Semantic Code Models
for Concept-aware Programming Environments

Toni Mattis
Robert Hirschfeld
Software Architecture Group
Hasso Plattner Institute, University of Potsdam, Germany

HPI Research School 20 Nov. 2018, Nanjing

Problem: Architectural Drift

Many software projects start with good modularity

- Modules with clear responsibilities can vary independently
- Recognizing and locating concepts is relatively cheap

modules

separation of concerns

Problem: Architectural Drift

With growing code bases…

- Concepts tend to scatter and entangle
- Programmers need more attention to recognize concepts

modules
separation of concerns

scattering / tangling

Goal

Help programmers...

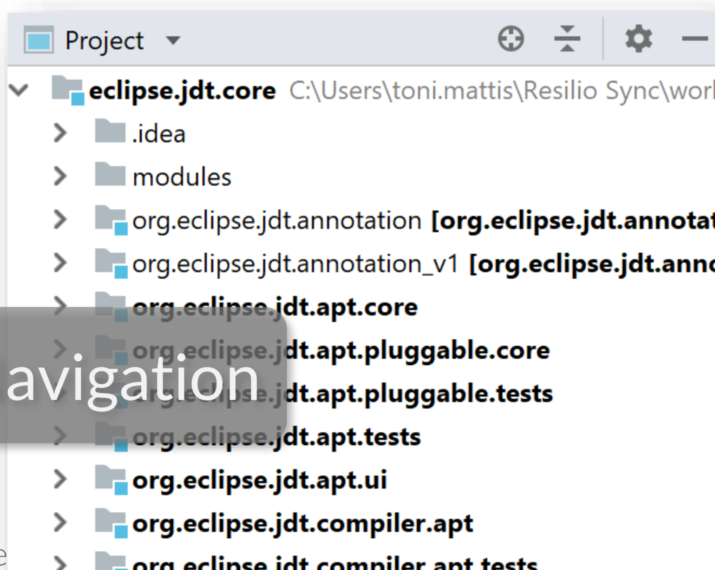
» Find, navigate, and relate existing concepts to code
» Improve architecture to better express underlying concepts

Goal (1)

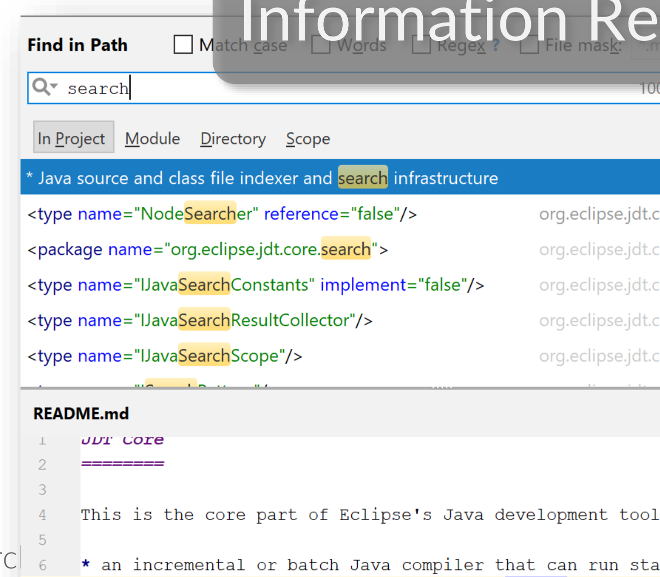
» **Reverse Engineering:** Help programmers understand the conceptual structure of a large system

```
grep -r -i --include \*.java "search"
```

Navigation

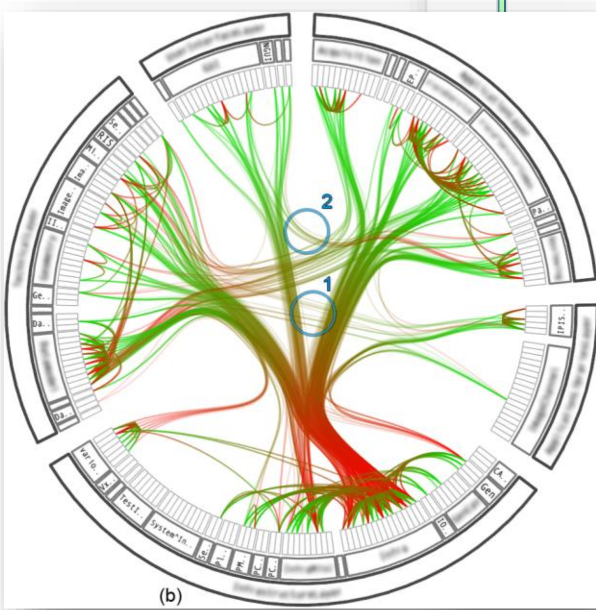


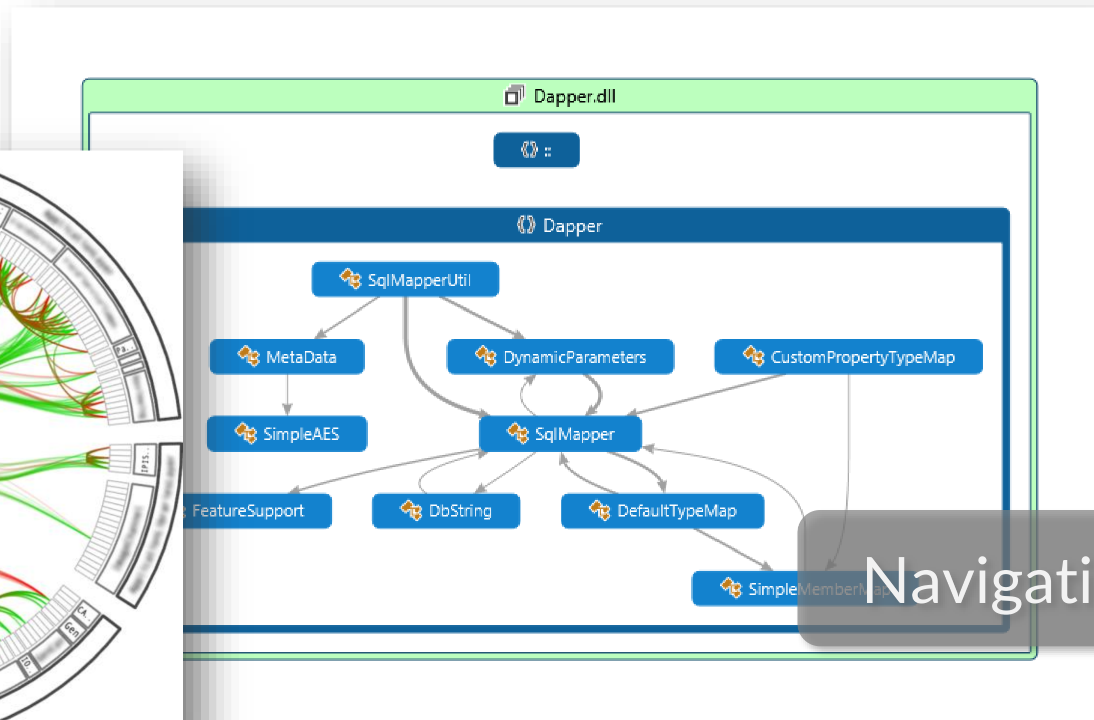
Information Retrieval



Goal (1)

» **Reverse Engineering:** Help programmers understand the conceptual structure of a large system



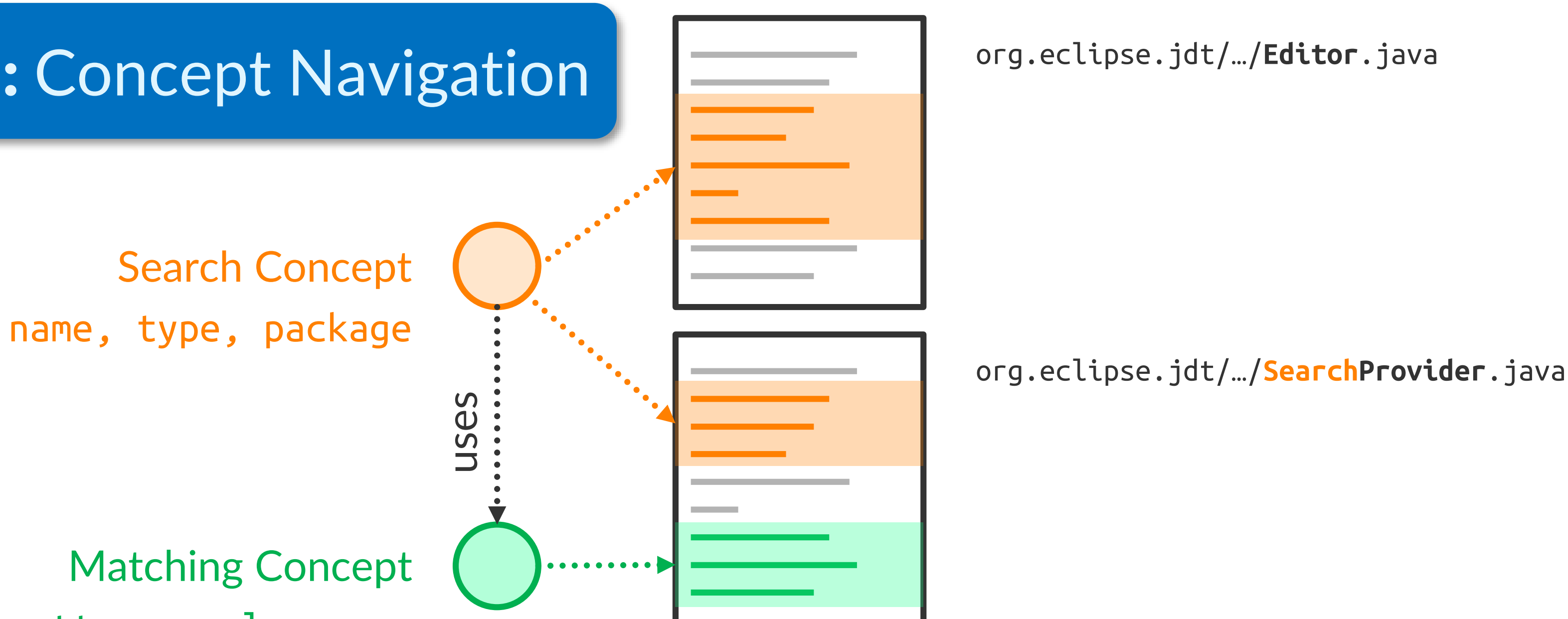


Navigation

Goal (1)

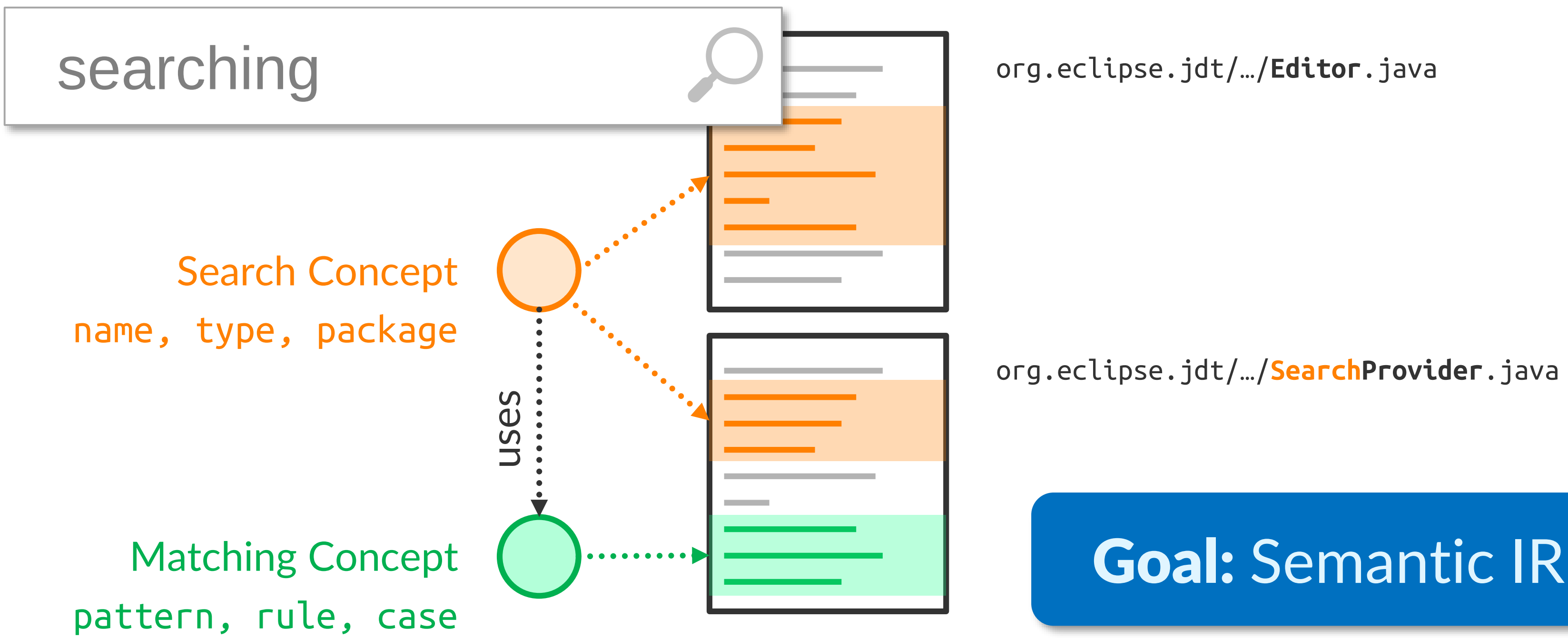
» **Reverse Engineering:** Help programmers understand the conceptual structure of a large system

Goal: Concept Navigation



Goal (1)

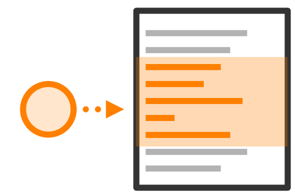
» **Reverse Engineering:** Help programmers understand the conceptual structure of a large system



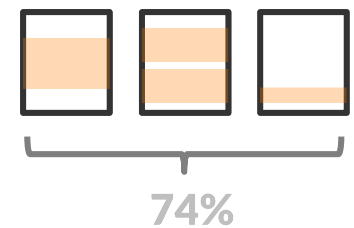
Goal: Semantic IR

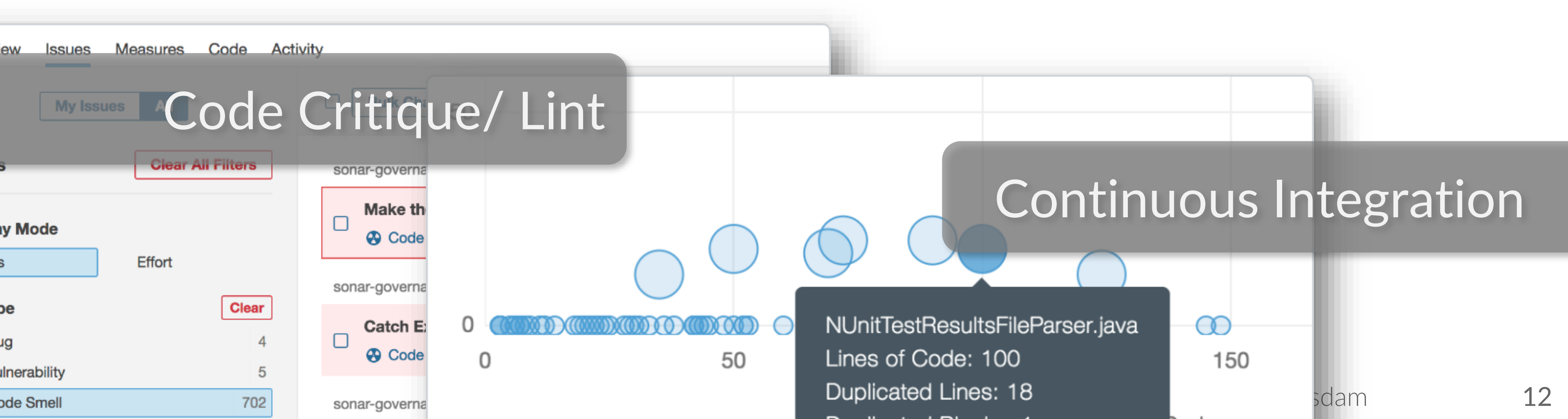
Goal (2)

- **Reverse Engineering:** Help programmers understand the conceptual structure of a large system



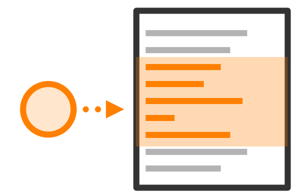
- **Metrics:** Quantify how architecture deviates from conceptual structure



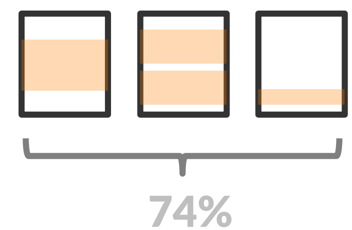


Goal (3)

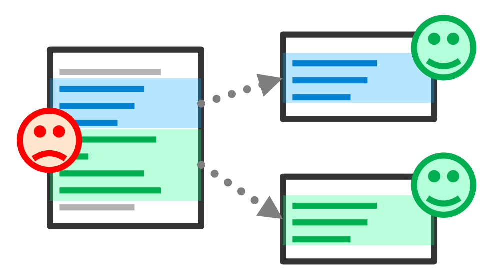
» **Reverse Engineering:** Help programmers understand the conceptual structure of a large system



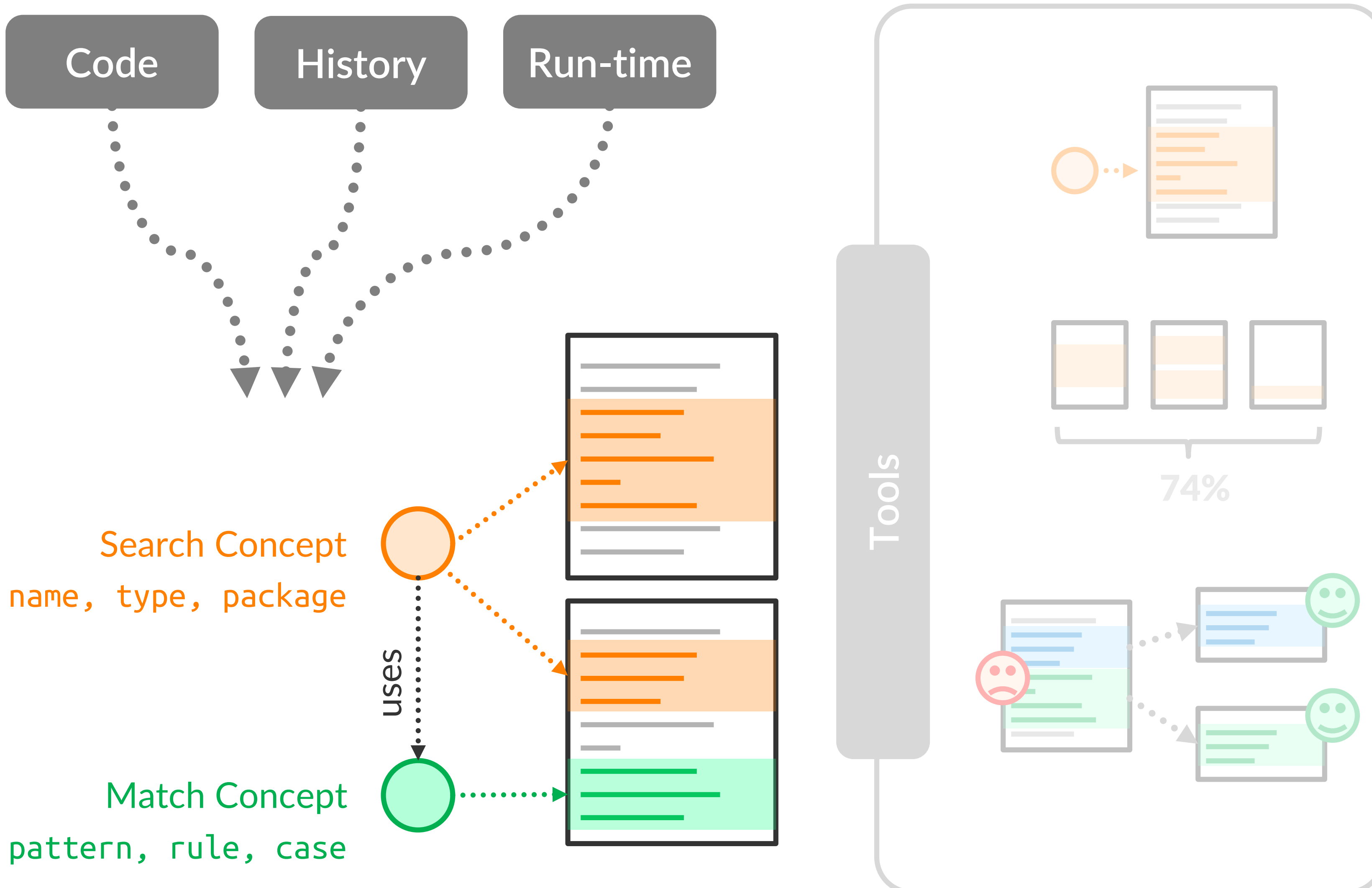
» **Metrics:** Quantify how architecture deviates from conceptual structure



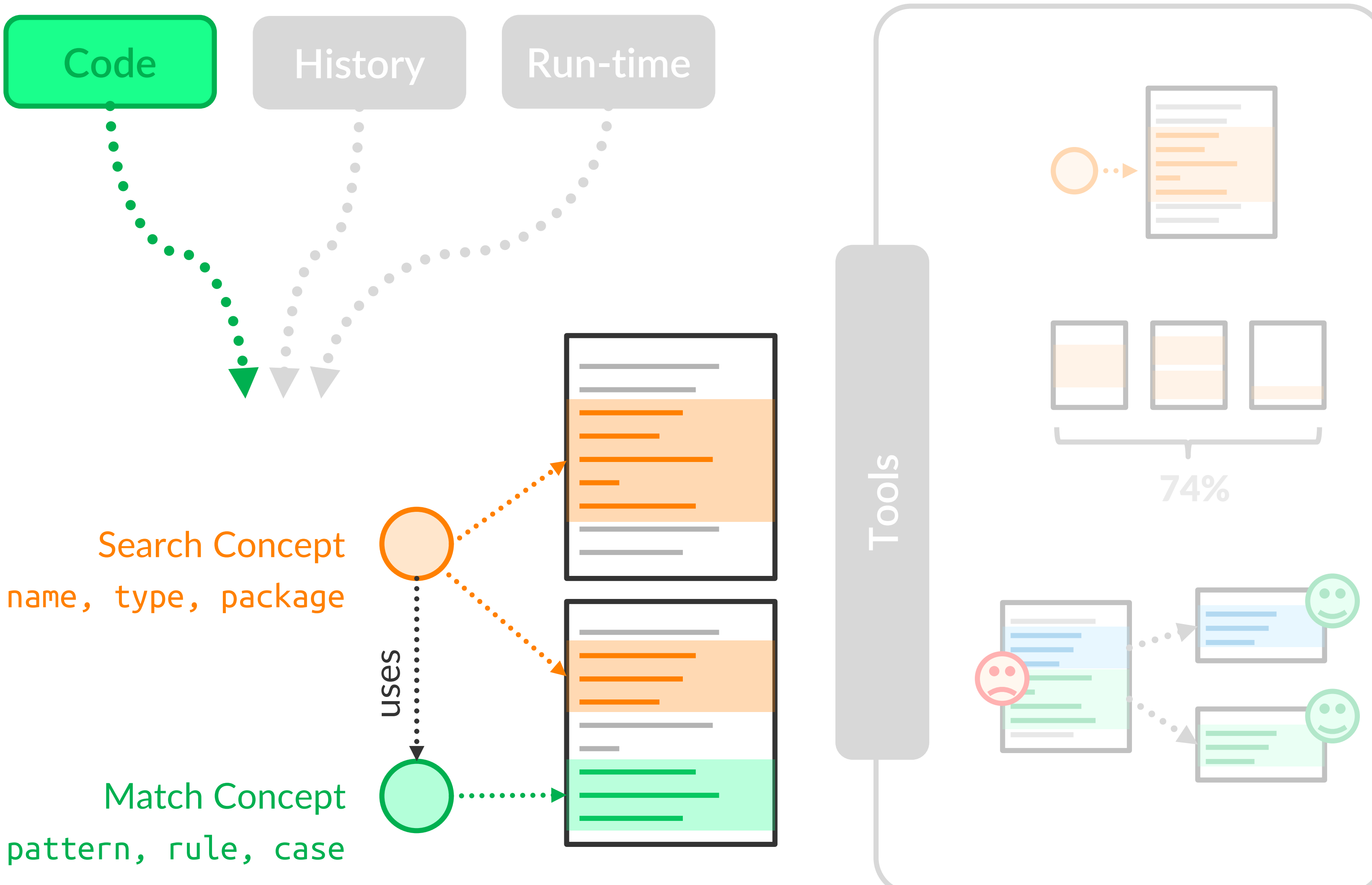
» **Forward Engineering:** Maintain and improve modularity by real-time feedback and recommendations



Approach: Repository Mining



Approach: Repository Mining



Concept Model

```
Canvas » draw: anObject
  ^ anObject drawOn: self
```

draw, canvas, fill, …

```
Morph » drawOn: aCanvas
  aCanvas fillRectangle: self bounds.
```

```
Morph » bounds: newBounds
  self position: newBounds topLeft;
 extent: newBounds extent.
```

bounds, position, extent, …

concept labels
which concept a name belongs to

concepts
prevalent names

Composition & Abstraction Barriers

```
Canvas » draw: anObject
  ^ anObject drawOn: self
```

draw, canvas, fill, ...

```
Morph » drawOn: aCanvas
  aCanvas fillRectangle: self bounds.
```

mixing

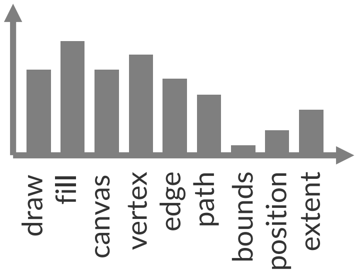
uses
(implemented through)

```
Morph » bounds: newBounds
  self position: newBounds topLeft;
  extent: newBounds extent.
```

bounds, position,
extent, ...

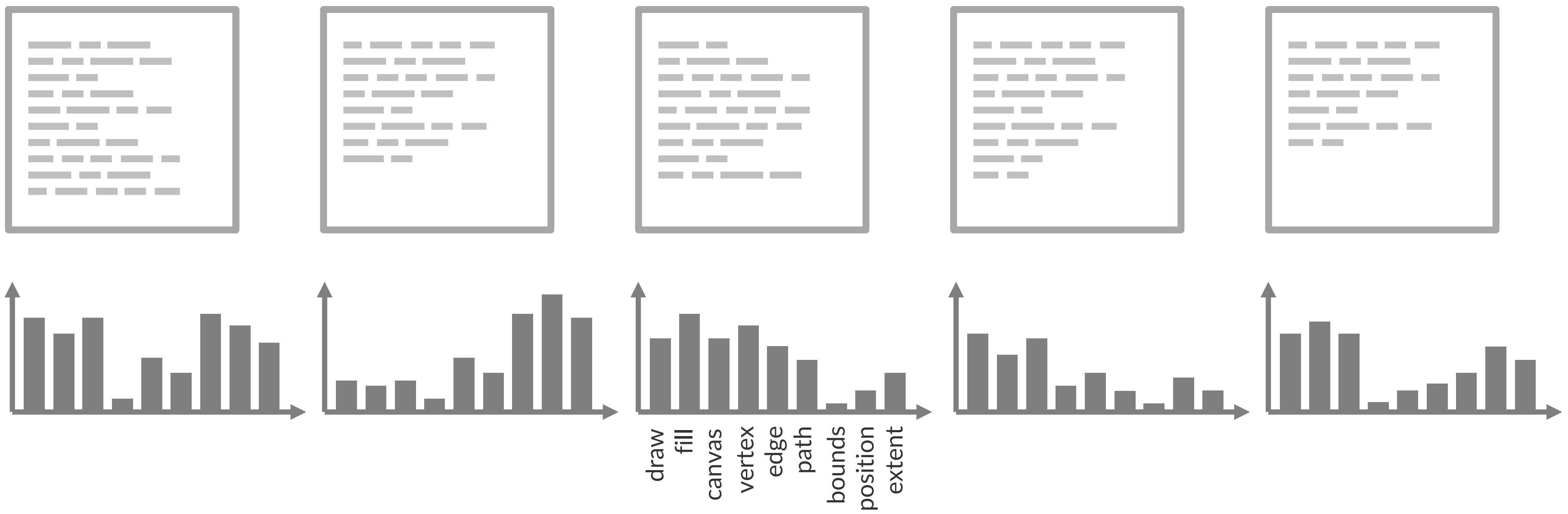
Recap: Topic Models

Document
(news article, tweet, paper, **code module?**)

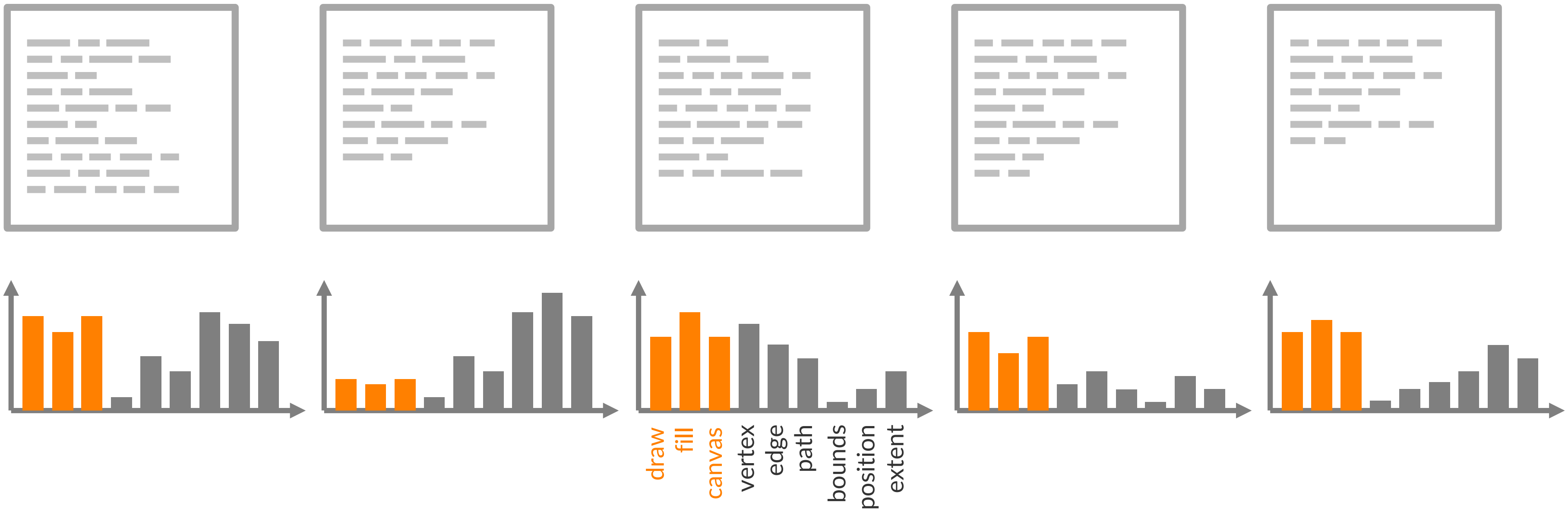


Bag of Words (BoW)
(histogram, multiset, ...)

Recap: Topic Models

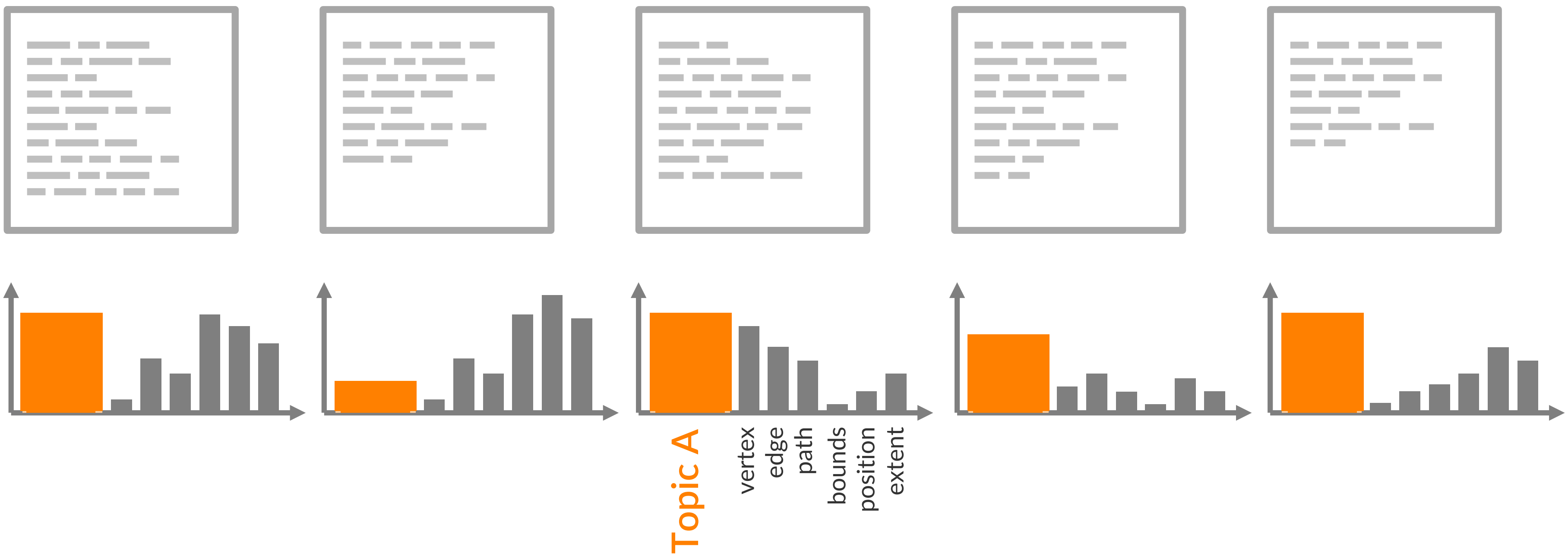


Recap: **Topic Models**

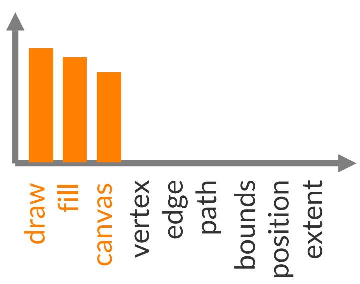


Semantically related words appear **correlated** (Distributional Hypothesis)

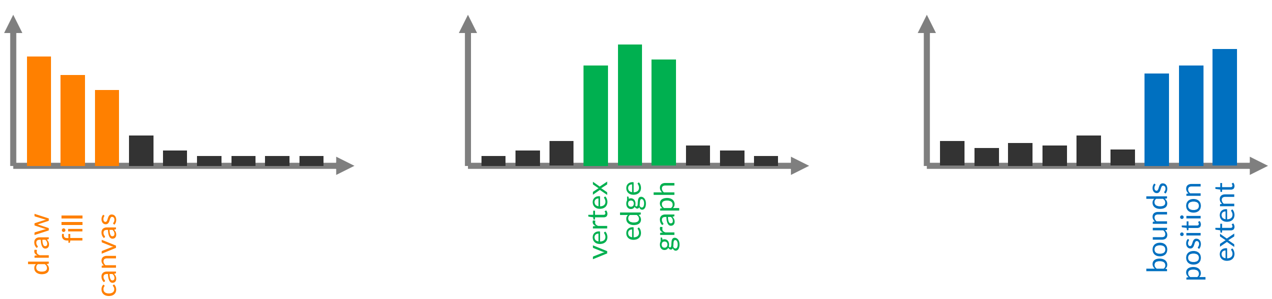
Recap: Topic Models



Topic A



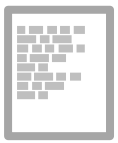
Recap: Topic Models

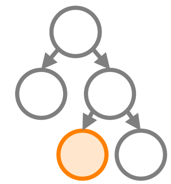


topics
terms sharing a common distribution

Limitations of Classical Topic Models

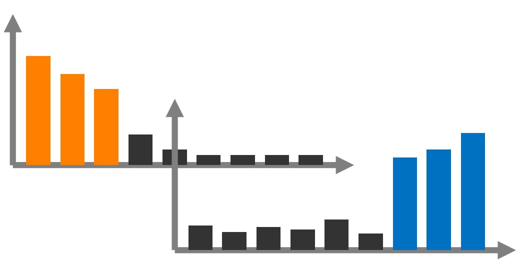
Flat Documents vs. **Hierarchy of Code**

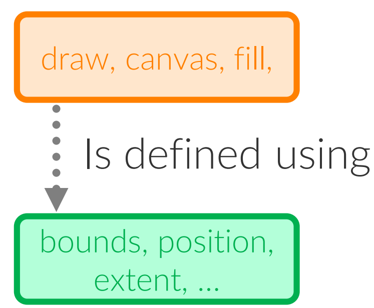




Document Independence vs. **Module Dependencies**

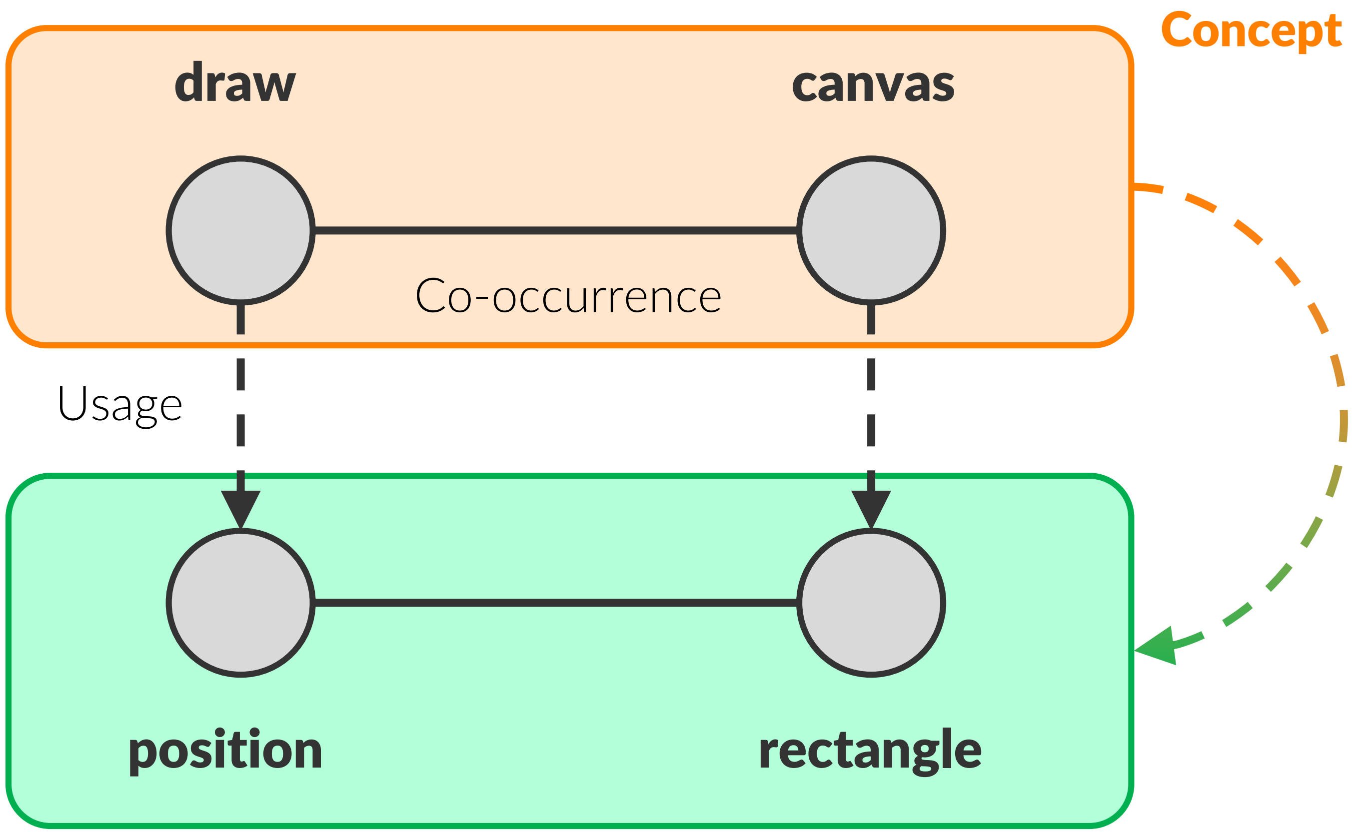
Topical Independence vs. **Interacting Concepts**





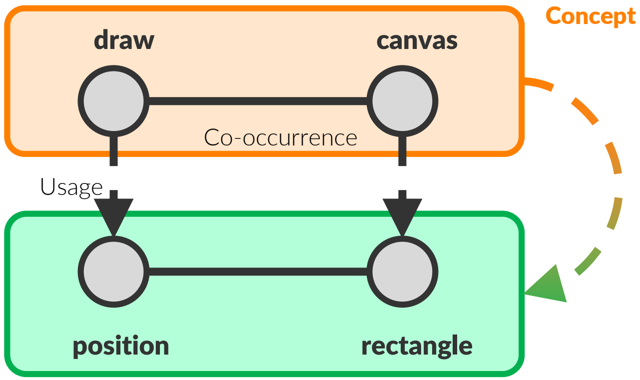
Graph-based Semantic Models

Nodes are **names**. **Edges** indicate they **co-occurr** in close proximity.



Challenges

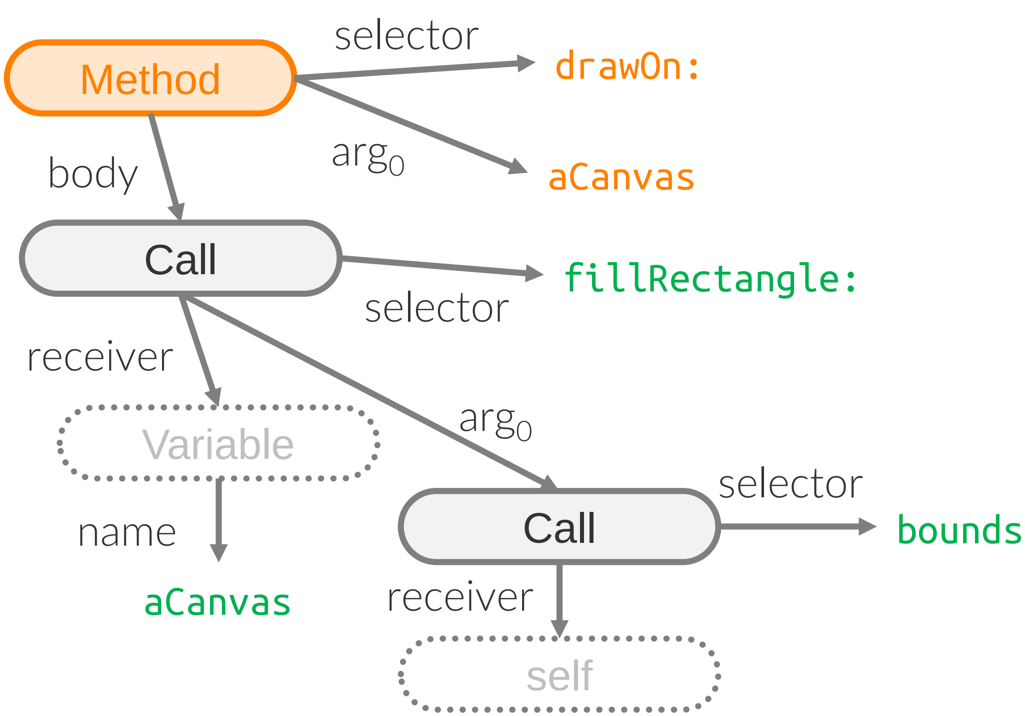
- Graph construction
  - From source code
  - From version history
  - From run-time data
- Concept inference
  - Graph clustering
  - Probabilistic models



Relating Names

```
Morph » drawOn: aCanvas
  aCanvas fillRectangle: self bounds.
```

AST (Abstract Syntax Tree)

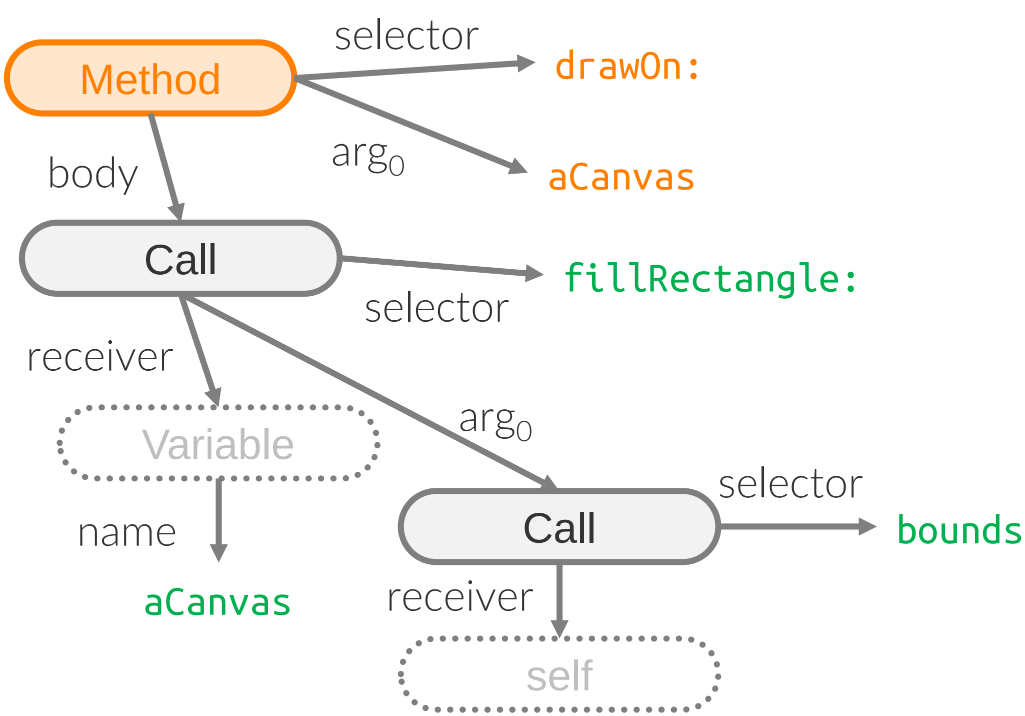


Relating Names

```
Morph » drawOn: aCanvas

  aCanvas fillRectangle: self bounds.
```

AST (Abstract Syntax Tree)



Abstracting nodes
(define new names)

Method
Assign
...

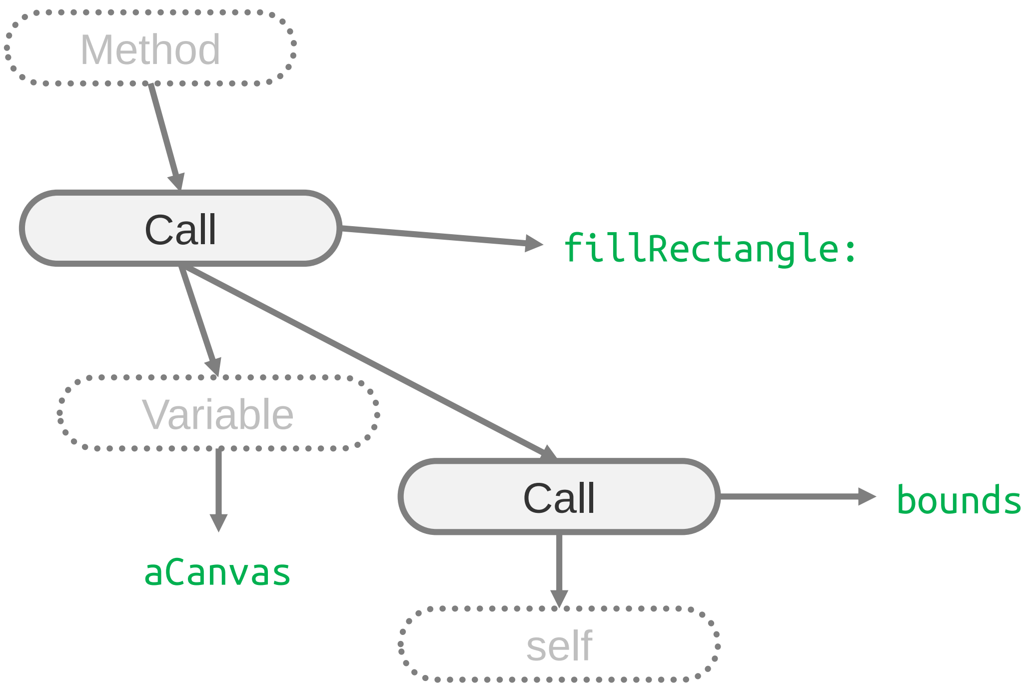
Combining nodes
(uses defined names)

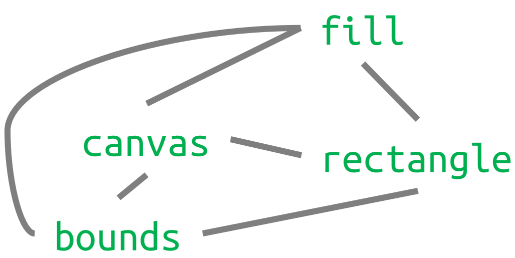
Call
Sequence
...

Relating Names

Combining nodes generate undirected edges

AST (Abstract Syntax Tree)

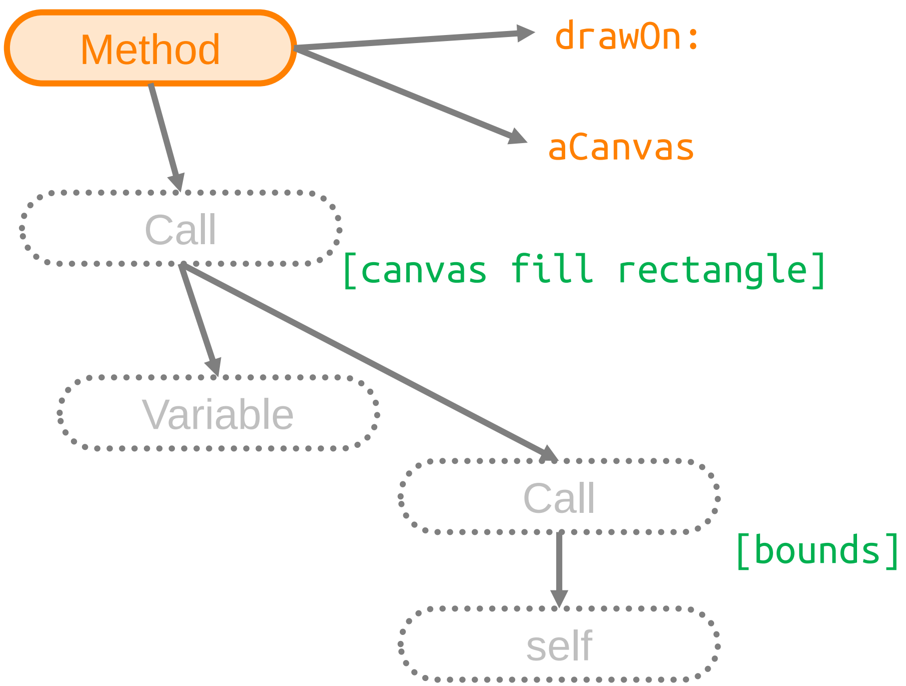


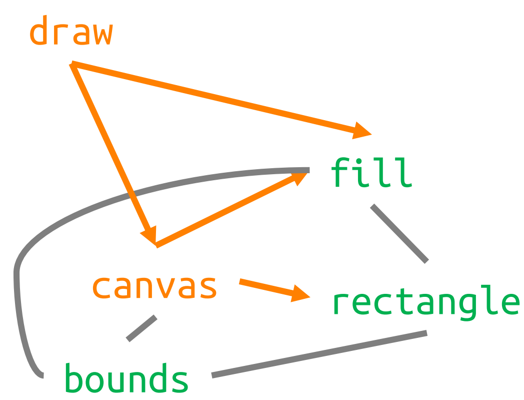


Relating Names

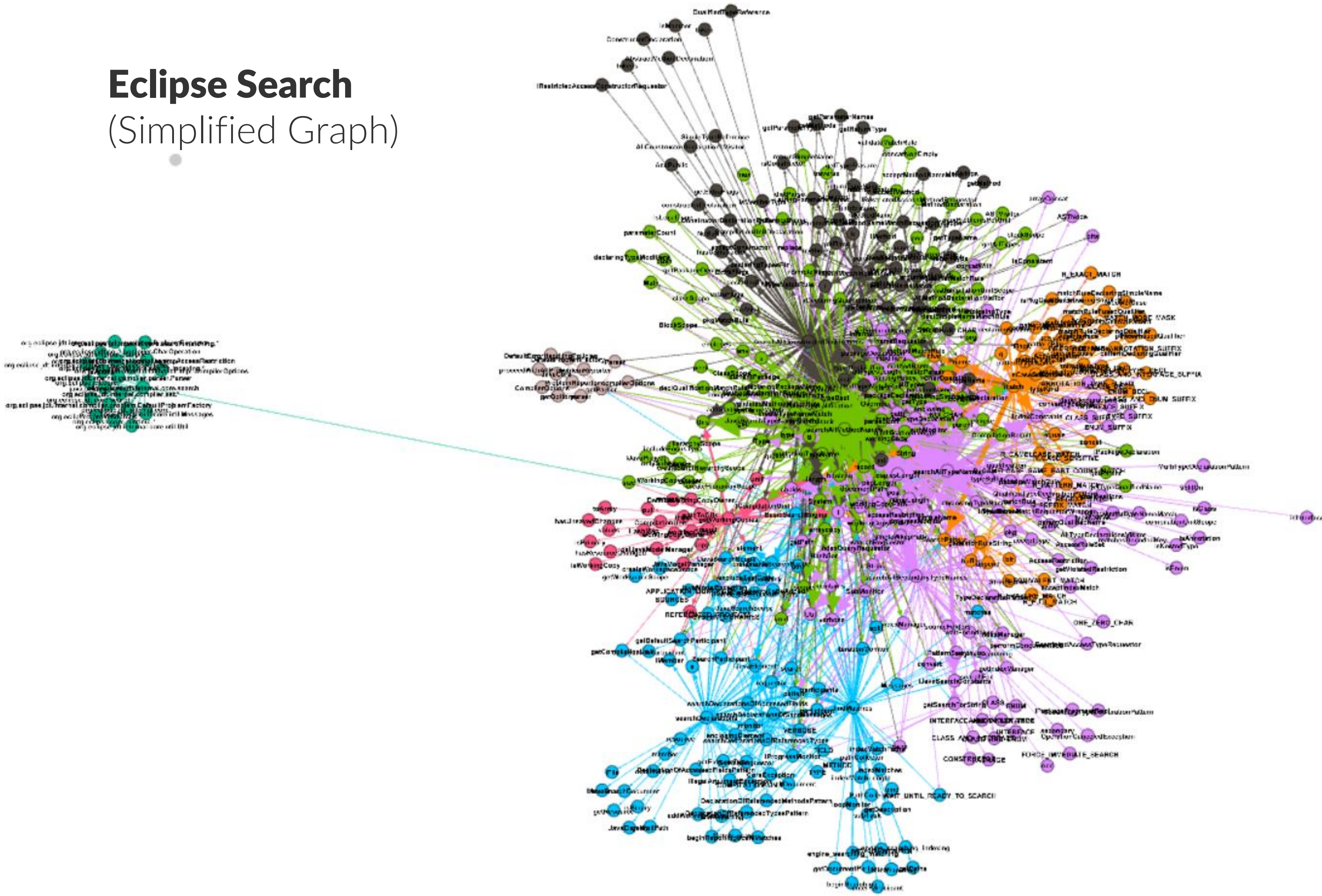
Abstracting nodes generate directed edges

AST (Abstract Syntax Tree)



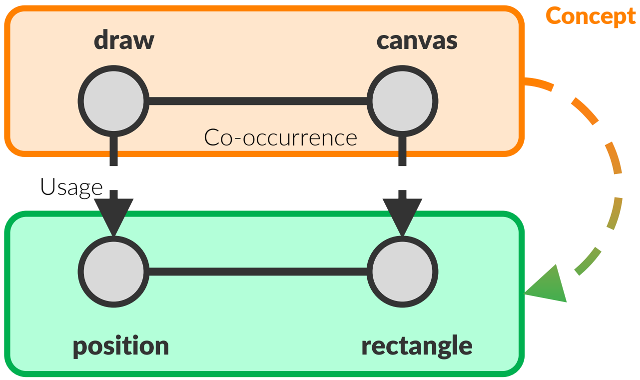


Eclipse Search
(Simplified Graph)

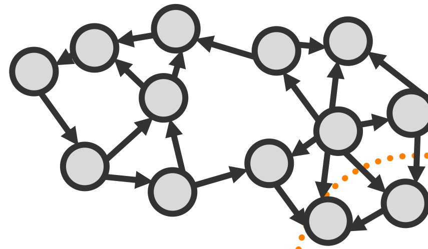


Challenges

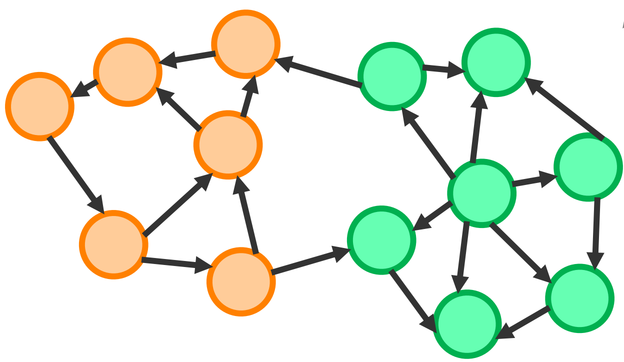
- Graph construction
  - From source code ✓
  - From version history
  - From run-time data
- Concept inference
  - Graph clustering
  - Probabilistic model



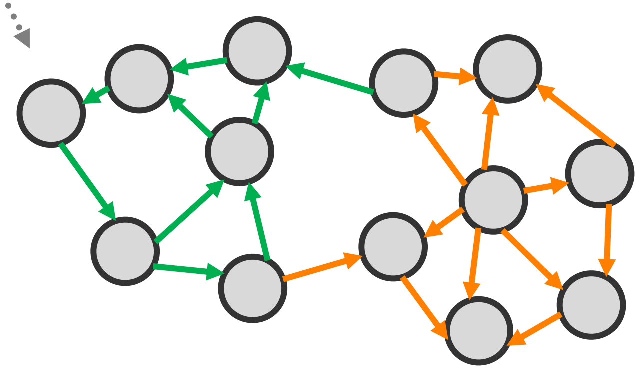
Concept Mining as Clustering Problem

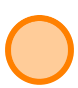


Node Clustering



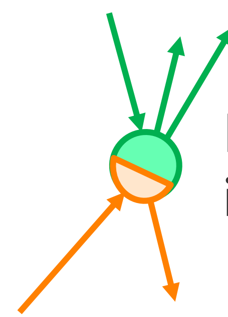
Edge Clustering





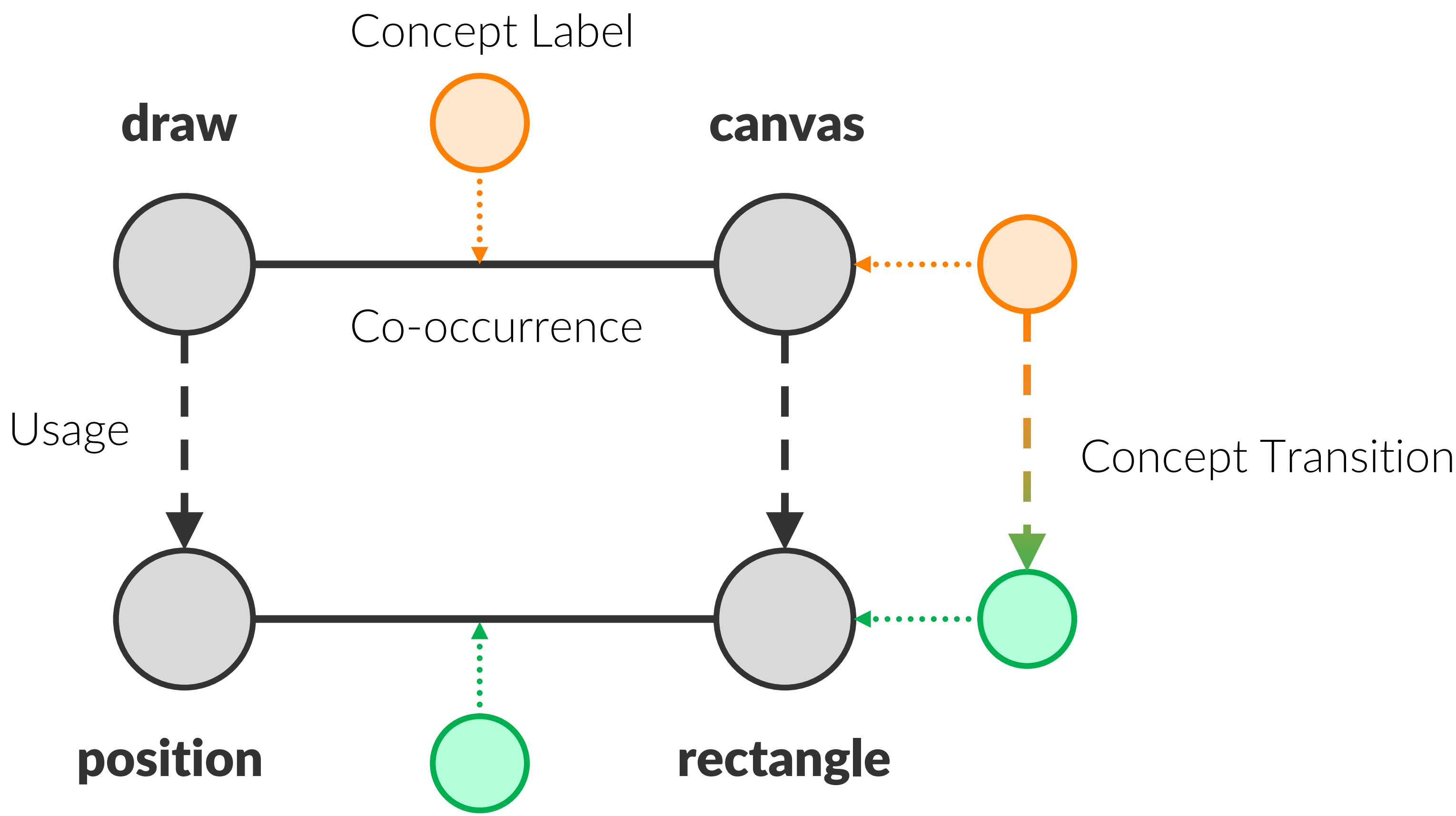
Single cluster per Node

Problem: **Ambiguity** in Names

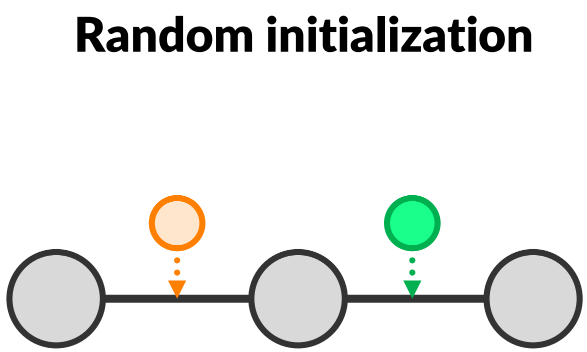


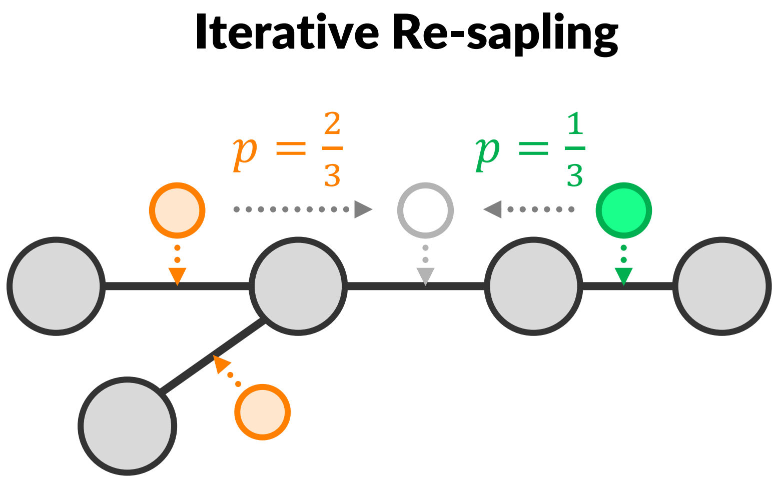
Node can participate in **multiple clusters**

Graph-based Semantic Models



Clustering Edges (Gibbs Sampling)





1. Decide on maximum number of concepts
2. Uniformly assign a concept to each edge
3. Re-assign each edge until convergence (beware of **local optima**)

Dirichlet-Multinomial Model

$p(\bigcirc) = \frac{2}{3}$ $p(\bigcirc) = \frac{1}{3}$

Sample:

(Multinomial)

α α α

Prior:

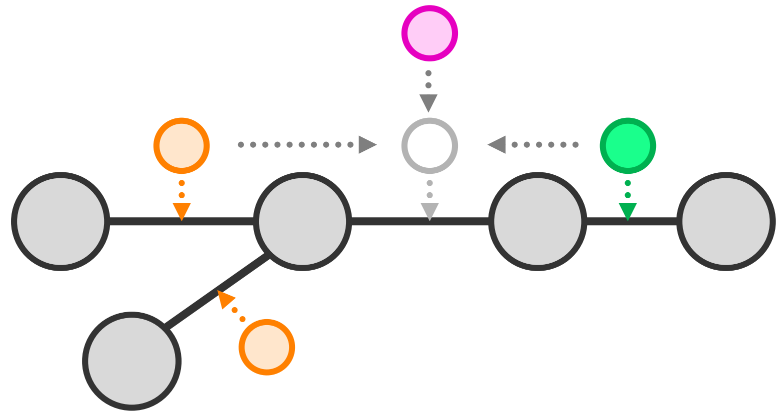
(Dirichlet)

Posterior:

$p(\bigcirc) = \frac{2+\alpha}{3+3\alpha}$ $p(\bigcirc) = \frac{1+\alpha}{3+3\alpha}$ $p(\bigcirc) = \frac{\alpha}{3+3\alpha}$

(Multinomial)

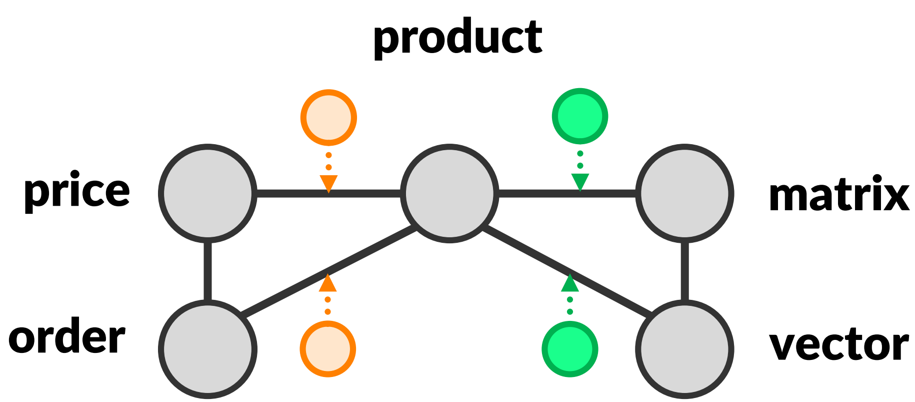
Iterative re-sampling



Disambiguating Names

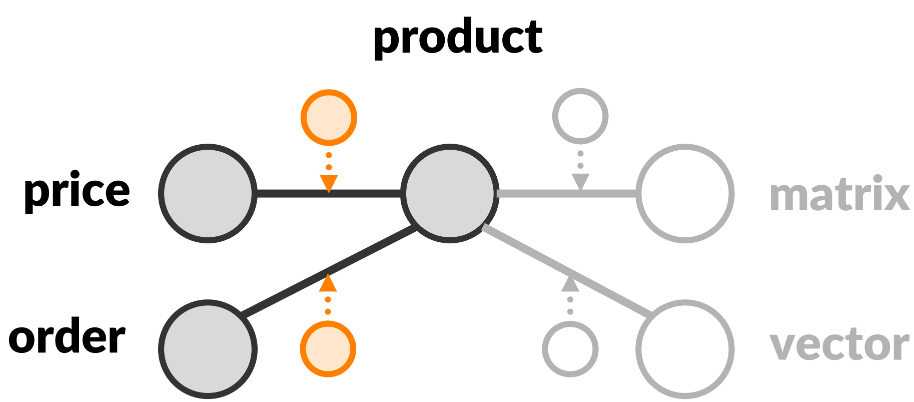
« product »

```
order.total += product.price;          product = matrix * vector;
```



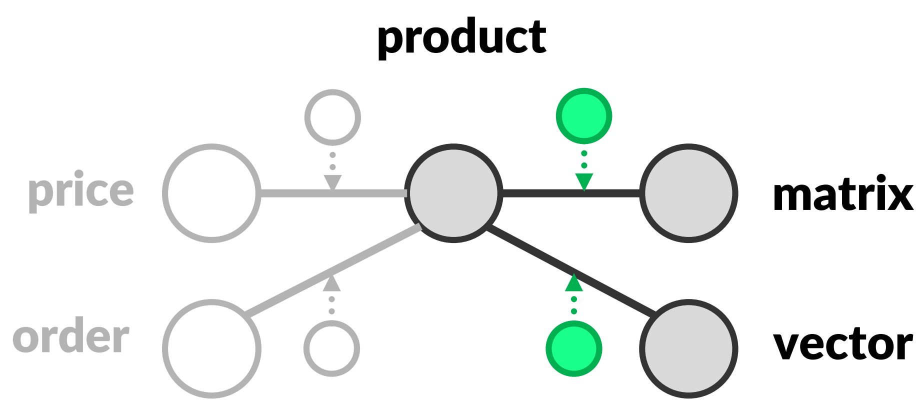
Disambiguating Names

```
order.total += product.price;        product = matrix * vector;
```

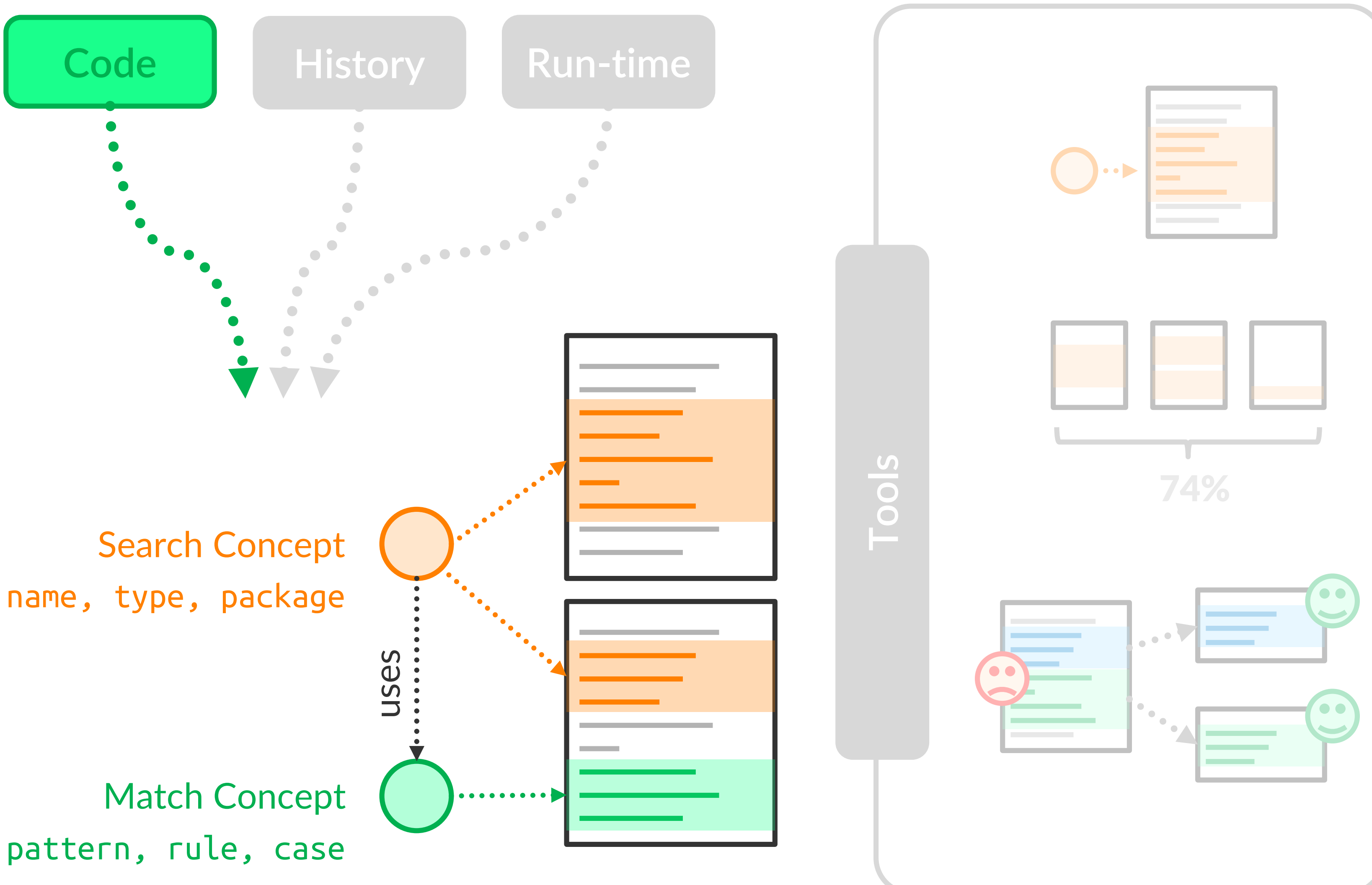


Disambiguating Names

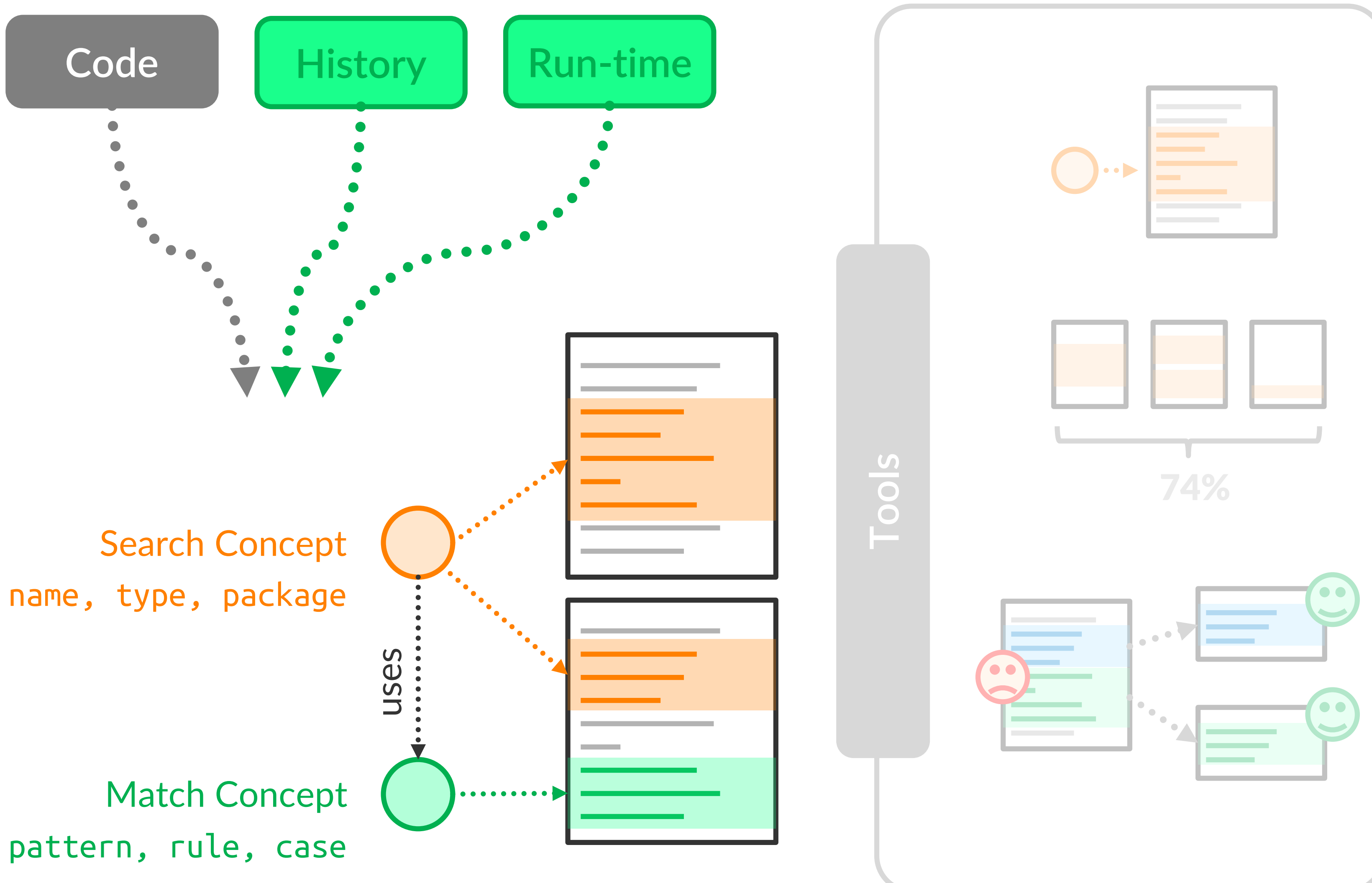
```
order.total += product.price;          product = matrix * vector;
```



Approach: Repository Mining



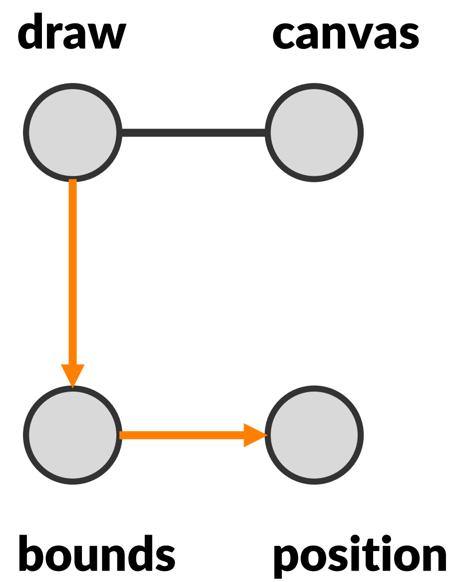
Approach: Repository Mining



Integrating Run-time Data

Call Stack

```
Canvas » draw:
Morph » drawOn:
Morph » bounds
Rectangle » position
```

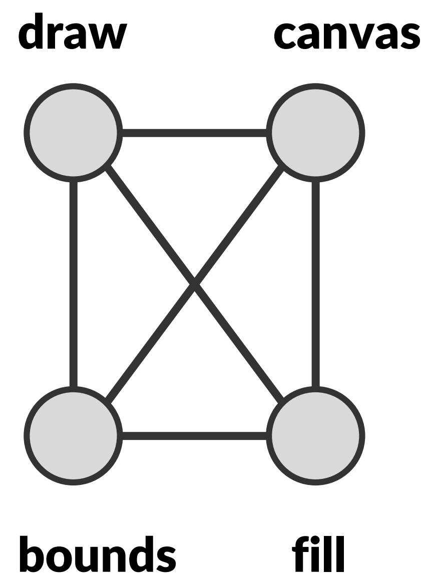


Integrating Program Evolution

Git Commit (Diff)

```
Canvas » draw: anObject
- self fillRect: anObject bounds.
+ anObject drawOn: self.
```

```
Morph » drawOn: aCanvas
- aCanvas draw: self.
+ aCanvas fillRect: self bounds.
```

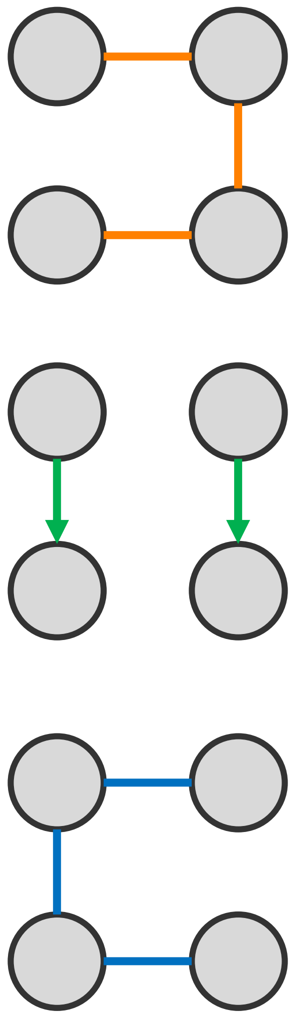


Multi-view Concepts

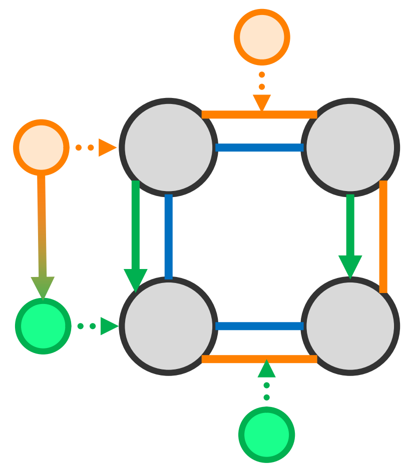
Co-located Names

Call Stack

Git Commit (Diff)

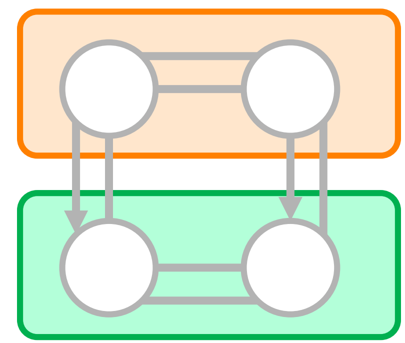


(Multi-)**Graph**



Concept Labeling

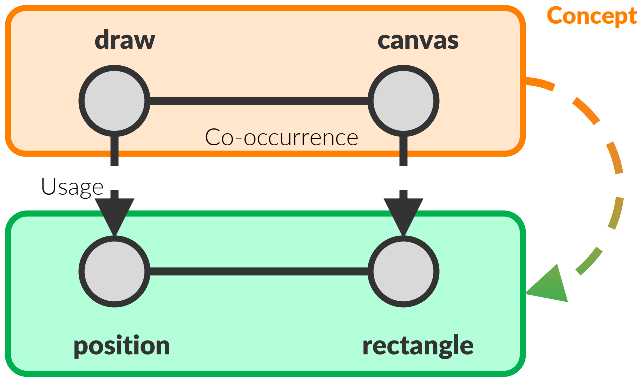
Concept Distribution



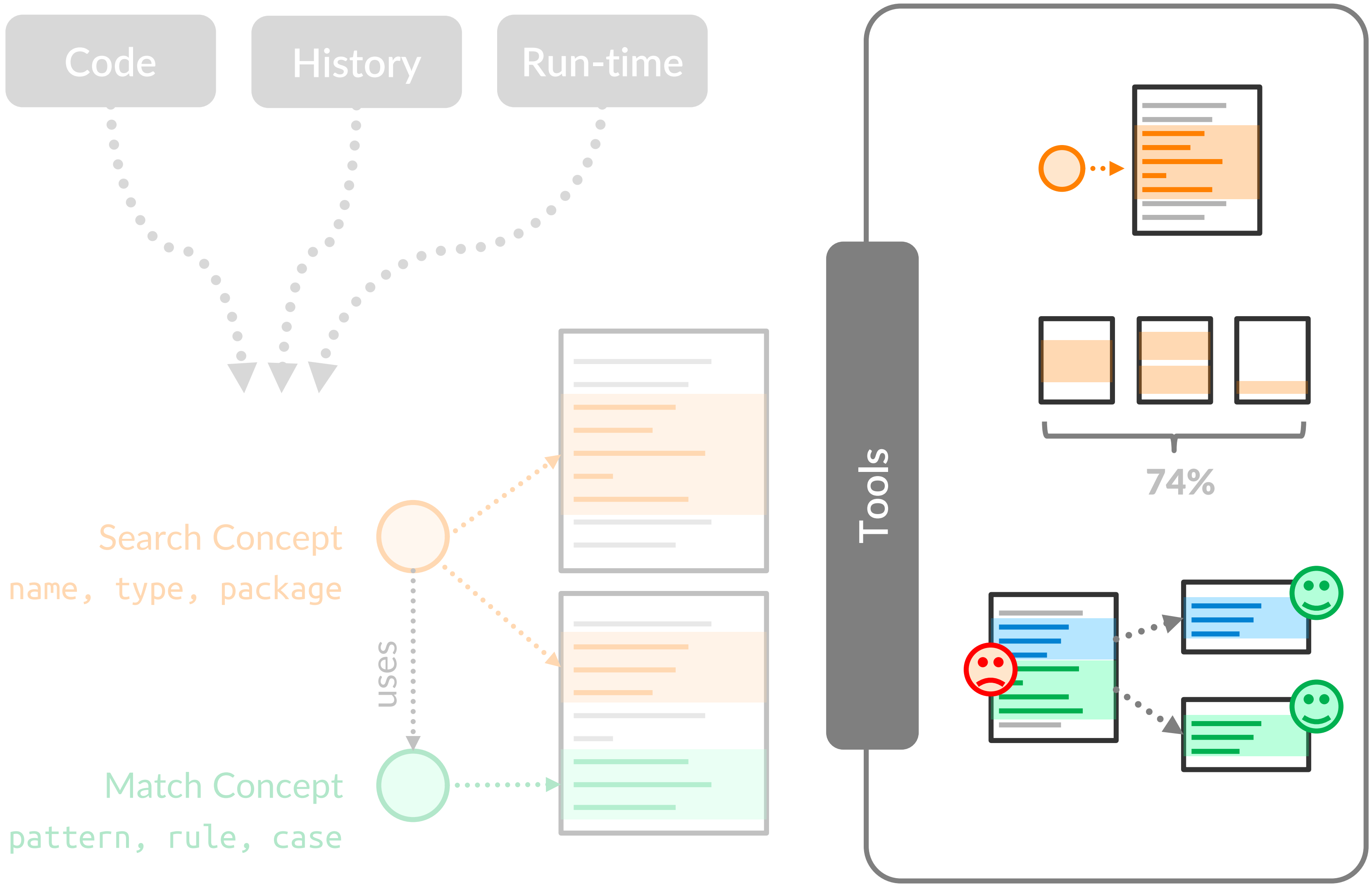
Challenges

» Graph construction
  › From source code ✓
  › From version history ✓
  › From run-time data ✓

» Concept inference ✓
  › Graph clustering
  › Probabilistic models



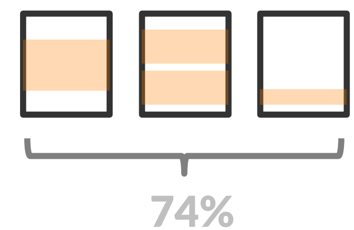
Approach: Repository Mining



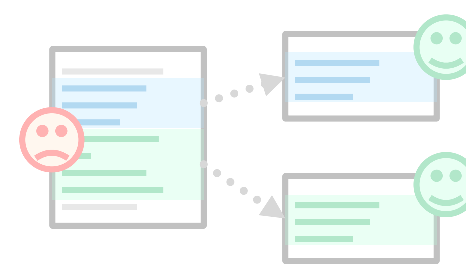
Goals

» **Reverse Engineering:** Help programmers understand the conceptual structure of a large system

» **Metrics:** Quantify how architecture deviates from conceptual structure



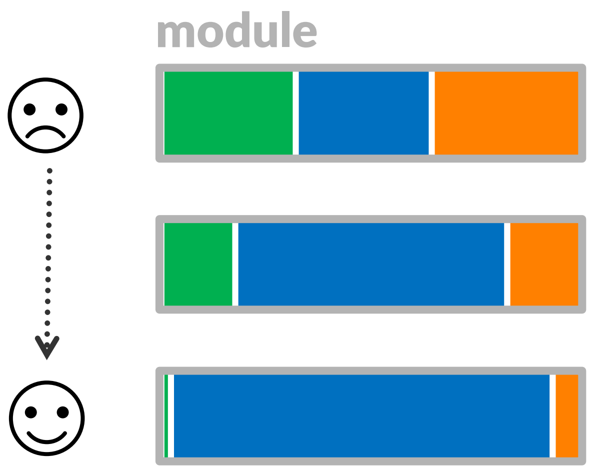
» **Forward Engineering:** Maintain and improve modularity by real-time feedback and recommendations



Modularity Metrics

module entropy:

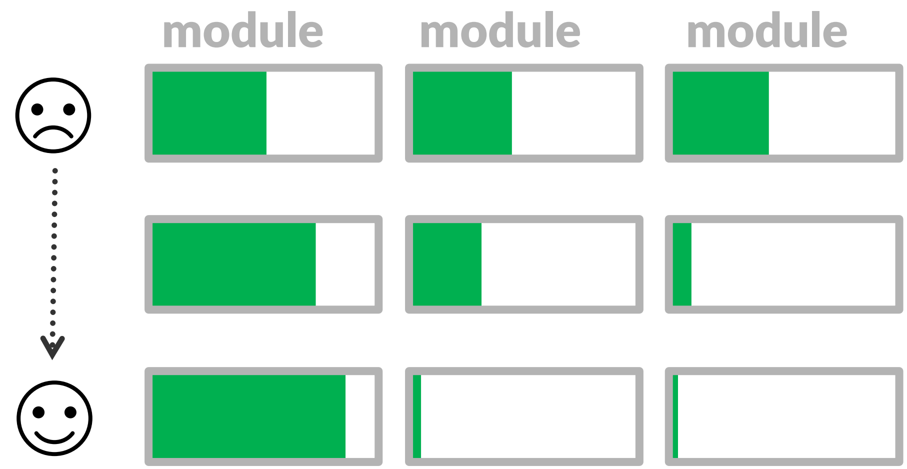
tangling



$$H(m) = -\sum_{c} p(c|m) \log_2 p(c|m)$$

concept entropy:

scattering



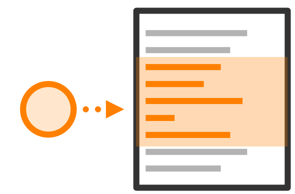
$$H(c) = -\sum_{m} p(m|c) \log_2 p(m|c)$$

…high values indicate need for refactoring or cross-cutting concerns

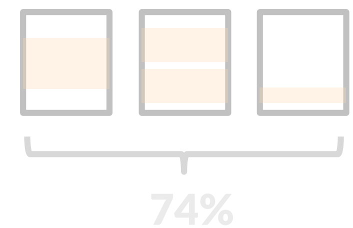
E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi, “Mining Concepts from Code with Probabilistic Topic Models,” *ASE*, 2007

Goals

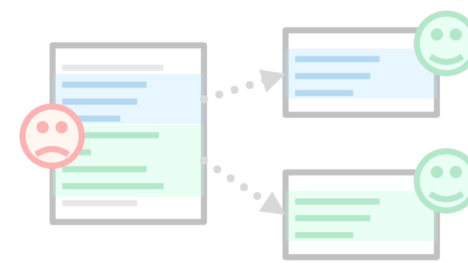
- » **Reverse Engineering:** Help programmers understand the conceptual structure of a large system



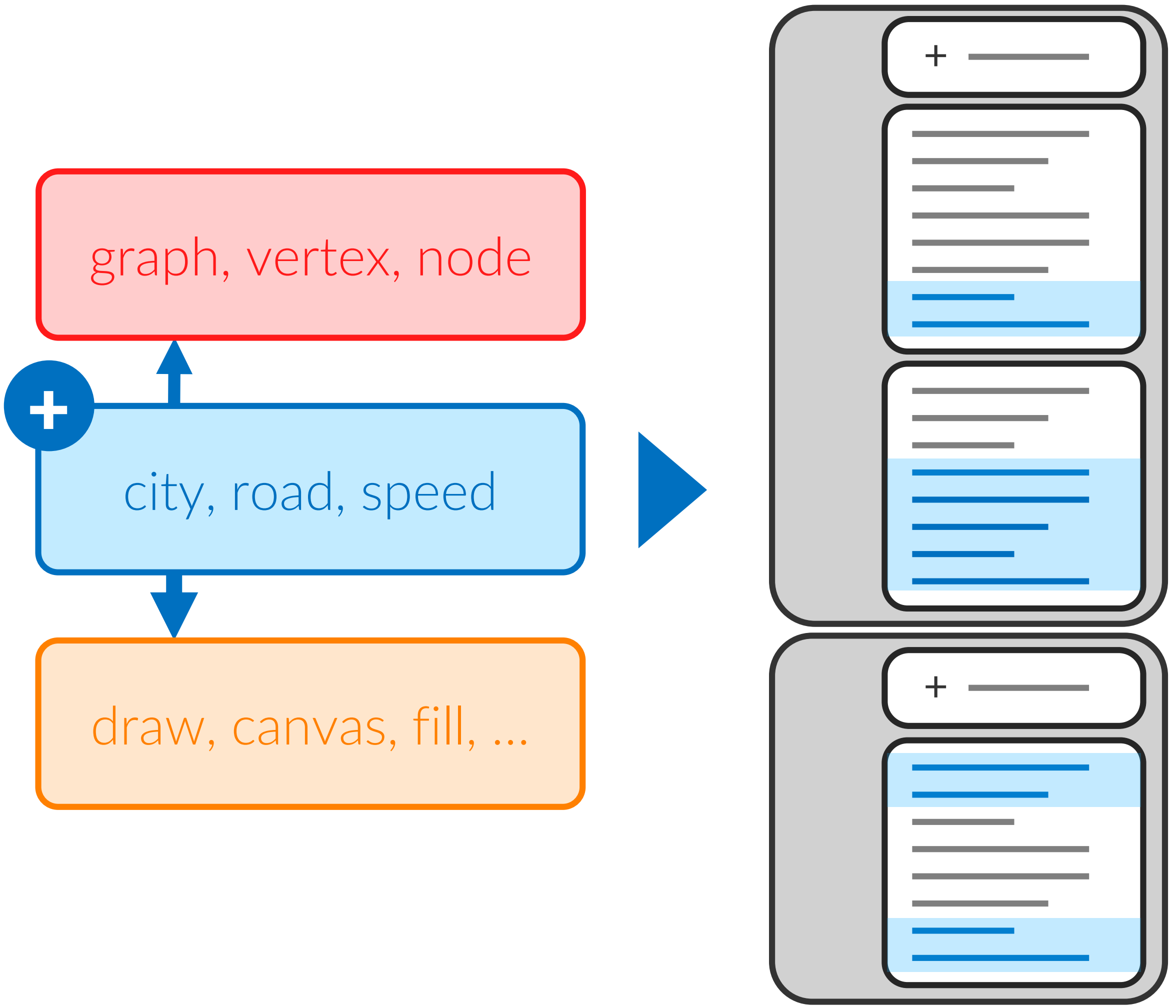
- » **Metrics:** Quantify how architecture deviates from conceptual structure



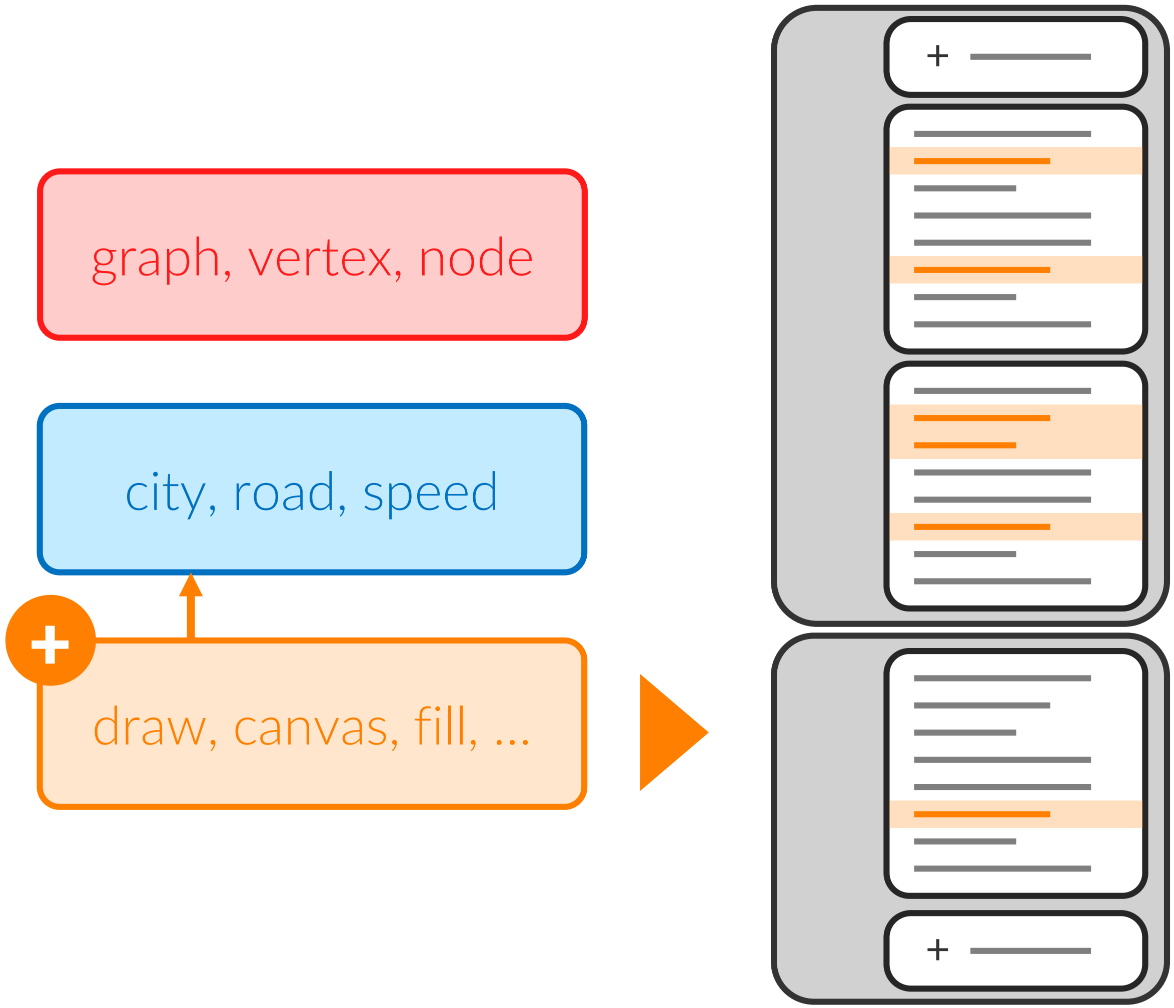
- » **Forward Engineering:** Maintain and improve modularity by real-time feedback and recommendations



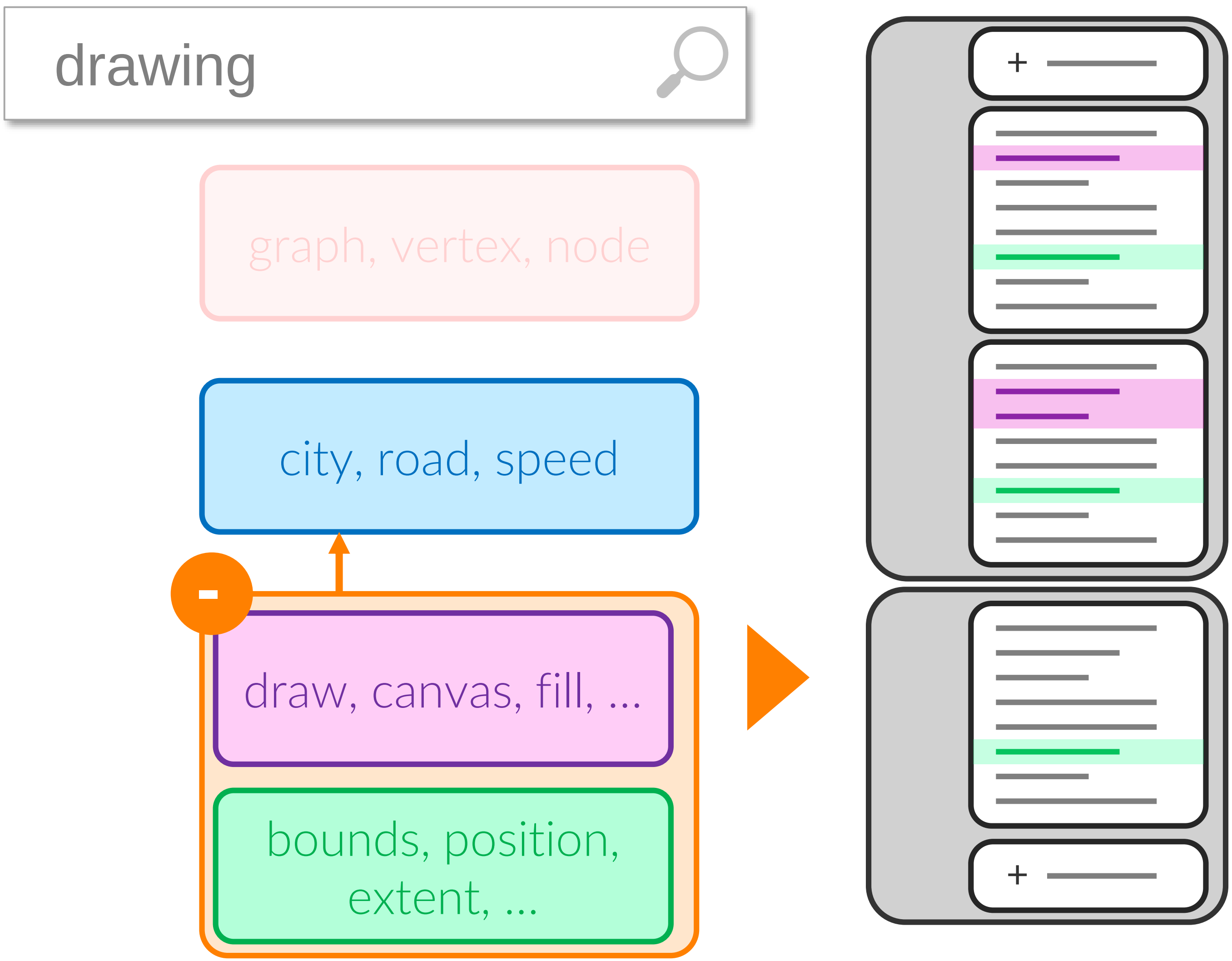
Exploring the Concept Graph



Exploring the Concept Graph

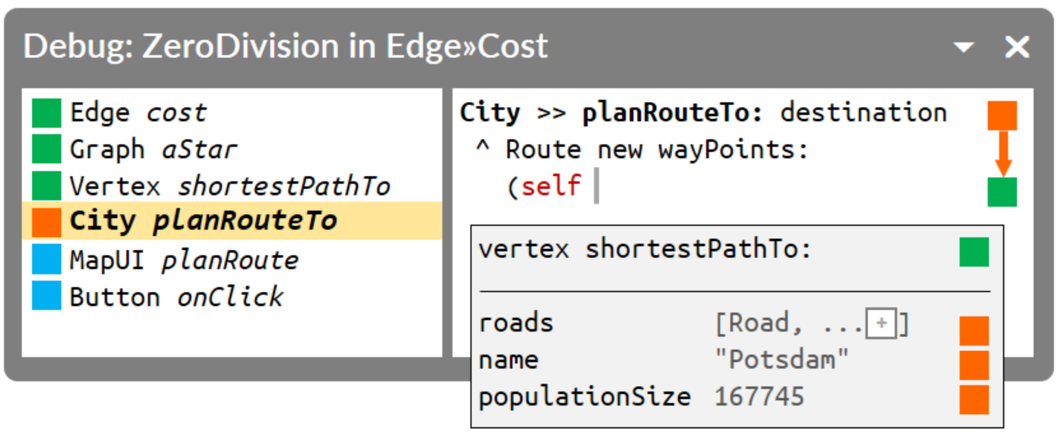


Exploring the Concept Graph



Concept-aware Tooling

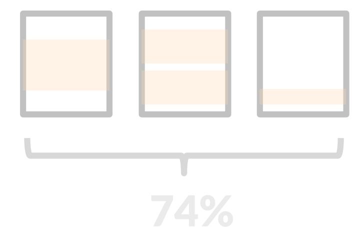
- Improve relevance of information displayed during
  - code completion
  - debugging



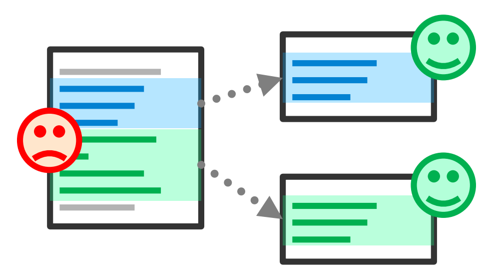
Goals

- **Reverse Engineering:** Help programmers understand the conceptual structure of a large system

- **Metrics:** Quantify how architecture deviates from conceptual structure



- **Forward Engineering:** Maintain and improve modularity by real-time feedback and recommendations



Forward Engineering

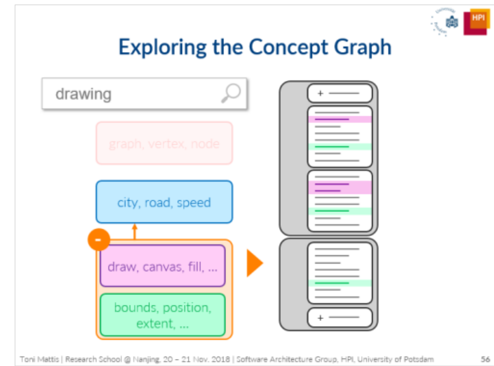
1. Awareness can help programmers to fix modularity issues before incurring **technical debt**
   - Metrics (Linting, Continuous Integration, …)
   - High-level overviews

2. Environments can **suggest** modularity improvements
   - Highlight ambiguous names, duplication, misplaced code
   - Recommend refactorings
   - Recommend names

Next Steps

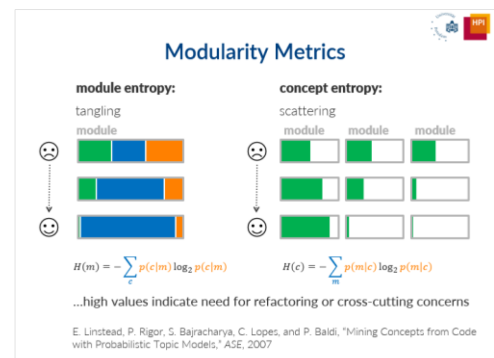
Tooling
(Qualitative evaluation)

- Equip programming environments with the capabilities to show, highlight, navigate by, filter by, and search for concepts



Analyzing repositories
(Quantitative evaluation)

- Measure architectural drift on large-scale projects
- Evaluate semantic code models on standard recommendation and clustering tasks



Open Questions

- How do our tools need to look like to keep programmers **aware of modularity issues** without distracting them?
- How can we balance the trade-off between **automated** (potentially surprising) and **manual** concept maintenance?
- How can the proposed concept model be maintained **collectively**?

Conclusion

1 Graph-based concept models can unify code, run-time, and evolutionary views on the program

2 Programming environments can be extended to include concept navigation, retrieval, and editing

3 Concept-aware environments have the potential to improve modularity during forward engineering