Software Visualization

Alexandra Tamas
1105037

A presentation of the papers:

Software Visualization in the Large,

Thomas A. Ball and Stephen G. Eick, IEEE Computer 29(4):33-43, 1996

CVSscan: Visualization of Code Evolution,

Lucian Voinea, Alex Telea, and Jarke J. van Wijk, Proc. SoftVis 2005, p 47-56.

Questions that are about to be answered in the next 15 minutes:

What are the papers about?

What is the thesis (main point) of these papers?

What are the weaknesses of these works?

What are the strengths of these works?

How could these works be applied?

What was the main experimental question(s) that the authors asked?

Why did they ask this question?

What method(s) did the authors use to address their question(s)?

What results did they obtain?

What new questions are revealed?

What questions remain unanswered?

Paper 1:

Software Visualization in the Large,

Thomas A. Ball and Stephen G. Eick, IEEE Computer 29(4):33-43, 1996

Software Visualization in the Large

Visualization:

- help software engineers cope with complexity
- increase programmer productivity.

Software visible through:

- the display of programs
- program artifacts
- program behavior

Software Visualization in the Large

Visual representations can make the process of understanding software easier

Basic properties of software to be visualized:

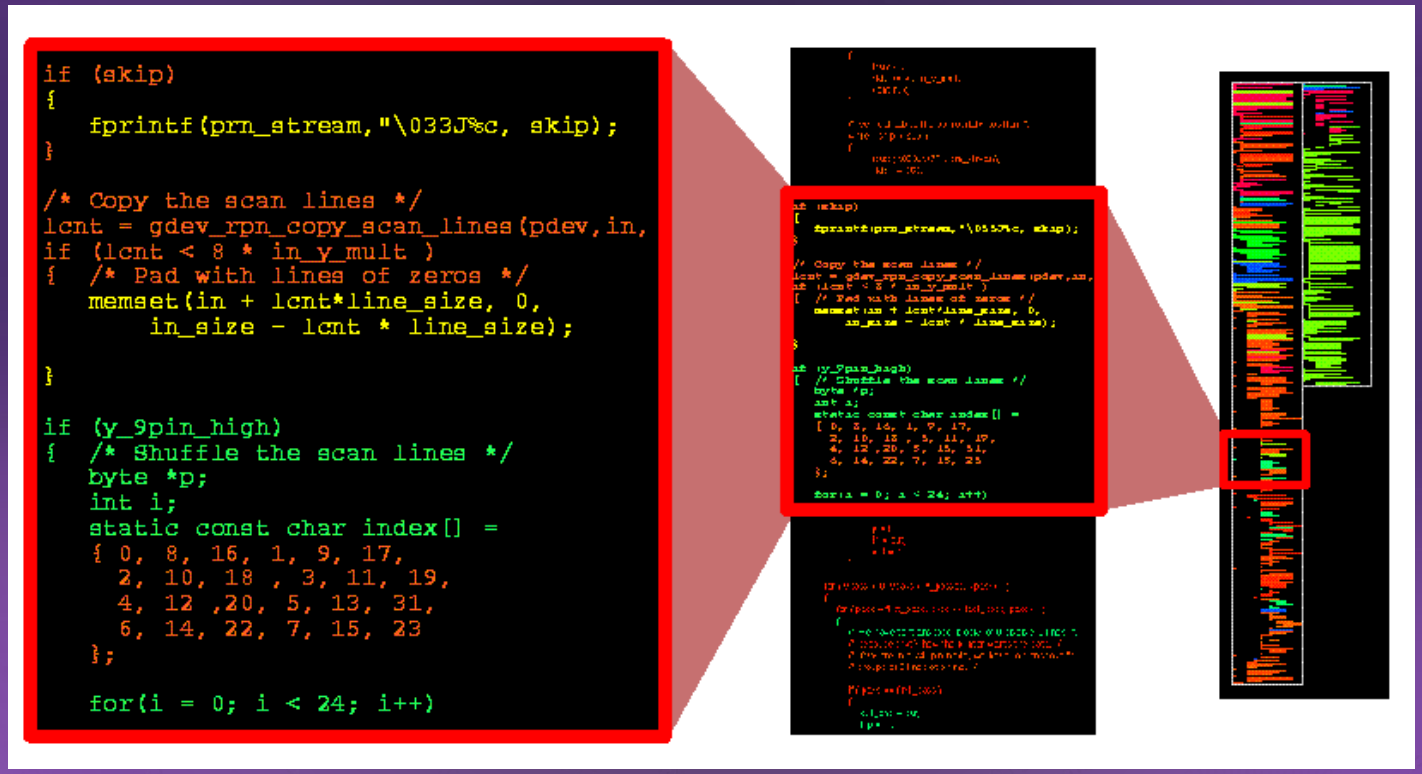
- Software structure
- Run-time behavior
- The code itself

Algorithm visualizations

- usually hand-crafted
- require understanding the code before visualizing it
- infeasible for large systems or tasks involving programmer discovery

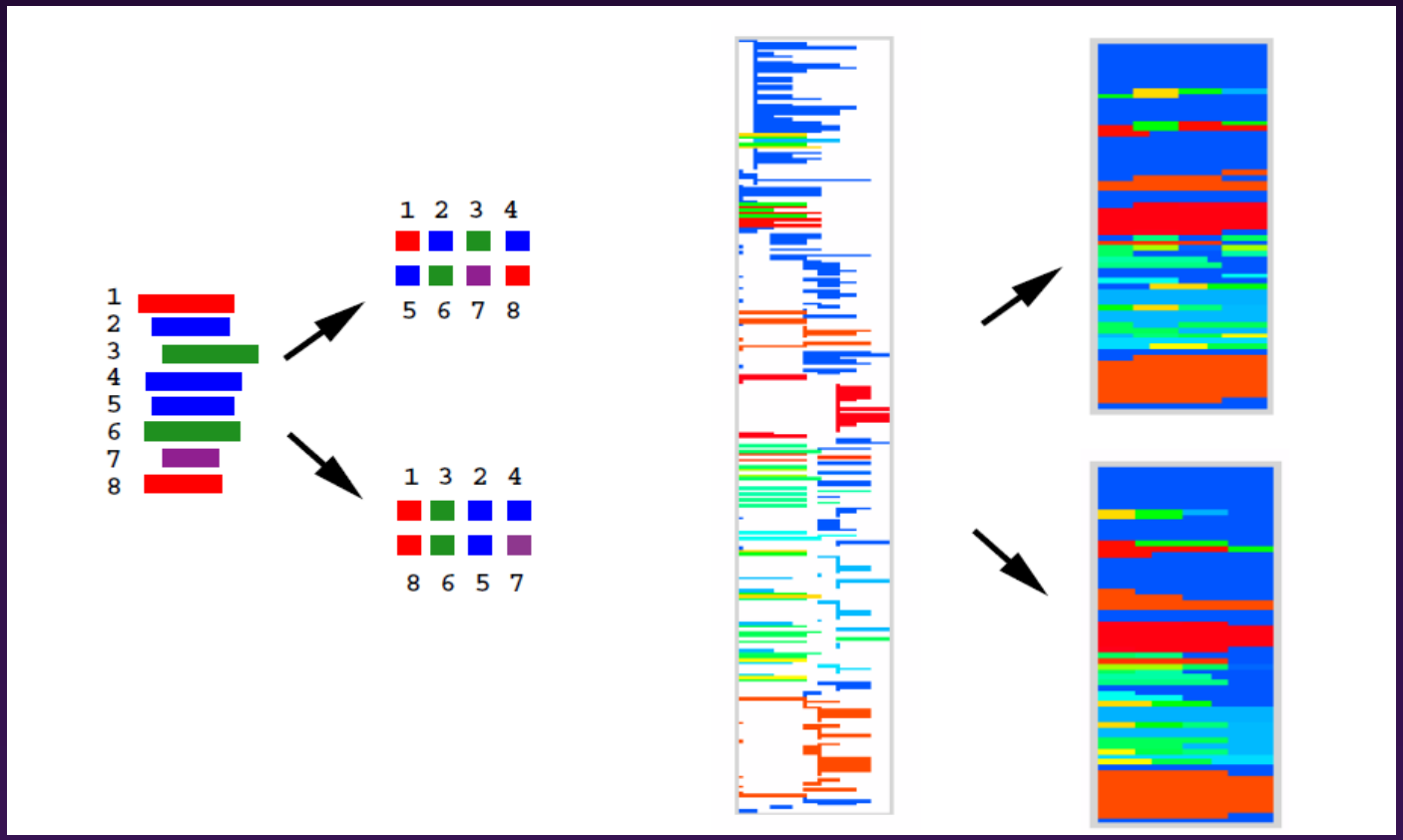
Visual representations for code

- Line Representation



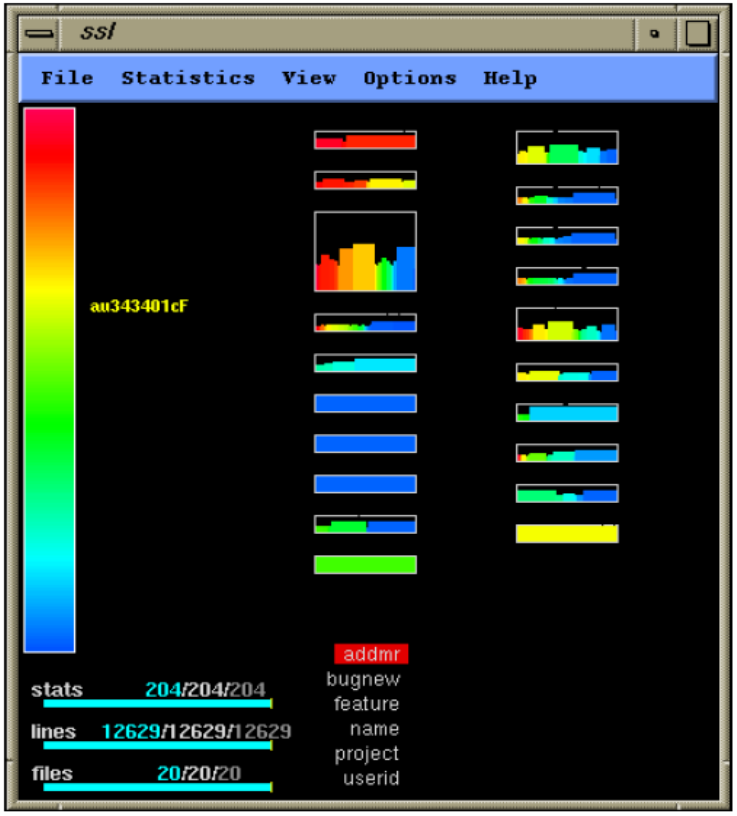
Visual representations for code

- Pixel Representation



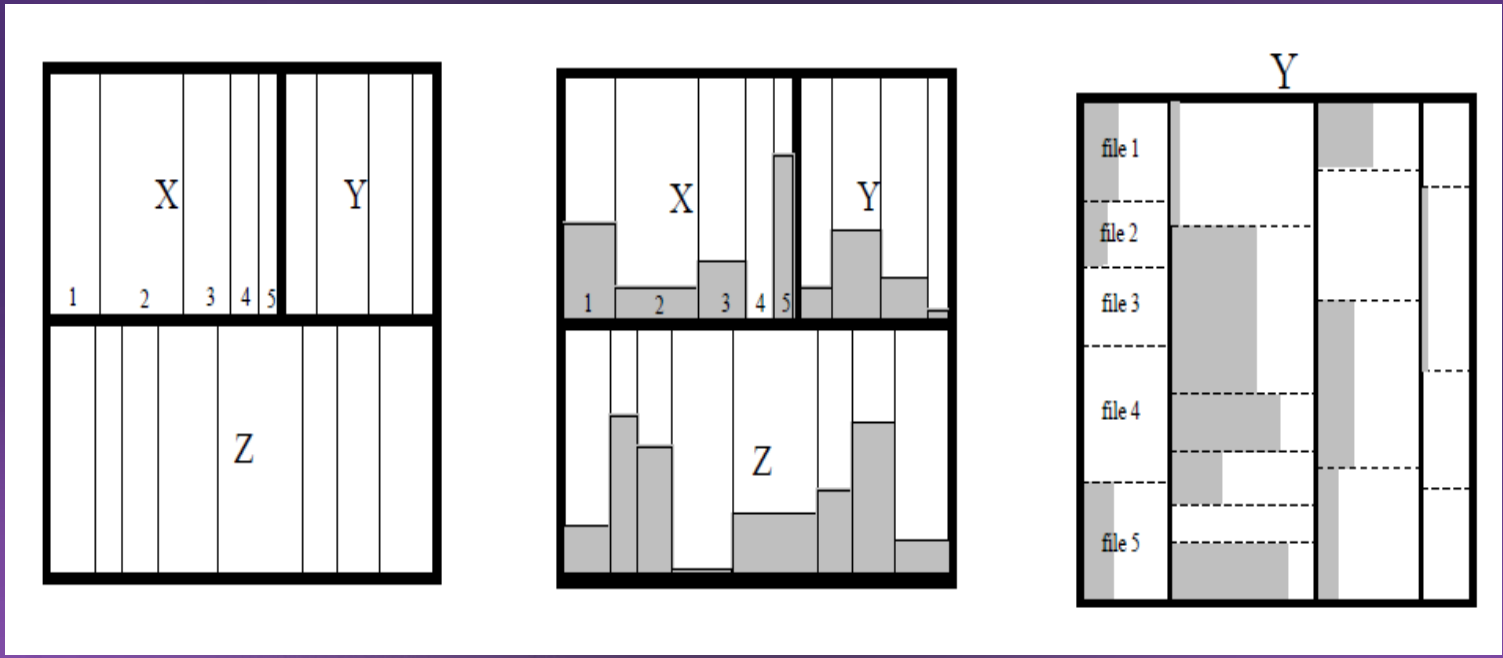
Visual representations for code

- File Summary Representation



Visual representations for code

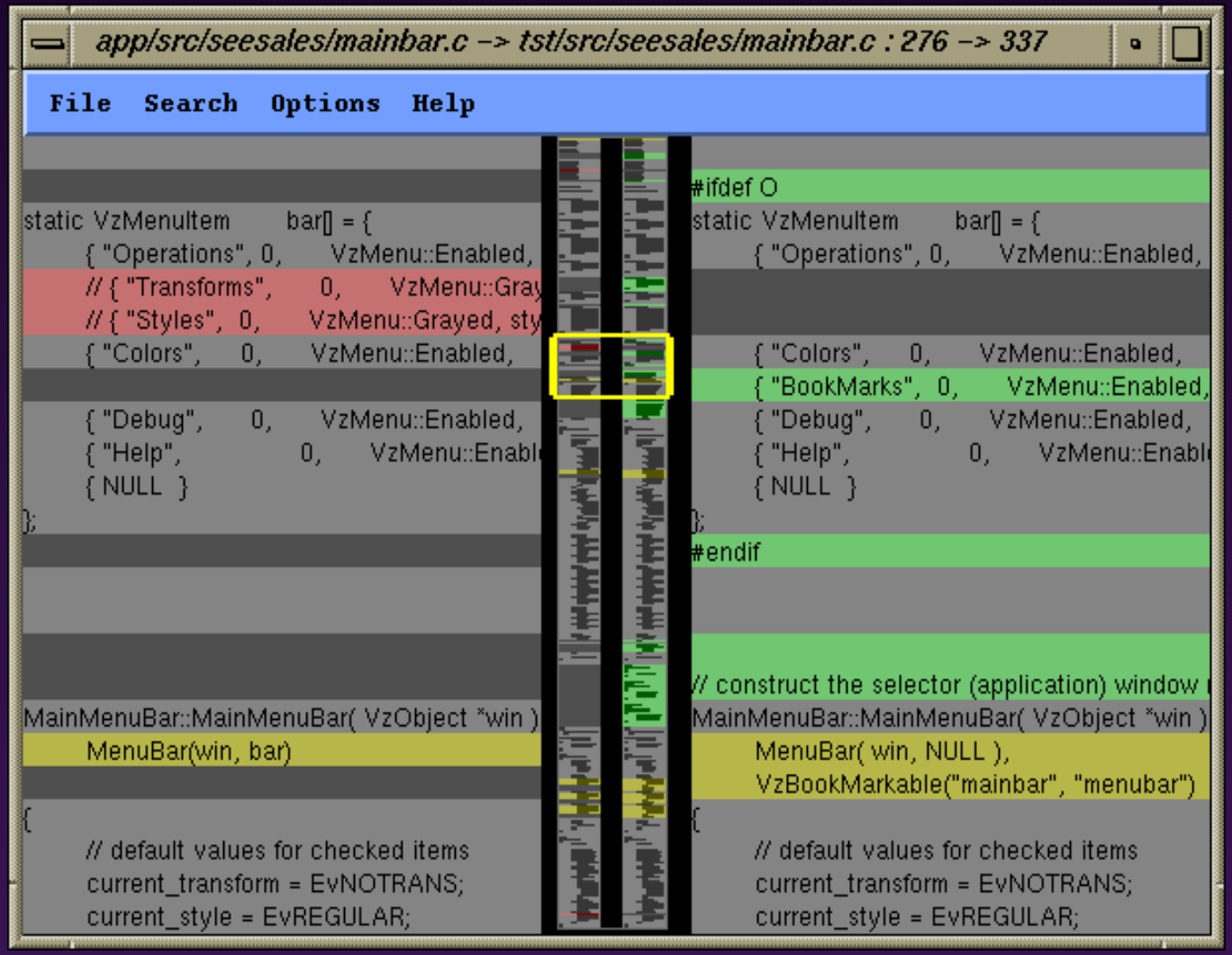
- Hierarchical Representation



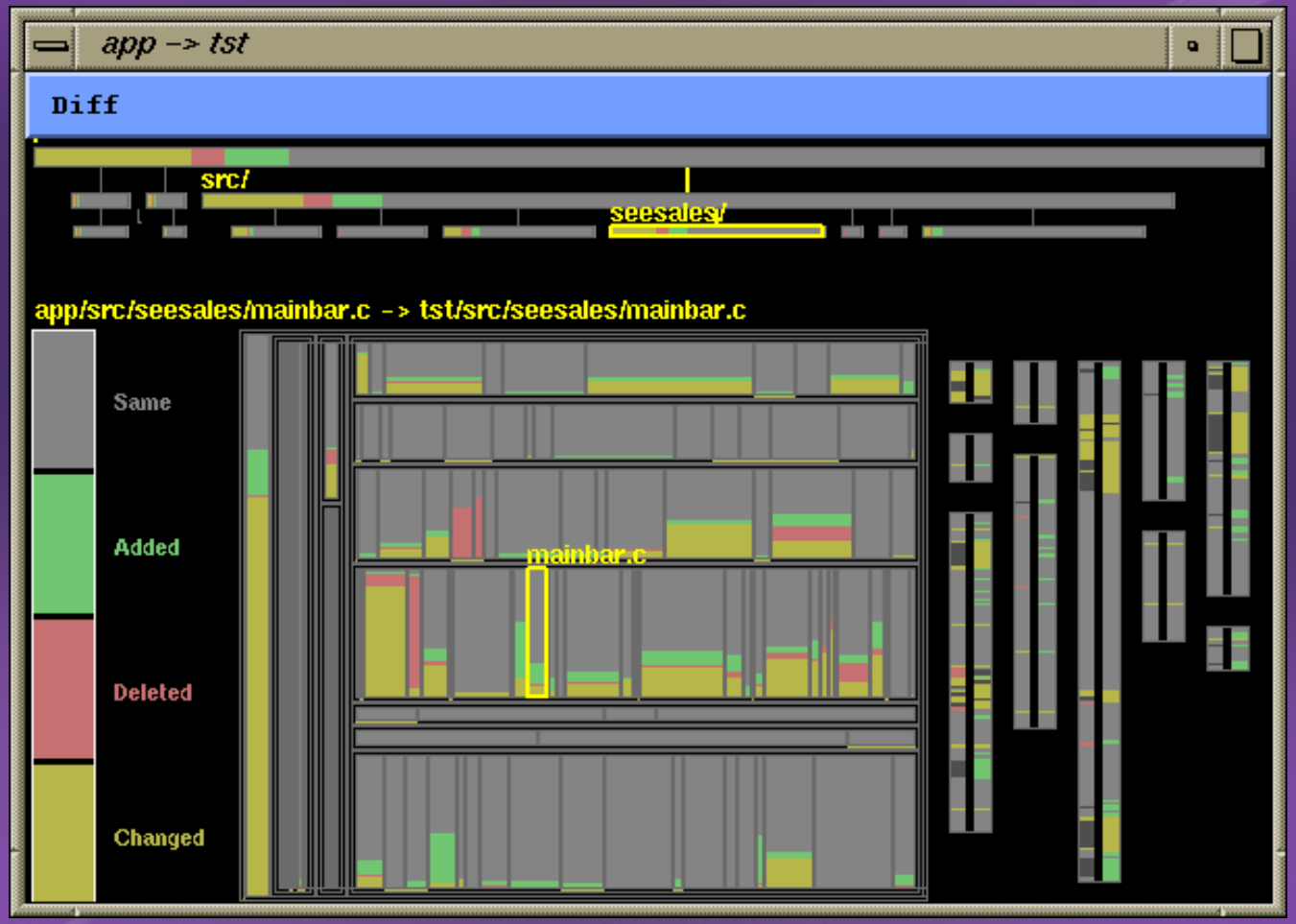
Ways productivity can be increased through Software tools

- code discovery
- highlight of regions that exhibit \code decay
- identify the current development activity
- inspect code to ensure that it meets coding standards.

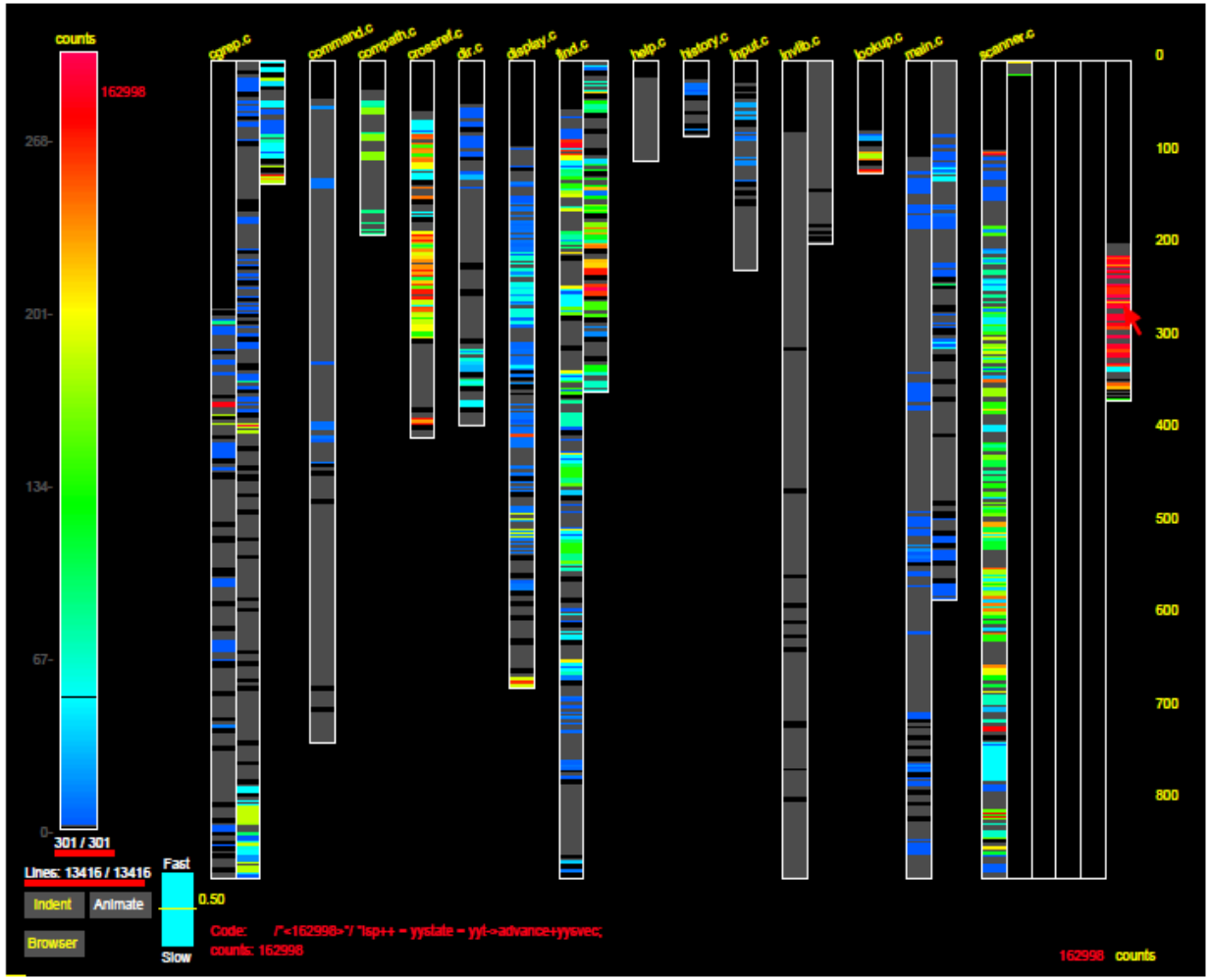
Program comparison



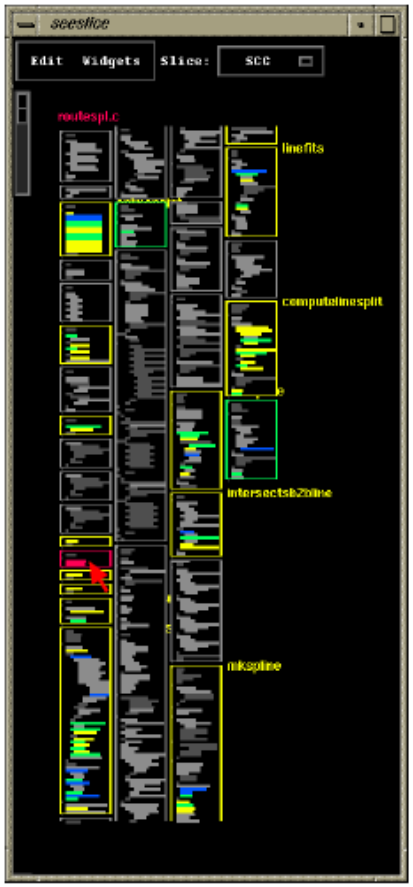
Program comparison



Program profiles and code coverage



Program Slices



Weakness(es) and strength(s)

(-) most part of the images at the end of the paper and not right after the corresponding paragraph

(+) the neatly arranged writing and the good documentation

The main experimental question that the authors asked and why did they ask this question

HOW to understand complex system behavior from code?

BECAUSE

- understanding,
- changing,
- and repairing

code in large systems is time consuming and costly

What method did the authors use to address their question?

description of four innovative visual representations of software that:

- scale to production-sized systems
- illustrate their usage in five software case studies

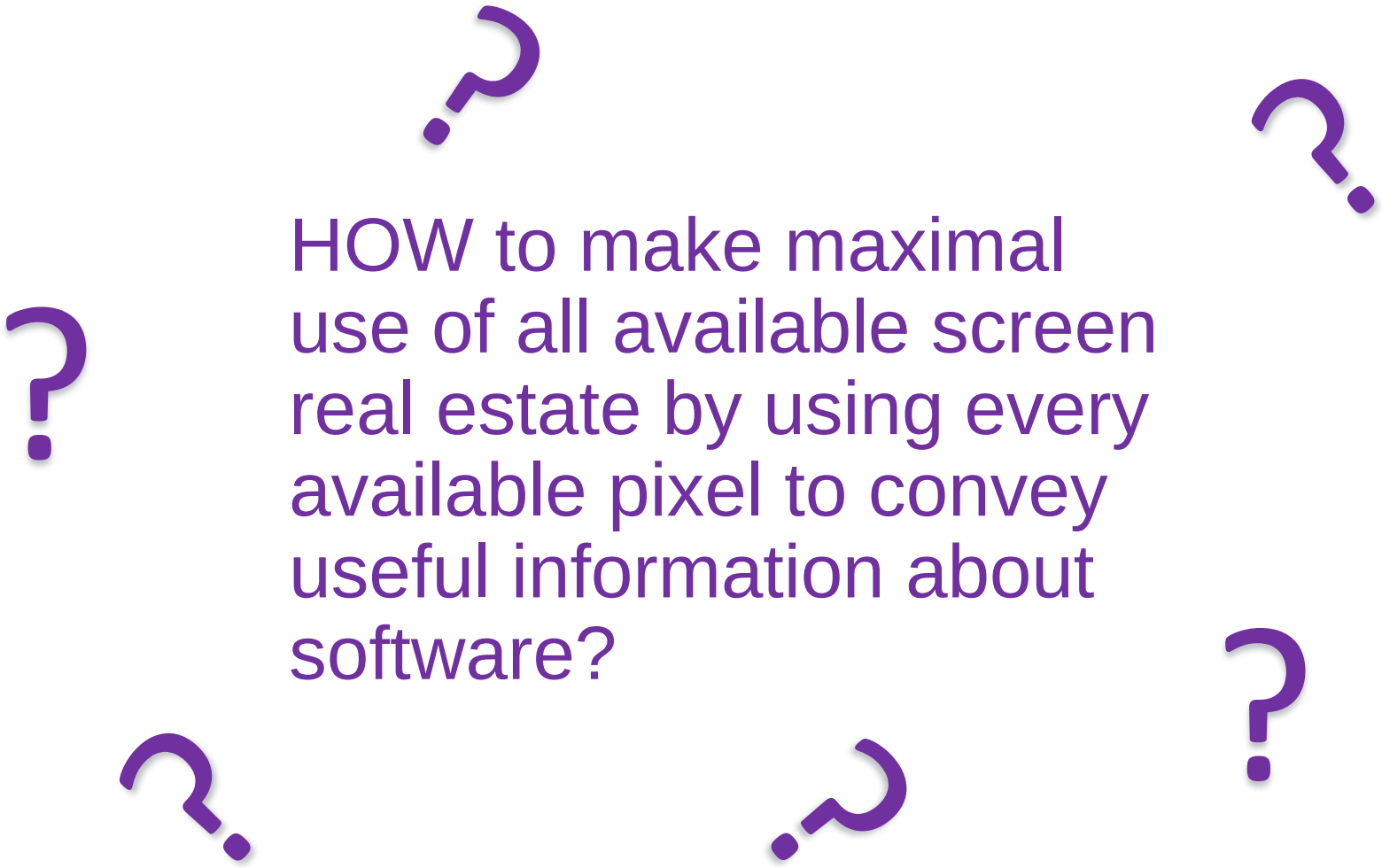
How could this work be applied

to help software developers working on Bell Laboratories 5ESS product

Result

software visualization is important because most software artifacts are naturally invisible

A new question that is revealed



HOW to make maximal use of all available screen real estate by using every available pixel to convey useful information about software?

A question that remains partially unanswered

the graph layout problem, that is the most diffcult aspect of showing software through graphs

Paper 2:

CVSscan: Visualization of Code Evolution,

Lucian Voinea, Alex Telea, and Jarke J. van Wijk, Proc. SoftVis 2005, p 47-56.

What is the paper about

- Section 2 : a review of the line-based visualization tools for software evolution and their challenges
- Section 3: introduction of CVSscan, a tool developed to test and validate the proposed visualization techniques
- Section 4: results of two case studies. These studies show how the approach can be successfully used to investigate the evolution of files from real life software projects
- Section 5: summarizes the novel contribution brought to software evolution visualization and outlines future directions of research

CVSscan: Visualization of Code Evolution

- during the life cycle of a software system, the source code is changed many times
- in the last decade, maintenance and evolution exceeded 90% of the total software development costs
- the corrective approach aims to facilitate the maintenance phase, and is supported by program and process understanding and fault localization tools, e.g.
  - SeeSoft,
  - Aspect Browser
  - Tarantula

CVSscan: Visualization of Code Evolution

- industry practice studies have shown that maintainers spend 50% of their time on understanding this code
- a novel concept, the bi-level code display that gives a detailed view of
  - the contents of a code fragment
  - its evolution in time

Visual mapping

- No indentation and line length to suggest code structure, BUT a fixed-length pixel line for all code lines and color to encode structure

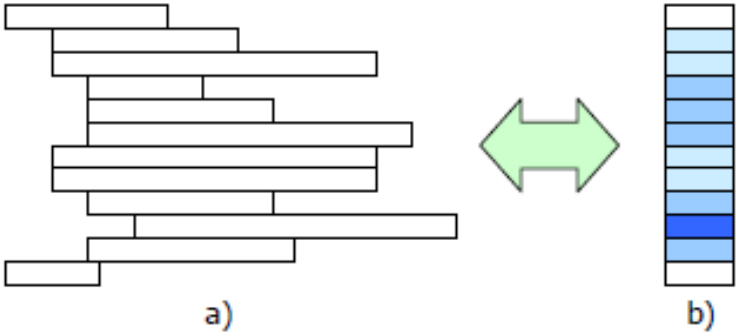


Figure 3: Line layout a) SeeSoft b) CVSscan

Visual mapping

visualize on the same screen all versions that a file has during its evolution, instead of all files in a project at a given time

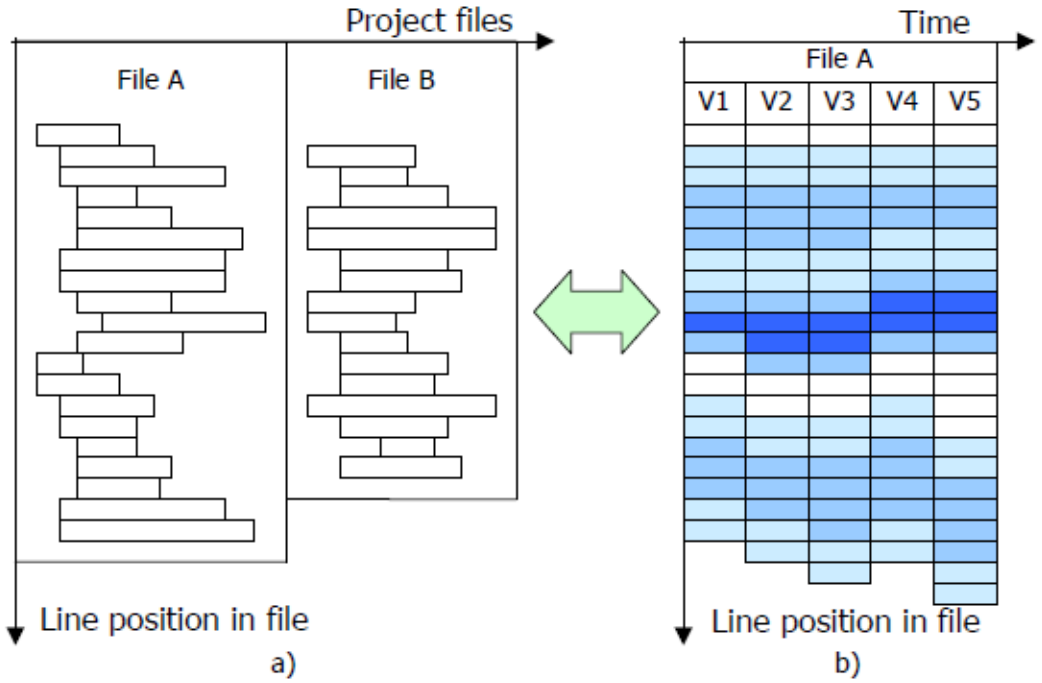


Figure 4: Use of horizontal axis in line-based visualizations a) files, in SeeSoft b) time, in CVSscan

Visual mapping

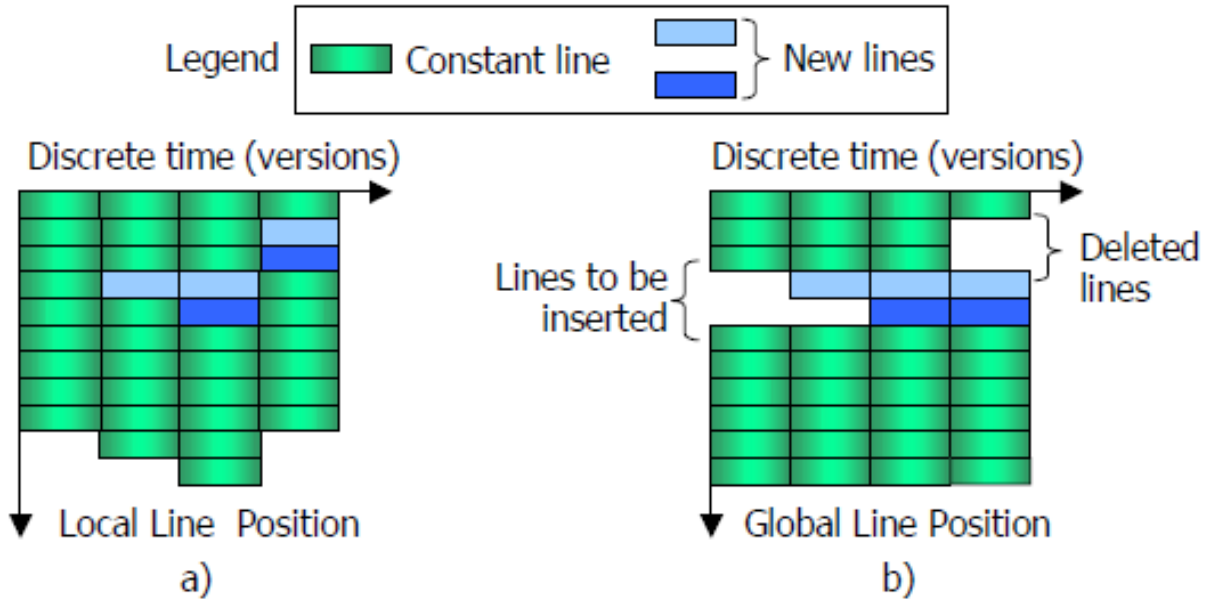


Figure 6: Line layout in CVSscan: a) file-based b) line-based

Visual mapping

the CVSscan visualization of a file evolution

through 65 versions

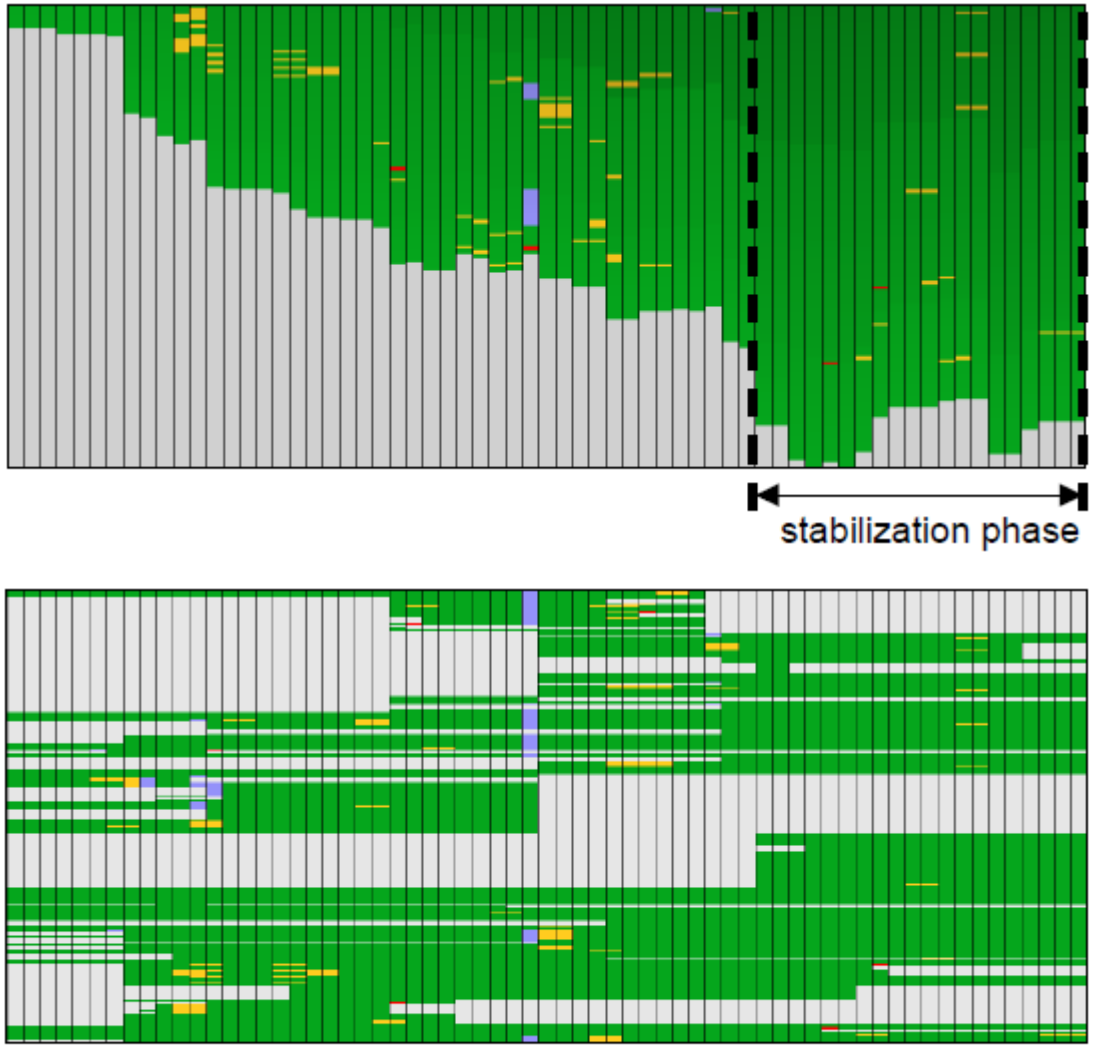


Figure 7: *Line status* visualization. File-based (top) and line-based (bottom) layouts

Visual mapping

different color encodings on a zoom-in of the line-based layout in Figure 7

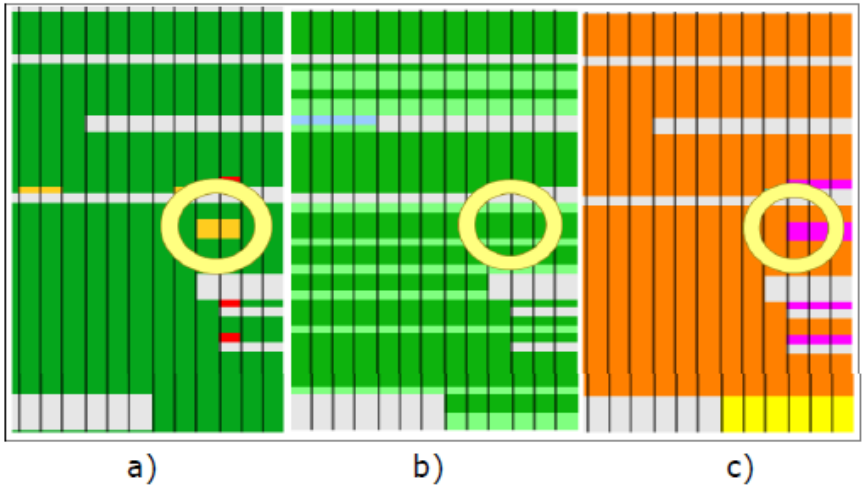


Figure 8: Attribute encoding: a) *line status*; b) *construct*; c) *author*

Weakness(es) and strenght(s)

(-) the images in the paper not very clear; difficult to understand the details

(+) the large amount of information compressed in a few pages

The main experimental question that the authors asked and why did they ask this question

HOW developers can be enabled to get insight in the changes of the source code?

BECAUSE

- to facilitate the understanding of the status, history and structure better
- to instance the roles played by various contributors

The method the authors used to address their question

using multiple correlated views on the evolution of a software project

The result of this paper

line-based evolution visualization of code supports a quick assessment of the important activities and artifacts produced during development, even for users that had not taken part in any way in developing the examined code

A new question that is revealed



how can be tested the efficiency of this approach for real world use cases?



A question that remains partially unanswered

how to extend this approach with higher-level overviews, such as whole-project evolution visualizations, to enable evolution analyses on entire systems?

Thank you for your attention!