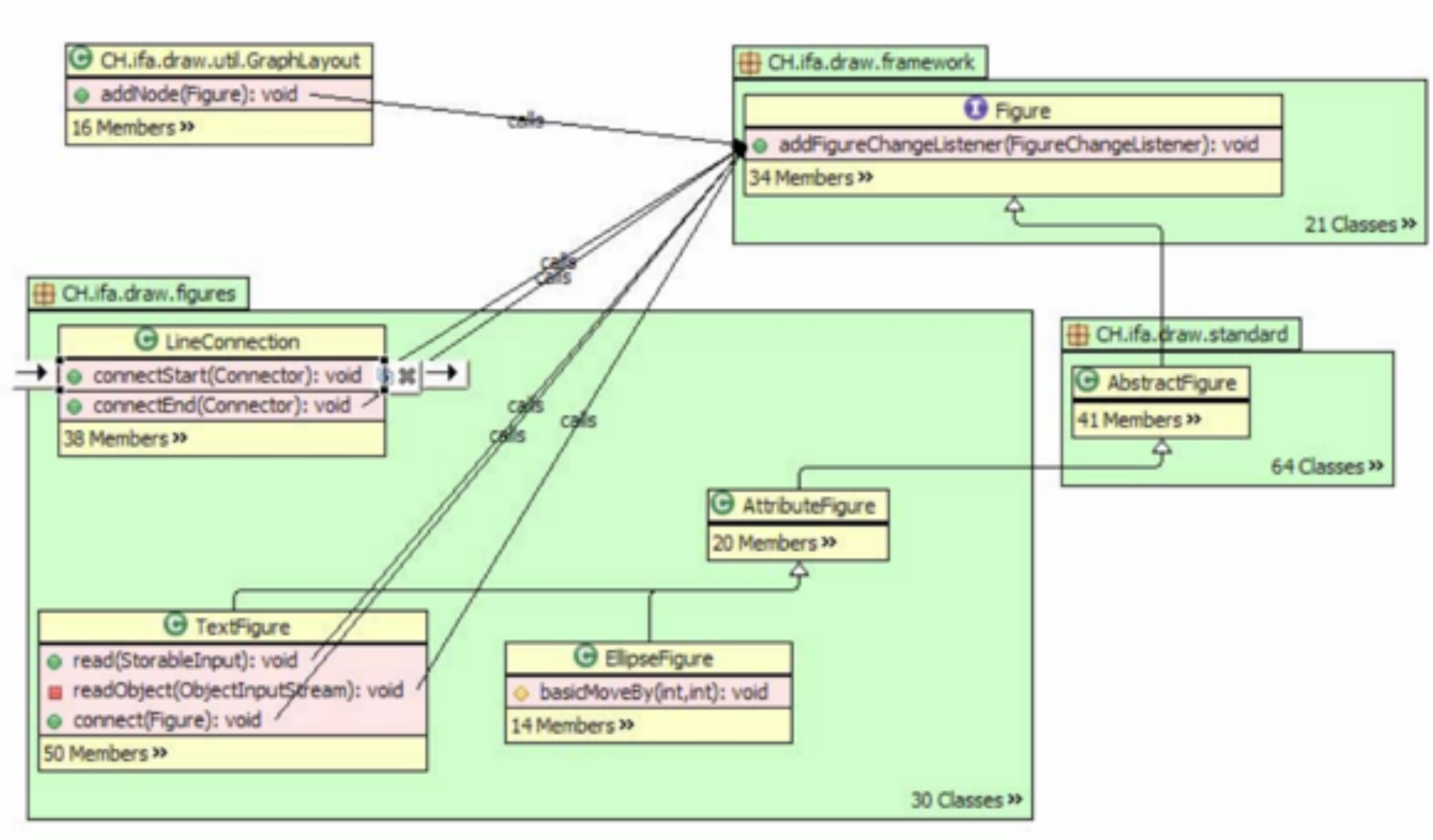
M.-A.D Storey, F.D Fracchia, H.A Müller, Cognitive design elements to support the construction of a mental model during software exploration, Journal of Systems and Software, Volume 44, Issue 3, January 1999, Pages 171-185.

Code Crawler (Polymetric Views)



Michele Lanza and Stéphane Ducasse. 2003. Polymetric Views-A Lightweight Visual Approach to Reverse Engineering. IEEE Trans. Softw. Eng. 29, 9 (September 2003), 782-795.

Relo



Vineet Sinha, David Karger, and Rob Miller. 2006. Relo: Helping Users Manage Context during Interactive Exploratory Visualization of Large Codebases. In Proceedings of the Visual Languages and Human-Centric Computing (VLHCC ’06), 187-194.

Lattix (Design Structure Matrices)

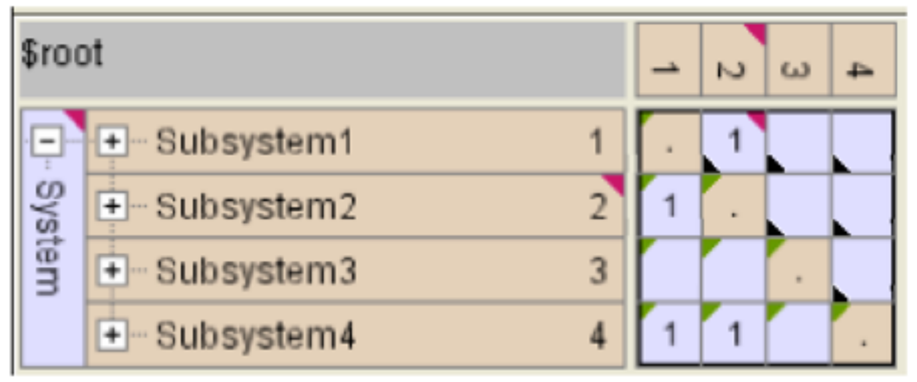


Figure 12: DSM with Rule View

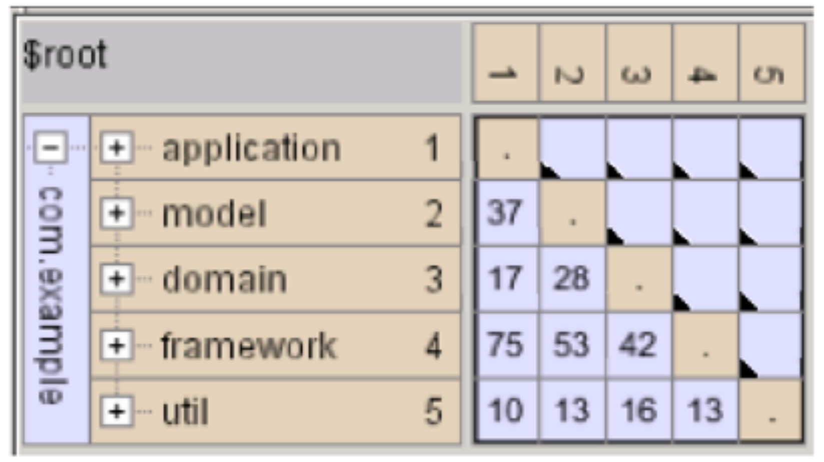


Figure 13: Design Rules for a Layered System

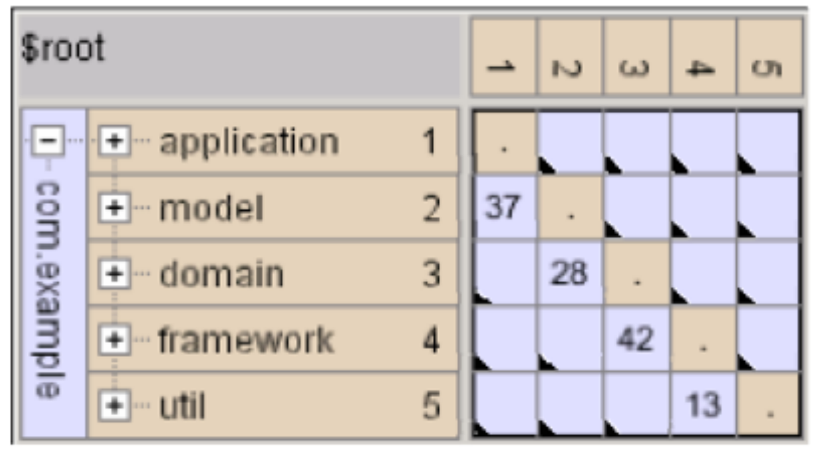
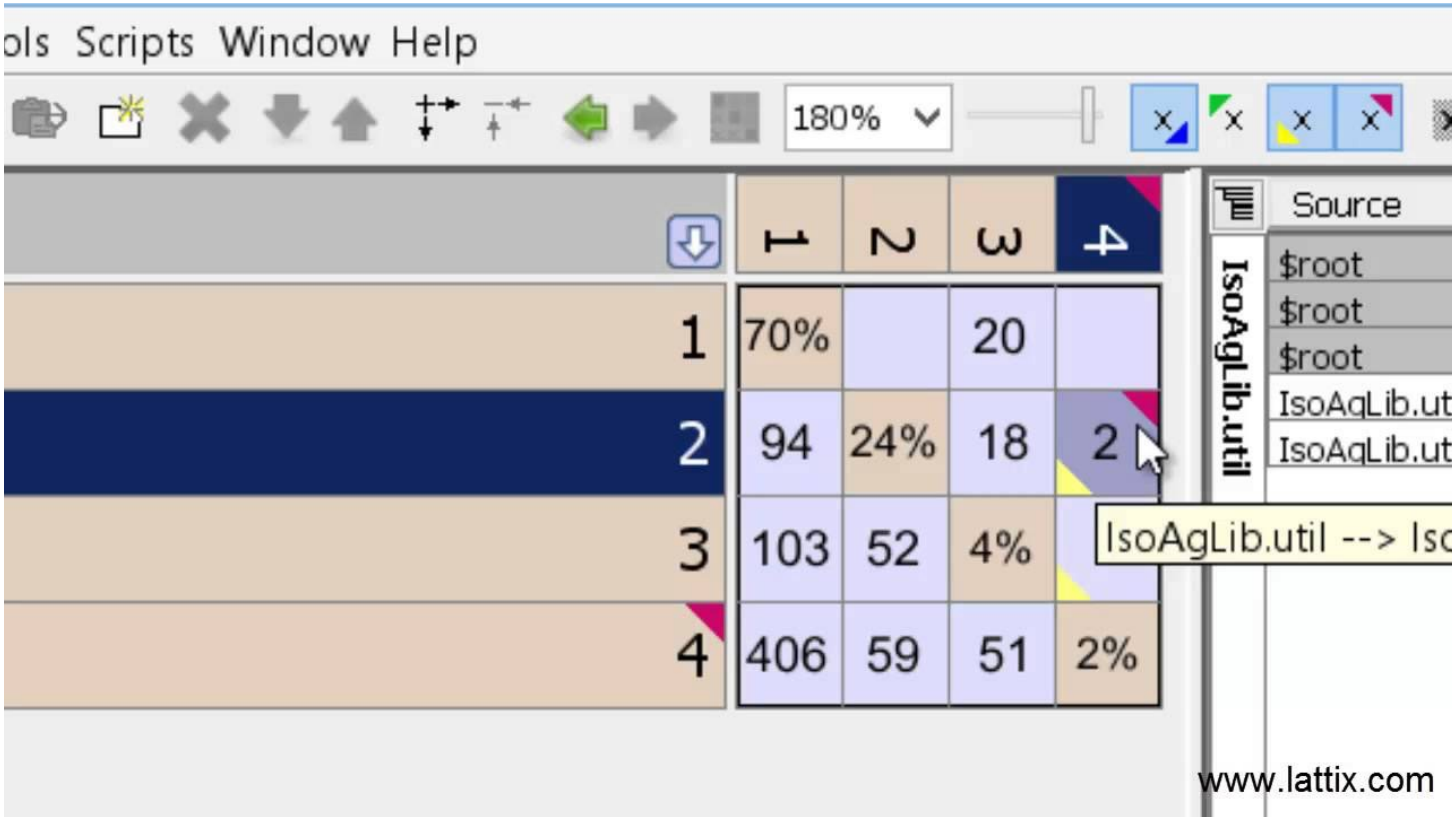


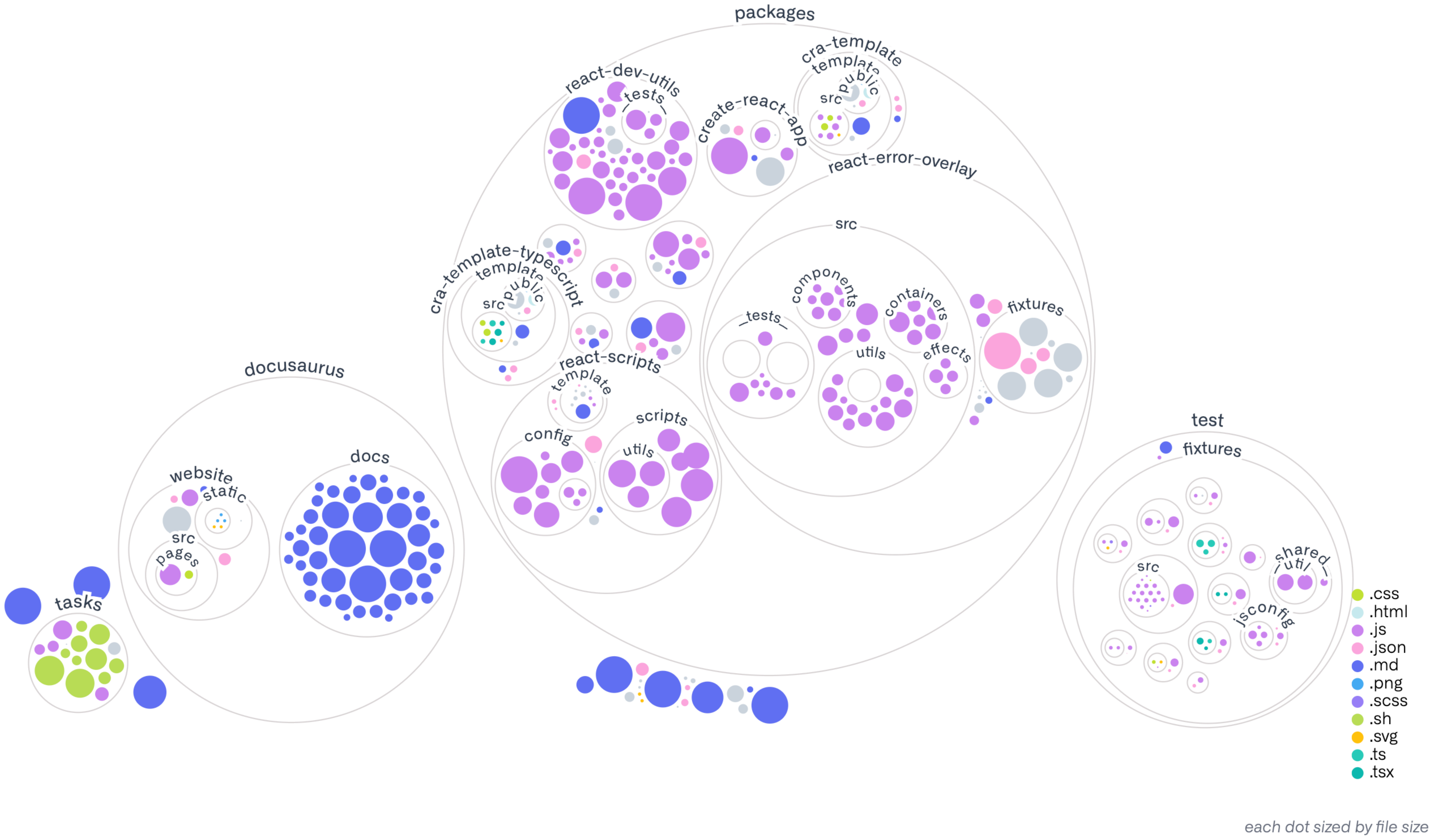
Figure 14: Design Rules for a Strictly Layered System

Figure 15: Design Rules for Independent Components



Neeraj Sangal, Ev Jordan, Vineet Sinha, and Daniel Jackson. 2005. Using dependency models to manage complex software architecture. Conference on Object-oriented programming, systems, languages, and applications (OOPSLA '05), 167-176.

GitHub Repo-Visualization



https://githubnext.com/projects/repo-visualization

CodeSee

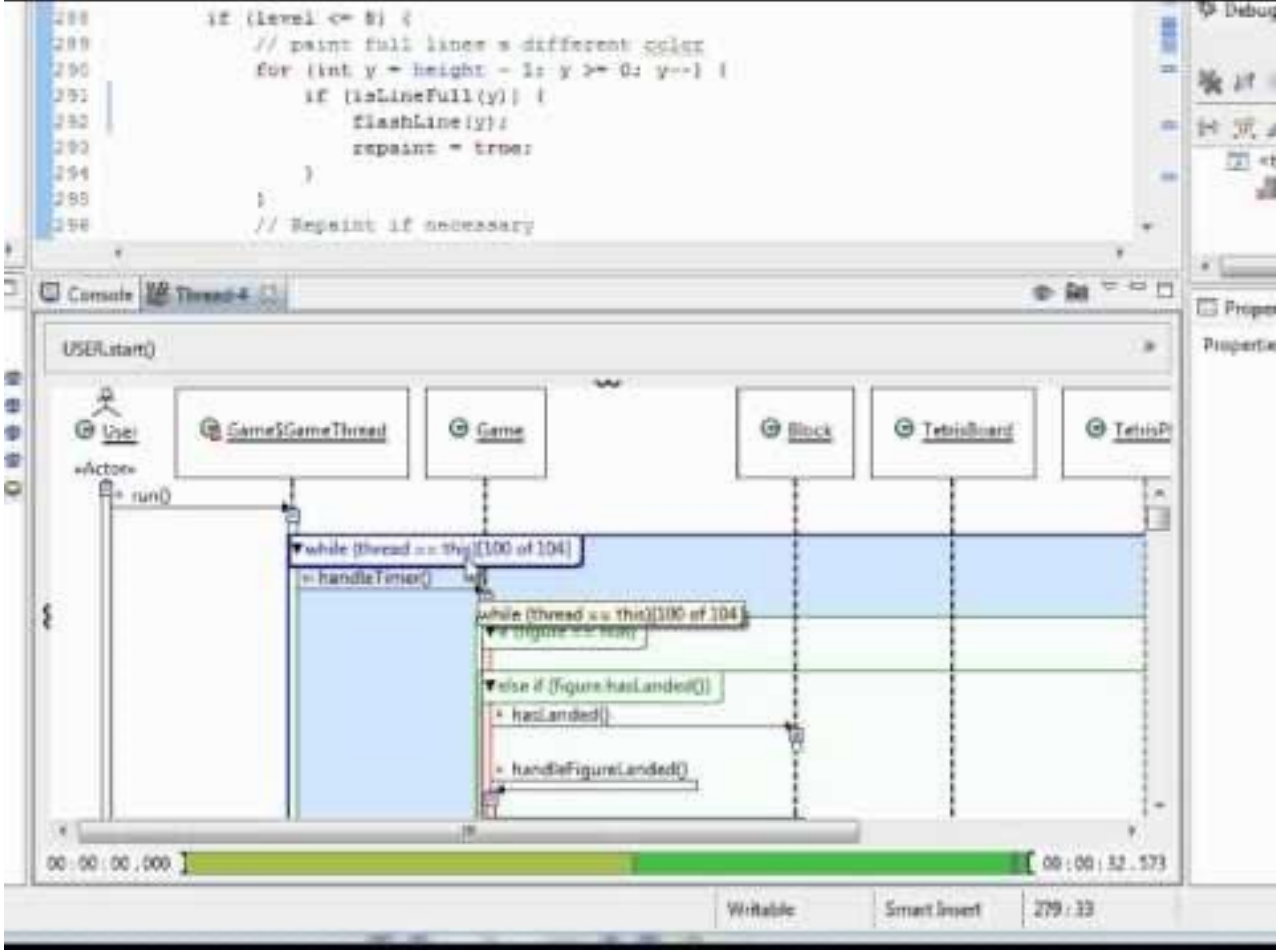
Runtime Structure Views


LOGO
← 200 px →
BANNER ← 500
NAVIGATION ← 800 px
CONTENT 1
CONTENT 2
500 px
FOOTER ← 800px

Function calls

- Depict function invocations
- Could be runtime view (specific execution) or static view (all possible executions)
- Many decisions about what to show & how to show it
  - Code centric? Timeline centric?
  - Show all functions? Show some functions? Which ones?
  - What information about functions to depict? Order, time, asycnonicity, …

Diver

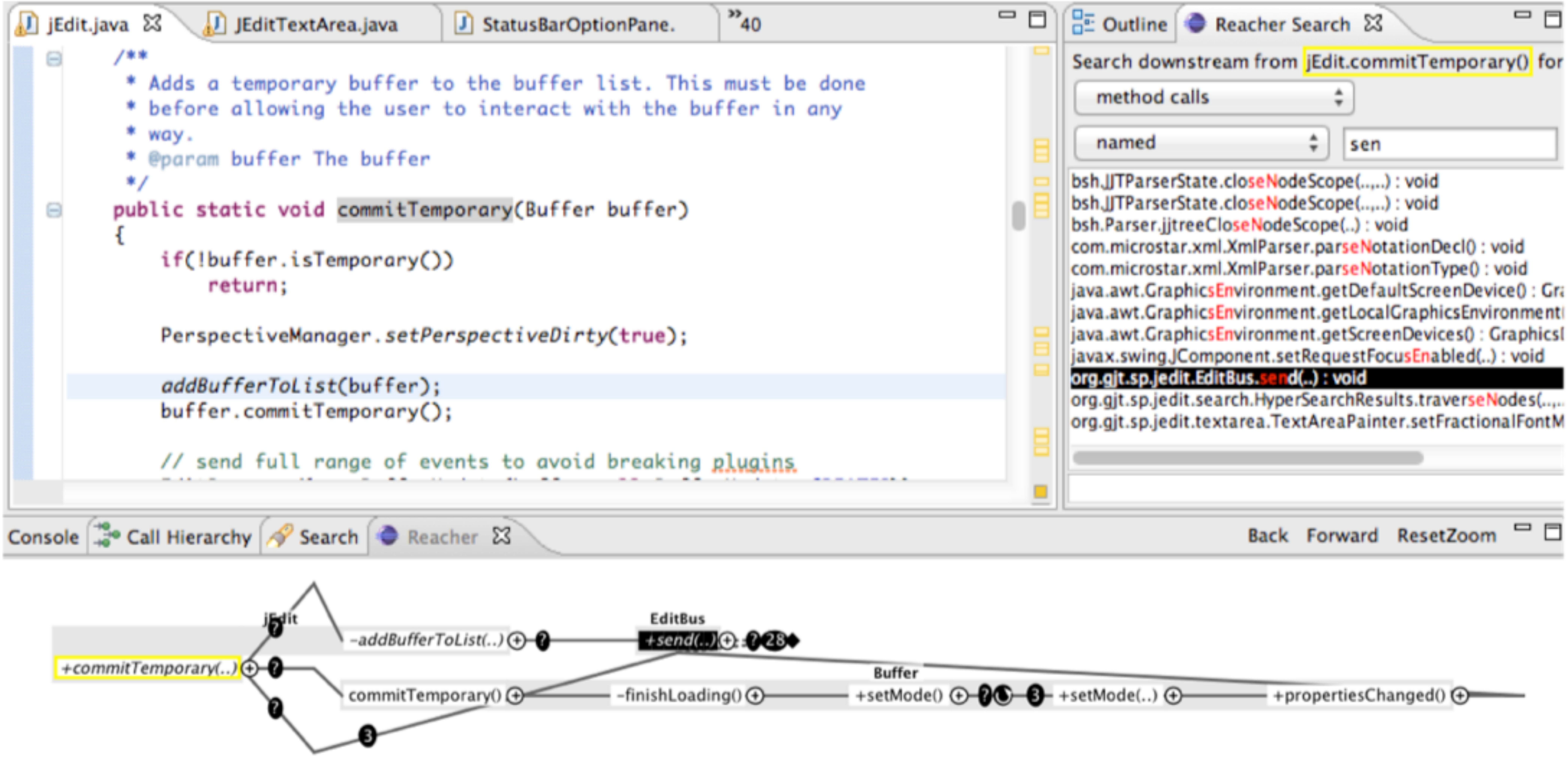


Del Myers and Margaret-Anne Storey. 2010. Using dynamic analysis to create trace-focused user interfaces for IDEs. In Proceedings of the eighteenth ACM SIGSOFT international symposium o
Foundations of software engineering (FSE '10). ACM, New York, NY, USA, 367-368.

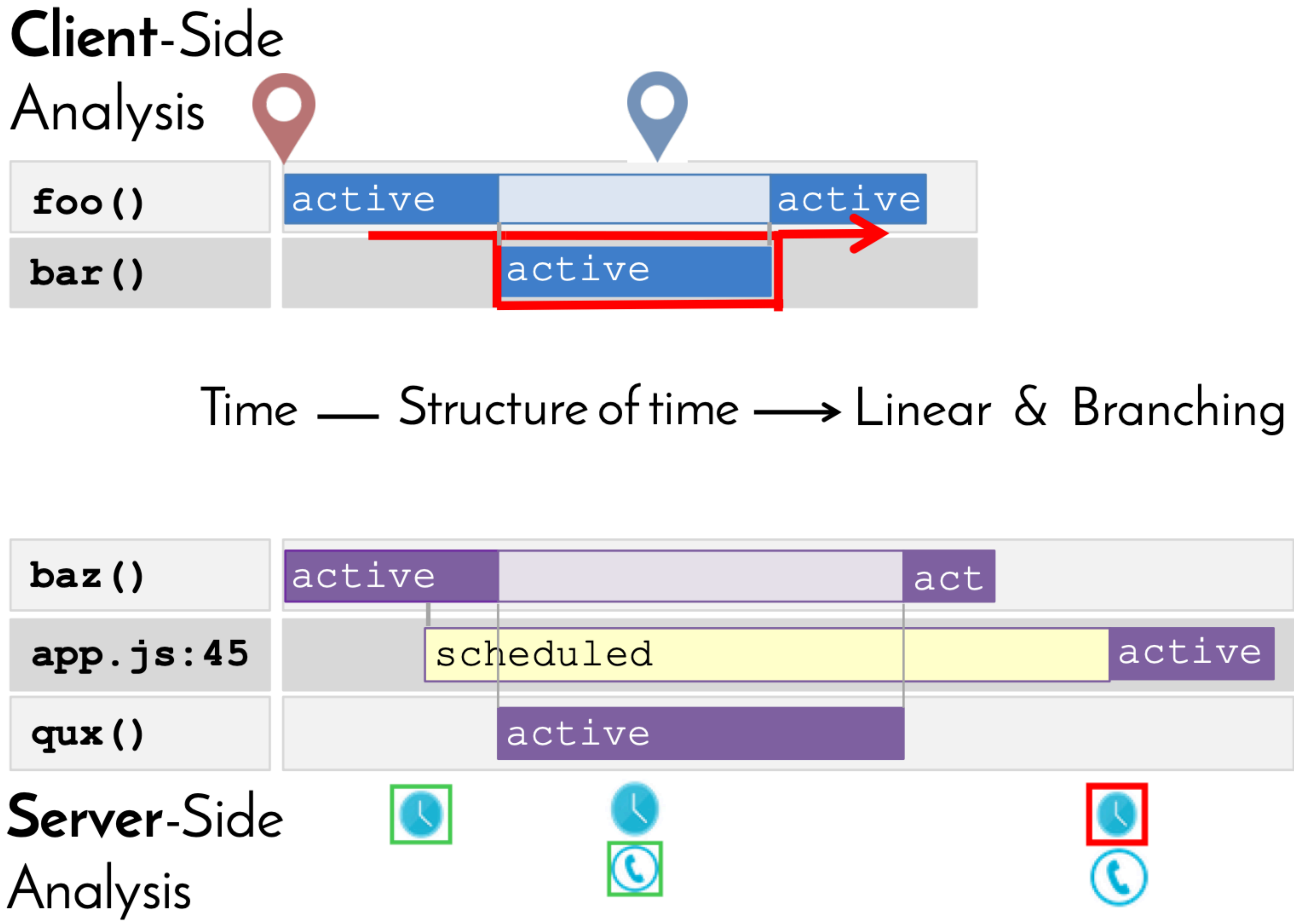
Theseus

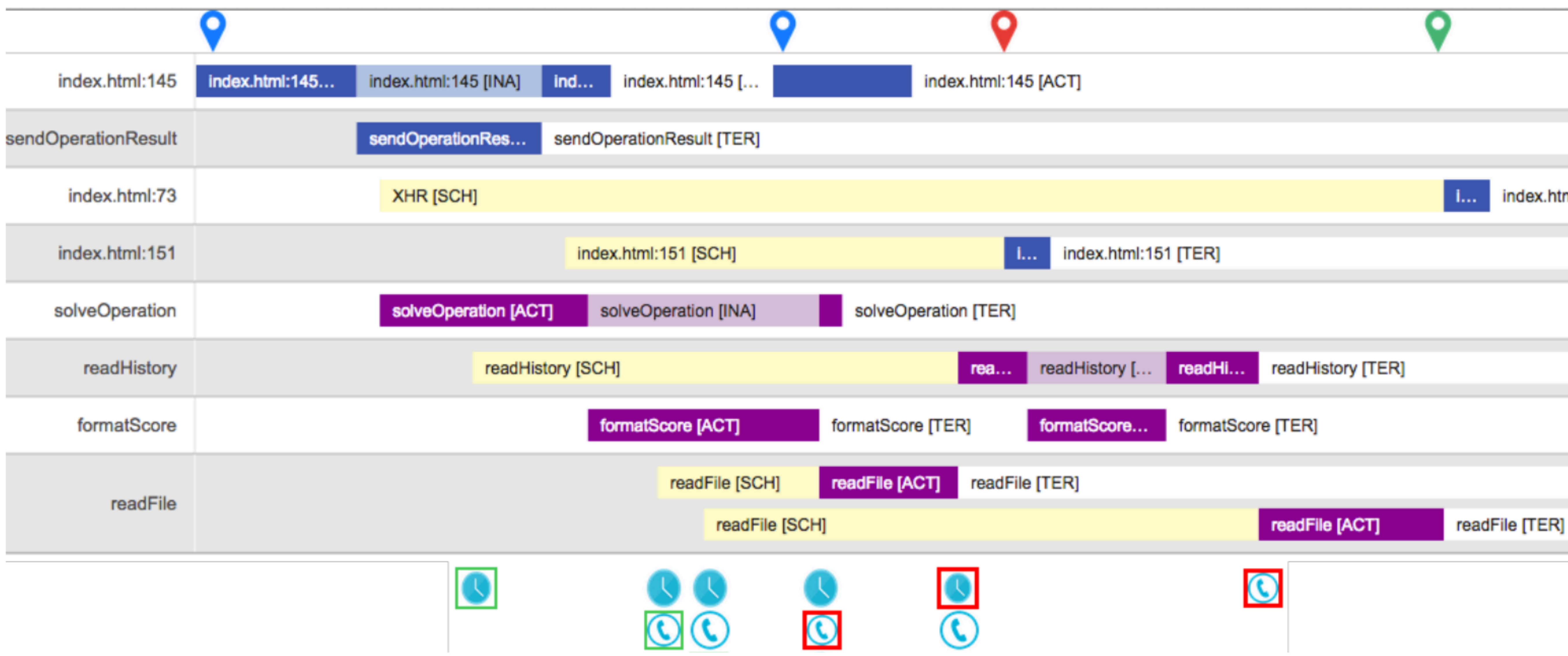
Tom Lieber, Joel R. Brandt, and Rob C. Miller. 2014. Addressing misconceptions about code with always-on programming visualizations. Conference on Human Factors in Computing Systems, 2481-2490.

Reacher


jEdit.java
JEditTextArea.java
StatusBarOptionPane.
/**
* Adds a temporary buffer to the buffer list. This must be done
* before allowing the user to interact with the buffer in any
* way.
* @param buffer The buffer
*/
public static void commitTemporary(Buffer buffer)
{
if(!buffer.isTemporary())
return;
PerspectiveManager.setPerspectiveDirty(true);
addBufferToList(buffer);
buffer.commitTemporary();
// send full range of events to avoid breaking plugins
Outline
Reacher Search
Search downstream from jEdit.commitTemporary() for
method calls
named
sen
bsh.JJTParserState.closeNodeScope(..,..) : void
bsh.JJTParserState.closeNodeScope(..,..) : void
bsh.Parser.jjtreeCloseNodeScope(..) : void
com.microstar.xml.XmlParser.parseNotationDecl() : void
com.microstar.xml.XmlParser.parseNotationType() : void
java.awt.GraphicsEnvironment.getDefaultScreenDevice()
java.awt.GraphicsEnvironment.getLocalGraphicsEnvironment
java.awt.GraphicsEnvironment.getScreenDevices()
javax.swing.JComponent.setRequestFocusEnabled(..) : void
org.gjt.sp.jedit.EditBus.send(..) : void
org.gjt.sp.jedit.search.HyperSearchResults.traverseNodes(
org.gjt.sp.jedit.textarea.TextAreaPainter.setFractionalFontM
Console
Call Hierarchy
Search
Reacher
Back
Forward
ResetZoom
jEdit
-addBufferToList(..)
EditBus
+send(..)
+commitTemporary(..)
Buffer
commitTemporary()
-finishLoading()
+setMode()
+setMode(..)
+propertiesChanged()

Sahand





Saba Alimadadi, Ali Mesbah, and Karthik Pattabiraman. 2016. Understanding asynchronous interactions in full-stack JavaScript. In Proceedings of the 38th International Conference on Software Engineering (ICSE '16),1169–1180. https://doi.org/10.1145/2884781.2884864

10 min break

Tech Talks

In-Class Activity

- In groups of 2 or 3, try out CodeSee.io
  - Go to CodeSee.io and sign up (use the free Community Plan)
  - Find a codebase that you can run CodeSee on (e.g., your 695 project)
  - Build a CodeMap for your codebase
  - Write a reflection on your experiences using CodeSee:
    - Is it helpful in understanding your project? What tasks would it help with? What questions does it help you answer?
    - What's hardest to use about the tool? What questions does it not help answer? What information would you like to see that it doesn't currently provide?
- Submission
  - Submit a pdf with your reflection through Blackboard. 1 submission per group. Due 7:10pm today.