

# On Feature Traceability in Object-Oriented Programs

Giuliano Antoniol, Ettore Merlo,  
Yann-Gaël Guéhéneuc, and Houari Sahraoui

TEFSE 2005  
Long Beach, CA, USA



Group of Open, Distributed Systems,  
Experimental Software Engineering  
Department of Informatics and Operations Research  
University of Montreal



SOCGER Laboratory  
Department of Informatics  
École Polytechnique de Montréal



# Problem and Solution

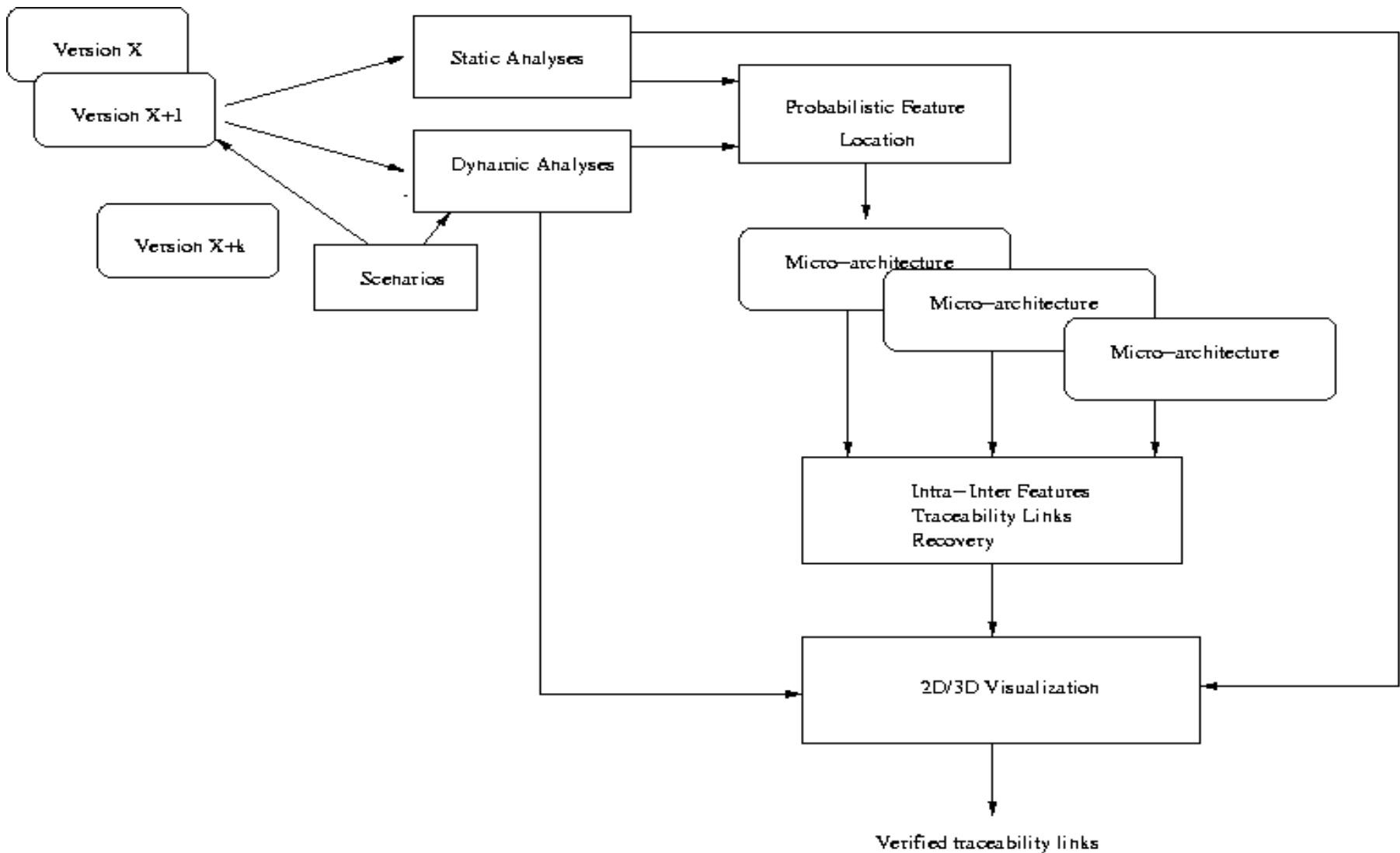
- Re(verse)-engineering based on source code browsing mainly
  - Source code browsing is expensive
- A solution
  - Higher-level abstractions through feature identification, comparison, and evolution
  - Maintaining traceability links
    - Between abstractions and source code
    - Between abstractions among versions



# Technologies

- Static and dynamic code analysis
  - Locate feature, collect metrics...
- 2D/3D visualization
  - Display large number of entities
  - Highlight overlapping abstractions
  - Capture trends in features evolution
- Information retrieval
  - Add semantics knowledge to feature location, to ease feature evolution

# The process





# Our Approach

1. Program model creation
  - Static analyses
2. Feature identification
  - Dynamic analyses
  - Knowledge-based filtering
  - Probabilistic ranking
3. Feature visualization
  - 3D mapping of features and entities

## 2. Feature Identification (1/3)

- Dynamic analysis
  - Traces = Sequences of intervals
  - Intervals = Sequences of events
  - Events are (ir)relevant to the feature
- A functionality, two scenarios
  - ⇒ Two traces
- Comparison of the two traces to identify the feature related to the functionality

## 2. Feature Identification (2/3)

### ■ Probabilistic ranking

- $\mathcal{F}$  ( $\overline{\mathcal{F}}$ ) scenarios (not) exercising a feature
- $\mathcal{I}^*$  ( $\mathcal{I}$ ) intervals with (ir)relevant events
- $\mathcal{I}^*$  may contain irrelevant events
  
- Wilde's relevance index for an event  $e_i$

$$p_c(e_i) = \frac{N_{\mathcal{F}}(e_i)}{N_{\mathcal{F}}(e_i) + N_{\overline{\mathcal{F}}}(e_i)}$$

## 2. Feature Identification (3/3)

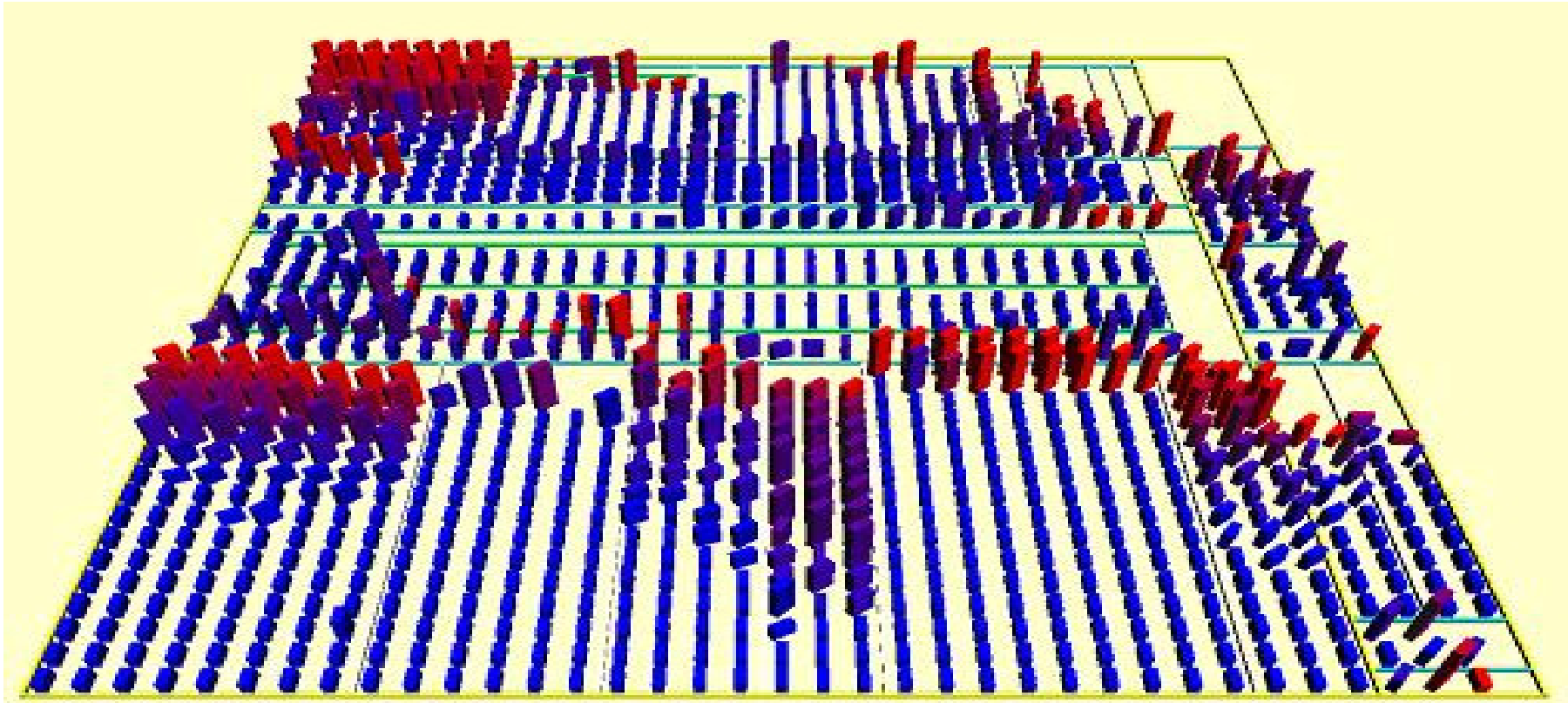
### ■ Probabilistic ranking

- Irrelevant events frequent with  $\mathcal{F}$
- Few intervals in  $\mathcal{I}^*$ , many intervals in  $\mathcal{I}$
- Renormalisation of Wilde's equation

$$f_{\mathcal{I}^*}(e_i) = \frac{N_{\mathcal{I}^*}(e_i)}{N_{\mathcal{I}^*}} \quad ; \quad f_{\mathcal{I}}(e_i) = \frac{N_{\mathcal{I}}(e_i)}{N_{\mathcal{I}}}$$

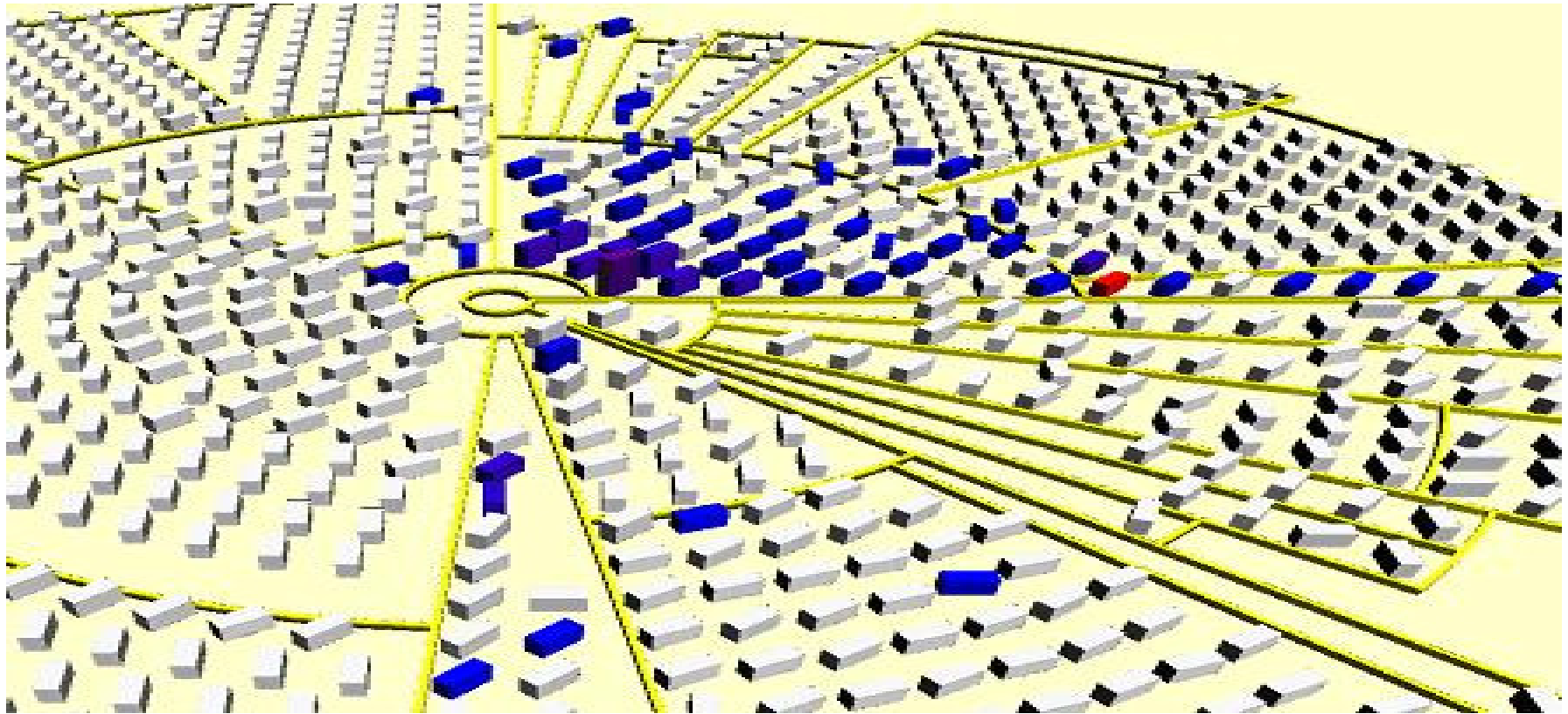
$$r(e_i) = \frac{f_{\mathcal{I}^*}(e_i)}{f_{\mathcal{I}^*}(e_i) + f_{\mathcal{I}}(e_i)}$$

# 3. Feature Visualization (1/3)



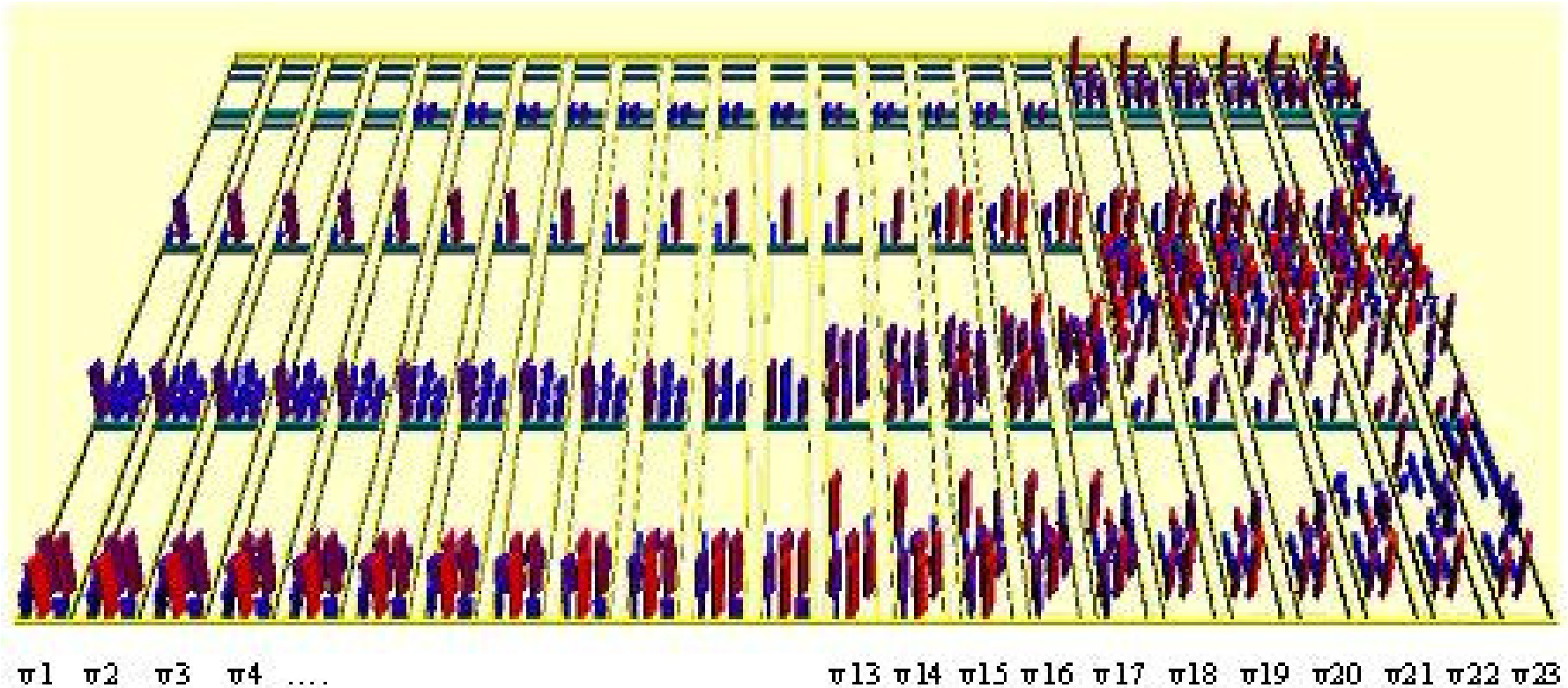
- Display large number of entities
- Colors, height, twist are metric values

# 3. Feature Visualization (2/3)



- Highlight overlapping entities among features
- Colors, height, twist are metric values

# 3. Feature Visualization (3/3)



- Capture trends in features evolution
- Colors, height, twist are metric values



# Questions on Feature Evolution

- Functionalities and micro-architectures common across release
  - Do they change?
- Newly added functionalities
  - How much do they reuse, integration?
- Disappearing functionalities
  - What happened to the micro-architectures?
- Micro-architectural change without feature change

# Example: Mozilla and Firefox

- Large software implementing similar or identical features
- Firefox largely derived from Mozilla
- Many architectural differences

	Mozilla	Firefox
Classes	4853	5438
Methods	53671	57748
Inheritance	5314	4667
Aggregations	6727	5035
Associations	17362	10397



# Conclusion

(Not really a conclusion 😊)

- There is so much to be done
- Reliable multi-language parsers
- Scalable visualization tools
- Scalable IR tools
- Fast and accurate pattern matching