Metrics to Identify Where Object-Oriented Program Comprehension Benefits from the Runtime Structure

Marwan Abi-Antoun  Radu Vanciu  Nariman Ammar

Department of Computer Science
Wayne State University

Code structure is different from runtime structure

- Developers often need to understand both code and runtime structure
- Runtime structure is hard to understand from looking at code
- No good sense of how different code and runtime structure are
- We propose metrics to measure difference between code and runtime structure
- Many tools focus on code structure; tools for reverse engineering runtime structure are less mature
Hierarchy of classes vs. Hierarchy of objects

+-java
  +-util
    | +-Hashtable
    |   +-Entry
    | +-class
  +-package
    | +-class
    |   +-innerclass
+-root
  +- TLD1
    | +- object1:B
    |   +- MAPS
    |     +- hash1:Hashtable
    |     +- OWNED
    |       +- hash2:Hashtable
  +- TLD2
    | +- object2:B
    |   +- OWNED
    |       +- hash3:Hashtable
Hierarchy of classes vs. Hierarchy of objects

- Packages and classes
- Shows `Hashtable` once

- Objects and groups of objects
- Shows multiple objects of type `Hashtable`

- Using hierarchy of objects leads to **less time and effort** for some code modification tasks [Ammar and Abi-Antoun, WCRE'12]
Objects matter (in addition to types), but specific instances do NOT matter

- Merge objects that have same role
- Object with same type can have different roles
- Collapse objects underneath other objects

```
+- object1:B
  +- MAPS
    +- hash:Hashtable<String,String>
      +- KEYS
        +- key1: String
        +- key2: String
        +- key3: String
      +- VALUES
        +- val1: String
        +- val2: String
        +- val3: String
```
Objects matter (in addition to types), but specific instances do NOT matter

- Merge objects that have same role
- Object with same type can have different roles
- Collapse objects underneath other objects

```
+- object1:B
  +- MAPS
    +- hash:Hashtable<String, String>
      +- KEYS
        +- key: String
      +- VALUES
        +- val: String
```
Strategies for merging objects

- **Dynamic analysis:** hierarchical abstract heap [Marron et al., TSE’13]
- **Static analysis:** sound, hierarchical Ownership Object Graph (OOG) [Abi-Antoun and Aldrich, OOPSLA’09]
  - Extracted from code with annotations
  - **Abstract object** merges objects of same type and in same domain
  - **Domain** = named, conceptual group of objects
  - Edges are **points-to** relations - due to field references
Describe role of object by *type+group+hierarchy*

- **Object hierarchy in OOG**
  - Each object has domains,
  - Each domain has objects
- **Describe role of an object, not just by type, but by named groups (domains) or by position in object hierarchy**
- **Triplet \( \prec A, D, B \succ \):**
  - object of type \( A \)
  - in domain \( D \)
  - in parent object of type \( B \)
Measure differences code vs. runtime structure

- **One** class to **many** abstract objects
  - **Which-A-In-B**,  
  - Which-A-In-Which-B
- **Many** object creation expressions to **one** abstract object
  - **Object Scattering**
- Unexpected enclosing declarations for objects, domains and edges
  - **Pulled Objects**,  
  - Inherited Domains,  
  - Inherited Edges,  
  - Lifted Edges
- More precise edges based on **type+group+hierarchy**
  - **Edge precision**
Measure *Which-A-In-B*: count triplet pairs of same A, same B, different D

Hierarchy of objects:

- `root/`:
  - `TLD1/`:
    - `object1: B`:
      - `MAPS`:
        - `hash1: Hashtable`
      - `OWNED`:
        - `hash2: Hashtable`
  - `TLD2/`:
    - `object2: B`:
      - `OWNED`:
        - `hash3: Hashtable`

Vary the domain D: same A, same B, different D

object of Type B inside domain D

object of Type A
Measure *Which-A-In-Which-B*: count triplet pairs of same *A*, different *B*, different *D*

Hierarchy of objects

```
+-root/
 | +- TLD1/
 |   | +- object1: B
 |   |   | +- MAPS
 |   |   |   | +- hash1: Hashtable
 |   |   +- OWNED
 |   |       | +- hash2: Hashtable
 |   |       |   | +- OWNED
 |   |       |     | +- object2: B
 |   |       |     |   | +- OWNED
 |   |       |     |     | +- hash3: Hashtable
```

Vary the domain *D*: same *A*, same *B*, different *D*

Vary the parent object of type *B*: same *A*, different *B*, different *D*
Results: *Which-A-In-B* in MiniDraw

- MiniDraw: framework for board games
  1,500 LOC, 31 classes and 17 interfaces

**Table:** \( \langle A, D, B \rangle \) triplets from MiniDraw that satisfy the metric *
Which-A-in-B*, grouped by the raw type A.

<table>
<thead>
<tr>
<th>A</th>
<th>D</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ArrayList</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ArrayList&lt;FigureChangeListener&gt;</td>
<td>owned</td>
<td>BoardDrawing</td>
</tr>
<tr>
<td>ArrayList&lt;Figure&gt;</td>
<td>owned</td>
<td>BoardDrawing</td>
</tr>
<tr>
<td>ArrayList&lt;BoardFigure&gt;</td>
<td>MAPS</td>
<td>BoardDrawing</td>
</tr>
<tr>
<td><strong>HashMap</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HashMap&lt;Position, List&lt;BoardFigure&gt;&gt;</td>
<td>MAPS</td>
<td>BoardDrawing</td>
</tr>
<tr>
<td>HashMap&lt;String, BoardFigure&gt;</td>
<td>owned</td>
<td>BoardDrawing</td>
</tr>
</tbody>
</table>
Many creation expressions to one abstract object

- Object creation expressions for one abstract object are scattered in multiple declaring types.
- Object $scattering(O)$: number of distinct declaring types of $O$

$$scatteringFactor(O) = 1 - \frac{1}{scattering(O)}$$
Many creation expressions to one abstract object

Type declarations

class BreakthroughPieceFactory {
    p = new Position(row,col);
}

class MoveCommand{
    from = new Position( ... )
    to = new Position( ... );
}

Hierarchy of objects

+- main:Breakthrough
    +- MODEL
        | +- p:Position

• In MiniDraw:
  • 3 creation expressions
  • 2 enclosing type declarations
  • 1 abstract object
• \textit{scattering}(p:Position) = 2
• \textit{scatteringFactor}(p:Position) = 0.5
Enclosing type declaration vs. type of parent object

class Board {
    f = new Figure();
}
class Breakthrough{
    b = new Board();
}

\[<\text{Figure,DATA,Breakthrough}>\]
created in enclosing type Board

Parent of \(f:\text{Figure}\) is
main:Breakthrough
NOT board:Board

\[+- \text{main:Breakthrough}\]
\[+- \text{CTRL}\]
\[| ++ \text{board:Board}\]
\[| ++ f:\text{Figure}\]
Precision of points-to edges

- More precise edges between objects
- Edge identified by pair of triplets and field name

points-to edge $E_f : \langle \prec A_{src}, D_{src}, B_{src} \succ, \prec A_{dst}, D_{dst}, B_{dst} \succ, f \rangle$

$$\text{precision}(E_f) = 1 - \frac{\text{concrete subtypes } A_{dst} \text{ in OOG}}{\text{all possible subtypes of type } C \text{ of field } f}$$

class Asrc {
  C f;
}
class Adst extends C {
}

Asrc a = new Asrc();
C c = new C();
a.f = new Adst();
Precision of points-to edges

<table>
<thead>
<tr>
<th>Field Declaration (C f)</th>
<th>( \langle A_{src}, D_{src}, B_{src} \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool fChild</td>
<td>( \langle \text{SelectionTool}, \text{CTRL}, \text{BreakThrough} \rangle )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AllPossibleSubClasses(Tool)</th>
<th>OOGPossibleSubTypes(Tool)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NullTool, SelectAreaTracker, BoardActionTool, SelectionTool, DragTracker, SelectionTool, DragTracker</td>
<td>NullTool, SelectAreaTracker, DragTracker</td>
</tr>
</tbody>
</table>

\[
\text{precision}(E_{fchild}) = 1 - \frac{3}{7} = 0.57
\]
Recent work: for some tasks, developers using \textit{type}+\textit{hierarchy}+\textit{group} spent less time and explored fewer code elements than developers using only \textit{type} \\
[\text{Ammar and Abi-Antoun, WCRE’12}]

This work: use metrics to better understand how different code and runtime structure are

Future work: run metrics across 8 systems (100 KLOC)
  - Find systems for which differences are higher
  - Find where in a system differences are higher
Future Work: compute metrics across systems

Points-to edge precision (Maximum)

Systems:
- MD
- CDB
- AFS
- DL
- PX
- JHD
- HC
- APD