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

```bash
grep -r -i --include \*\.java "search"
```
Goal (1)

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

**Goal (1)**

**Goal: Concept Navigation**

- **Search Concept**
  - name, type, package

- **Matching Concept**
  - pattern, rule, case
Goal (1)

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

**Search Concept**
- `org.eclipse.jdt/.../SearchProvider.java`
- `org.eclipse.jdt/.../Editor.java`

**Matching Concept**
- Name, type, package
- Pattern, rule, case

**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

- **Code**
- **History**
- **Run-time**

**Search Concept**
- name, type, package

**Match Concept**
- pattern, rule, case

**Tools**

74%
Approach: Repository Mining

Search Concept
name, type, package

Match Concept
pattern, rule, case

74%
Concept Model

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

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

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

**concept labels**
which concept a name belongs to

**concepts**
prevalent names
Composition & Abstraction Barriers

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

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

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

**draw**, **canvas**, **fill**, ...

**uses**
(implemented through)

**bounds**, **position**, **extent**, ...
Recap: **Topic Models**

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

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

---

Toni Mattis | Research School @ Nanjing, 20 – 21 Nov. 2018 | Software Architecture Group, HPI, University of Potsdam
Recap: Topic Models
Recap: Topic Models

Semantically related words appear **correlated** (Distributional Hypothesis)
Recap: Topic Models
Recap: Topic Models

topics

terms sharing a common distribution

draw
fill
canvas

vertex
edge
graph

bounds
position
extent

Toni Mattis | Research School @ Nanjing, 20 – 21 Nov. 2018 | Software Architecture Group, HPI, University of Potsdam
Limitations of Classical Topic Models

**Flat Documents** vs. **Hierarchy of Code**

**Document Independence** vs. **Module Dependencies**

**Topical Independence** vs. **Interacting Concepts**

- Draw, canvas, fill
- Is defined using bounds, position, extent, ...

---

Toni Mattis | Research School @ Nanjing, 20 – 21 Nov. 2018 | Software Architecture Group, HPI, University of Potsdam
Graph-based Semantic Models

Nodes are names. Edges indicate they co-occur 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)

- **Method**
  - selector: drawOn:
  - arg0: aCanvas

- **Call**
  - selector: fillRectangle:
  - arg0: bounds

**Abstracting nodes**
(define new names)
- Method
- Assign

**Combining nodes**
(uses defined names)
- Call
- Sequence

Toni Mattis | Research School @ Nanjing, 20 – 21 Nov. 2018 | Software Architecture Group, HPI, University of Potsdam
Relating Names

Combining nodes generate undirected edges

**AST** (Abstract Syntax Tree)

Method

Call

fillRectangle:

Variable

aCanvas

Call

bounds

self

fill
canvas

rectangle

bounds
Relating Names

Abstracting nodes generate directed edges

**AST** (Abstract Syntax Tree)

- Method
  - drawOn:
    - aCanvas
  - [canvas fill rectangle]

- Variable
- Call
  - [bounds]

- Call
  - self

- canvas
  - rectangle
  - bounds
  - fill
  - draw
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

- Single cluster per Node

Problem: **Ambiguity** in Names

Edge Clustering

- Node can participate in **multiple clusters**
Graph-based Semantic Models

Concept Label

Co-occurrence

Usage

Concept Transition
Clustering Edges (Gibbs Sampling)

Random initialization

Iterative Re-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

$\text{Sample: } \begin{align*}
p(\bigcirc) &= \frac{2}{3} \\
p(\bigotimes) &= \frac{1}{3}
\end{align*}$

$\text{Prior: } \begin{align*}
\alpha &\quad \alpha &\quad \alpha
\end{align*}$

$\text{(Dirichlet)}$

$\text{Iterative re-sampling}$

$\text{Posterior: } \begin{align*}
p(\bigcirc) &= \frac{2+\alpha}{3+3\alpha} \\
p(\bigotimes) &= \frac{1+\alpha}{3+3\alpha} \\
p(\bullet) &= \frac{\alpha}{3+3\alpha}
\end{align*}$

$\text{(Multinomial)}$
Disambiguating Names

```
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

Search Concept
name, type, package

Match Concept
pattern, rule, case

Tools

74%
Approach: Repository Mining

Search Concept
name, type, package

Match Concept
pattern, rule, case

Tools
74%
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 Distribution

Concept Labeling
Challenges

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

» Concept inference ✓
  › Graph clustering
  › Probabilistic models
Approach: Repository Mining

- Code
- History
- Run-time

Search Concept
- name, type, package

Match Concept
- pattern, rule, case

Tools

74%
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

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

graph, vertex, node

city, road, speed

draw, canvas, fill, ...

+ ———+
   |
   +——+

+ ———+
   |
   +——+

+ ———+
   |
   +——+

Toni Mattis | Research School @ Nanjing, 20 – 21 Nov. 2018 | Software Architecture Group, HPI, University of Potsdam
Exploring the Concept Graph

city, road, speed

draw, canvas, fill, ...

graph, vertex, node
Exploring the Concept Graph

drawing

graph, vertex, node

- draw, canvas, fill, ...

bounds, position, extent, ...

- city, road, speed

+ ————
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 refactoring
   - Recommend names

![Diagram showing forward engineering process]
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.