Towards

Concept-aware Programming Environments
for Guiding Software Modularity

Toni Mattis, Patrick Rein, Stefan Ramson,
Jens Lincke, Robert Hirschfeld

Software Architecture Group
Hasso Plattner Institute, University of Potsdam, Germany

PX/17.2 22 Oct. 2017, Vancouver
metaphor
deriving names

projection
(into code)

extrapolation
(from code)

metaphor
mental shortcut
Problem Statement

With growing code bases...

» Concepts tend to **scatter** and **entangle**
» Programmers need **more time** to recognize concepts
Problem Statement

With growing code bases...
» Concepts tend to scatter and entangle
» Programmers need more time to recognize concepts

Consequences of incomplete recognition
» Architectural Drift: Code written in the wrong place
» Duplication: Missed existing functionality
» Inconsistent Naming: Metaphor misunderstood

Approaches
» Proactive: Tools/Language features to maintain concepts
  [ e.g. AOP/COP/ ... discipline during development! ]
» Retroactive: Tools to recover concepts
» Proactive: Tools to support concept maintenance
Basic Concept Model

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

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

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

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

**concepts**
prevalent names

- `draw, canvas, fill, ...`

- `bounds, position, extent, ...`
Names

» Typical **identifiers** can consist of **multiple names**

  › Camel Case:
    
    
    fillRectangle → fill, rectangle

  › Underscore:
    
    fill_rectangle → fill, rectangle

  › Acronyms:
    
    HTTPServer → http, server

  › Multi-part message names:
    
    fillRectangle:color: → fill, rectangle, color

» **Constant strings** (or symbols) can be relevant, too:

  › config['backgroundColor']

  › config at: #backgroundColor
    
    → background, color
AST-based View

**AST**
Abstract Syntax Tree

**Expanded Identifiers**

```
"camelCase"
"camel"
"case"
```

**Concept AST**

```
aggregated concept distribution
```

**concept labels**
Maintaining Concepts

1. Automated bootstrap phase ("concept mining")
   - Deciding which names belong to **the same / a different concept**
   - Setting the **granularity**
   - Selecting useful **data/features**

2. User refinements
   - Types of **operations** provided to users
   - (Partially) **re-running** concept mining
   - **Synchronizing** refinements between team members
Maintaining Concepts

1. Automated bootstrap phase ("concept mining")
   - Deciding which names belong to the same / a different concept
   - Setting the initial granularity
   - Selecting useful data/features

2. User refinements
   - Types of operations provided to users
   - (Partially) re-running concept mining
   - Synchronizing refinements between team members
Distributional Hypothesis

» Lexical tokens with a similar distribution have a similar meaning

\[
f(A \land B) = \frac{3}{16}
\]

\[
f(A \land C) = \frac{7}{16}
\]

\[
E[f(A \land B)] = \frac{4}{16} \text{ (if both were random)}
\]
Names belonging to the same concept co-occur more frequently in the same scope.

\[ \hat{f}(a, b) = \frac{f(a \land b)}{f(a)f(b)} = \ldots \]

- \( \approx 0.0 \) (mutually exclusive)
- \( \approx 1.0 \) (random)
- \( \gg 1.0 \) (intentional?)
# Co-occurrence

## Examples (Squeak/Smalltalk Image)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>$\hat{f}(a,b)$</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>visit</td>
<td>accept</td>
<td><strong>70.1</strong></td>
<td>same design pattern</td>
</tr>
<tr>
<td>bounds</td>
<td>draw</td>
<td><strong>15.4</strong></td>
<td>geometry &amp; drawing</td>
</tr>
<tr>
<td>collect</td>
<td>select</td>
<td><strong>6.8</strong></td>
<td>same API</td>
</tr>
<tr>
<td>parse</td>
<td>next</td>
<td><strong>2.2</strong></td>
<td>parsing &amp; streams</td>
</tr>
<tr>
<td>collect</td>
<td>color</td>
<td><strong>1.5</strong></td>
<td>incidental</td>
</tr>
<tr>
<td>visitor</td>
<td>color</td>
<td><strong>0.0</strong></td>
<td>mutually exclusive</td>
</tr>
</tbody>
</table>
Co-occurrence Relations

» Same module
  › class, **method**, package
  › file
  › lexical scope

» Within certain distance
  › ... **in the AST**
  › ... in text
  › ... in execution

» Edited close in time
  › Git commits
  › IDE interactions
Concept Mining

» Clustering
  › Maximize intra-cluster **similarity**
  › Minimize inter-cluster **similarity**
  › **One concept per name**

» Mixture Models
  › Every name has a **probability** of occurring in each concept
    › **Bag-of-words (Topic Models)**
    › **Graph-based (Stochastic Block Models)**

**co-occurrence,**
pointwise mutual information, cosine similarity, ...
Topic Models

60% + 40%

topics
terms sharing a common distribution
AST-based View

**AST**
Abstract Syntax Tree

**Expanded Identifiers**

**Concept AST**

Name Extraction

Training

Mixture Model

"camelCase"

"camel"

"case"

aggregated concept distribution

concept labels
(most likely topic)
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.

uses (implemented through)

draw, canvas, fill, ...

bounds, position, extent, ...
Composition & Abstraction Barriers

**abstraction** (concepts being defined)

Morph » drawOn: aCanvas

aCanvas fillRectangle: self bounds.

**implementation** (defining concepts)

concepts use in their implementation
Composition & Abstraction Barriers

**abstraction** (concepts being defined)

Morph » `drawOn: aCanvas`

`aCanvas fillRectangle: self bounds`.

**implementation** (defining concepts)

`draw, canvas, fill, ...`

`bounds, position, extent, ...`
Concept Graph

city, road, speed

draw, canvas, fill, ...

bounds, position, extent, ...

graph, vertex, node
Maintaining Concepts

1. Automated bootstrap phase ("concept mining")
   - Deciding which names belong to the same / a different concept
   - Setting the initial granularity
   - Selecting useful data/features

2. User refinements
   - Types of operations provided to users
   - (Partially) re-running concept mining
   - Synchronizing refinements between team members
Operations on Concepts

Reassign concept label:

Morph » drawOn: aCanvas
aCanvas fillRectangle: self bounds.

Challenges

» Inconsistencies
  › Re-computing clustering might avalanche into many other concepts being re-assigned to restore optimality
  › Not doing so might leave programmers with lots of manual re-assignment work

» Synchronization
  › Share manual refinements across a team of programmers
Operations on Concepts

Merge concepts

draw, canvas, fill, ...

bounds, position, extent, ...

Split concepts

bounds, canvas position, fill, ...

position, x, y, ...

bounds, extent, origin, ...

› Optimal Split
› Hierarchical Topic Model
› Manual / Semi-supervised
Exploring the Concept Graph

graph, vertex, node

city, road, speed

draw, canvas, fill, ...

Mattis, Rein, Ramson, Lincke, Hirschfeld | PX/17.2, Vancouver, 2017 | Software Architecture Group, HPI Potsdam
Exploring the Concept Graph

graph, vertex, node

city, road, speed

draw, canvas, fill, ...
Exploring the Concept Graph

- graph, vertex, node
- city, road, speed
- draw, canvas, fill, ...
- bounds, position, extent, ...

Mattis, Rein, Ramson, Lincke, Hirschfeld | PX/17.2, Vancouver, 2017 | Software Architecture Group, HPI Potsdam
Concept-aware Tooling

» Highlight concepts in Code

Editor

Canvas » draw: anObject
  ^ anObject drawOn: self

Morph » drawOn: aCanvas
  aCanvas fillRectangle: self bounds.

Morph » bounds: newBounds
  self position: newBounds topLeft;
  extent: newBounds extent.
Concept-aware Tooling

» Improve relevance of information displayed during
  › code completion
  › debugging

» In live programming
  › Arranging and prioritizing live objects and meta-objects
  › Live feedback on modularity, name choices, recommended code artifacts, ...
A Perspective on Modularity

module entropy:

- tangling

concept entropy:

- scattering

\[
H(m) = - \sum_c p(c|m) \log_2 p(c|m)
\]

\[
H(c) = - \sum_m p(m|c) \log_2 p(m|c)
\]

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

Counteracting Architectural Drift

Architectural Drift:
Conceptual model misaligned with module structure

- Quantifiable using *entropy over time*
- Motivates integration into *version control*
- **Hypothesis:** Awareness can help programmers to fix modularity issues before incurring *technical debt*
metaphor
deriving names

projection
(into code)

extrapolation
(from code)

metaphor
mental shortcut
metaphor
deriving names

projection
(into code)

metaphor
mental shortcut

maintenance
(understanding code &
keeping it modular)
Open Questions

» Which additional information needs can be assessed using our concept model?

» 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 assignment?

» How can the proposed concept model be maintained collectively?
Conclusion

1. First-class concepts are complementary to language features to manage concepts.

2. Existing tools can be extended to include concept information, new tools can navigate and manipulate concepts.

3. Concepts are not restricted to reverse engineering, but support modularity during forward engineering.