Empirical Evaluation of Diagrams of the Run-Time Structure for Coding Tasks

Nariman Ammar             Marwan Abi-Antoun
Department of Computer Science

Wayne State University

October 15-18 2012
Kingston | Ontario | Canada

WCRE 2012
19th Working Conference on Reverse Engineering
Diagrams (can?) help developers with code modifications during maintenance

- Program comprehension is hard:
  - Software maintenance costs 50%-90%
  - Of that, 50% spent in program comprehension
  [Bennett et al., Advances in Computers’02]
- Widespread belief that diagrams can help developers with program comprehension:
  - Diagrams of static/code structure
  - Diagrams of dynamic/runtime structure
  - Other diagrams
- In object-oriented design, runtime structure very different from code structure
Structure

Diagram of Code Structure (DCS)
E.g., class diagram

A typical object structure might look like this:

Diagram of Run-time Structure (DRS)
E.g., object diagram
Diagram of Code Structure (DCS) vs. Diagram of Run-time Structure (DRS)

**Diagram of Code Structure (DCS)**
- Static **code organization** (packages, classes, etc.)
- E.g., UML class diagram
- **Class**-based view
- Shows code relationships
  - Inheritance, Imports, Uses Calls/References/Instatiates
- Shows one box per class

**Diagram of Run-time Structure (DRS)**
- Dynamic runtime structure as networks of communicating **objects**
- E.g., UML object diagram
- **Object**-based view
- Shows run-time relationships
  - Points-to
- Shows multiple instances of the same class

- **Motivation**
- Theory
- Experiment
- Results
- Discussion
- Related Work
- Conclusion
Diagram of Code Structure (DCS) vs. Diagram of Run-time Structure (DRS)

Diagram of Code Structure
- Static code organization (packages, classes, etc.)
- E.g., UML class diagram
- Class-based view
- Shows code relationships
  - Inheritance, Imports, Uses Calls/References/Instantiates
- Shows one box per class

Diagram of Run-time Structure
- Dynamic runtime structure as networks of communicating objects
- E.g., UML object diagram
- Object-based view
- Shows run-time relationships
  - Points-to
- Shows multiple instances of the same class

• Motivation • Theory • Experiment • Results • Discussion • Related Work • Conclusion
Naïve object graph extraction

Flat object graph [Jackson and Waingold, TSE’01]

MiniDraw: 1,500 LOC
31 classes and 17 interfaces

- Motivation - Theory - Experiment - Results - Discussion - Related Work - Conclusion
A theory of object-oriented program comprehension
In object-oriented code, objects matter (in addition to types) [Gamma et al., 1994]

**Composite**

```
   aPicture
    /\    \   
   /  \  /  \  
aText aLine aRectangle
```

**Decorator**

```
   aDecorator
      \___/  
     |    |   
    |    |   
   aComponent
```

**Mediator**

```
   aFontDialogDirector
       \___/  
      |    |   
     |    |   
   aButton
```

**Proxy**

```
aClient
  \___/  
 |    |   
|    |   
aProxy
   \___/  
     |    |   
aRealSubject
```

**Strategy**

```
aComponent
   \___/  
  |    |   
|    |   
aStrategy
```

Source: Design Patterns, 1994.

- Motivation
- Theory
- Experiment
- Results
- Discussion
- Related Work
- Conclusion
Specific instances do NOT matter

- How to make such an object diagram scale?
  - Merge related objects
  - Collapse objects underneath other objects

Raw object graph

OOG

Motivation • Theory • Experiment • Results • Discussion • Related Work • Conclusion
Previously proposed solution: OOG
[Abi-Antoun & Aldrich, OOPSLA’09]

- Ownership Object Graph (OOG)
  - **Statically extracted** DRS
  - "Sound": reflect all objects and relations
  - Developers can base decisions on sound diagram

- OOG describes role of an object, not just
  - by type, but
  - by named **groups (domains)** and
  - by position in **object hierarchy**
Abstraction by *type+group+hierarchy*

[Abi-Antoun & Aldrich, OOPSLA’09]

- Compared to flat object graph, OOG:
  - **Group** related objects
  - Impose **hierarchy on objects**
  - (Do not delete objects, just push up/down)

- **Group** = "domain"
  - **Definition**: a domain is a named, conceptual group of objects.
  - Design intent expressed using annotations that are added to the code
  - Some annotations can be automatically inferred
Abstraction by type+group+hierarchy
[Abi-Antoun & Aldrich, OOPSLA’09]

• Use abstraction by object hierarchy:
  • Architecturally-relevant objects higher up
  • Low-level objects below

• Hierarchy promotes both:
  • High-level understanding; and
  • Detailed understanding

• Two kinds of object hierarchy:
  • Logical containment: Is-Part-Of
  • Strict encapsulation: Is- Owned-By
Abstraction by *type+group+hierarchy*

[Abi-Antoun & Aldrich, OOPSLA’09]

- **Motivation**
- **Theory**
- **Experiment**
- **Results**
- **Discussion**
- **Related Work**
- **Conclusion**
Is-Part-Of vs. Is-Owned-By

- **Strict encapsulation**
  **Is-Owned-By**
  (private domain)

- **Logical containment**
  **Is-Part-Of**
  (public domain)
Information content of DCS vs. DRS

DCS (hierarchy of classes)
+- package/
  +- package/
    | +- Hashtable
    | +- class/
    |   +- innerclass/
  +- package/
    | +- class/
    |   +- innerclass/

DRS (hierarchy of objects)
+-root/
  +- TLD1/
    | +- object1: B
    |   +- MAPS
    |     +- hash1: Hashtable
    |     +- hash2: Hashtable
  +- TLD2/
    | +- object2: B
    |   +- OWNED
    |     +- hash3: Hashtable

class B {
  public void m() {
    @Domain("OWNED")
    Hashtable hash2 = new Hashtable();
  }
}

Trace to code

- Motivation
- Theory
- Experiment
- Results
- Discussion
- Related Work
- Conclusion
Describe role of object more precisely than type alone

**DRS (hierarchy of objects)**

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

- Motivation
- **Theory**
- Experiment
- Results
- Discussion
- Related Work
- Conclusion
Describe role of object by \textit{type+group+hierarchy}

- Describe role of an object, not just by \textit{type}, but by named \textit{groups (domains)} or by position in \textit{object hierarchy}

- \langle A, D, B \rangle:
  - object of type A
  - in domain D in
  - parent object of type B

\textbf{DRS (hierarchy of objects)}

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

\begin{itemize}
  \item Motivation
  \item \textbf{Theory}
  \item Experiment
  \item Results
  \item Discussion
  \item Related Work
  \item Conclusion
\end{itemize}
Key program comprehension questions that involve the role of an object

- `<A,D,B>` information provides **key facts**:
  - **Is-In-Tier**: `<A,D_{TLD},B>`
  - **Is-Owned-By**: `<A,D_{private},B>`
  - **Is-Part-Of**: `<A,D_{public},B>`

- **Facts can answer key questions**:
  - How-To-Get-A
  - How-To-Get-A-In-B
  - Which-Tier-Has-A
  - Which-A-In-B
Some program comprehension questions not easily answered using DCS tools

<table>
<thead>
<tr>
<th>Attempt</th>
<th>Question</th>
<th>Information source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>How-To-Get-A</strong>: Search for an instance of container type A e.g., <em>ArrayList</em></td>
<td>Eclipse search (e.g., &quot;list&quot; returns 74 results)</td>
</tr>
<tr>
<td>2</td>
<td><strong>How-To-Get-A</strong>: Search for where an instance of a contained element is created e.g., <em>ArrayList&lt;Piece&gt;</em></td>
<td>Eclipse search (e.g., &quot;piece&quot; returns 37 results)</td>
</tr>
<tr>
<td>3</td>
<td><strong>How-To-Get-A-In-B</strong>: Search for an instance of type A in an enclosing type B e.g., piece in <em>BoardDrawing</em></td>
<td>Eclipse search/ association on class diagram</td>
</tr>
<tr>
<td>4</td>
<td><strong>Which-A-In-B</strong>: Search for an instance of type A in domain D in an instance of type B e.g., <em>ArrayList,MAPS,BoardDrawing</em></td>
<td>&lt;A,D,B&gt; fact on OOG</td>
</tr>
</tbody>
</table>
Key question: Which-A-in-B?
B has many objects of type A, which one do I need?

* MiniDraw: many HashMap objects

* A DCS shows only one.

* OOG shows that boardDrawing object has two objects:
  Is-Owned-By:
  <HashMap,owned,BoardDrawing>
  Is-Part-Of:
  <HashMap,MAPS,BoardDrawing>
An experiment to evaluate the theory of program comprehension
Tool Used by Participants

- Motivation
- Theory
- **Experiment**
- Results
- Discussion
- Related Work
- Conclusion
Contributions

• Theory in comprehension
  • Type+group+hierarchy \(<A,D,B>\) describes object's role better than type alone
  • OOG conveys \(<A,D,B>\) facts to answer comprehension questions

• Controlled experiment
  • Do code modifications involve questions about run-time structure?
  • Does OOG help in addition to DCS tools?
Research Hypothesis

For code modification tasks that require knowledge about the run-time structure, developers who use DRS tools require less comprehension effort, explore less irrelevant code, and spend less time than developers who use only DCS tools.
Participants

- 10 participants
  - 4 professionals
  - 3 Ph.D., 4th year
  - 2 M.S.
  - 1 senior undergraduate
- Total programming experience:
  - Median = 8.5 years
- Java programming experience:
  - Median = 4 years
Study Design

- Between-subjects
  - Control group, Experimental group
  - 5 participants per group
- Dependent variable
  - Having access to OOG
- Independent variables
  - Time spent on a task
  - Code explored in a task
Tools and Instrumentation

- Both groups:
  - Eclipse IDE
  - Instruction sheet
  - 10 class diagrams
  - Record screen and audio (Camtasia)

- Experimental group:
  - OOG printout
  - Interactive OOG viewer in Eclipse
Task Design

- MiniDraw drawing framework
  - BreakThrough board game
- Three feature implementation tasks
- Missing game logic
  - T1: Validate moves
  - T2: Capture pieces
  - T3: Undo a move
## Procedure

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
<th>Time (3 hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Brief intro to MiniDraw</td>
<td>2 minutes</td>
</tr>
<tr>
<td></td>
<td>Tutorial on Eclipse navigation</td>
<td>3 minutes</td>
</tr>
<tr>
<td></td>
<td>Tutorial on OOG notation and tool navigation features</td>
<td>20 minutes (Control group during last 20 minutes)</td>
</tr>
<tr>
<td>2</td>
<td>Perform tasks and answer questionnaires between tasks</td>
<td>2.5 hours</td>
</tr>
<tr>
<td>3</td>
<td>Exit interview questions</td>
<td>5 minutes</td>
</tr>
</tbody>
</table>

- Motivation
- Theory
- **Experiment**
- Results
- Discussion
- Related Work
- Conclusion
Data Analysis

• Transcribed recordings offline
  • Time in video
  • Code explored

• Quantitative analysis
  • Non-parametric hypothesis tests (Wilcoxon)
  • Non-standardized effect size (Cliff’s delta, 95% CI)

• Qualitative analysis
  • Hierarchical task decomposition
    [Crystal et al., AMCIS’04]
    • Task →* Activities →* Questions →* Strategies/Facts
  • Think-aloud in the transcripts
    • Quotes
Developers who used DRS tools always outperformed developers who used only DCS tools

Code explored
Less irrelevant code
(by 10%--60%)

Effect sizes
T1  0.008, large
T2  0.264, small
T3  0.068, medium

Time spent
Less time (by 22%--60%)

Effect sizes
T1  0.147, medium
T2  0.232, medium
T3  0.048, medium
Observation: Control group struggled more with questions about the run-time structure

- **How-To-Get-A** (Experimental:48, Control:135)
  - “How can I get the data structure representing the game board?”
- **How-To-Get-A-In-B** (Experimental:19, Control:75)
  - “I want to get the figureMap. Why isn't it in Game?”
- **Which-Tier-Has-A** (Experimental:10, Control:17)
  - “What I'm trying to do is find the UI part of the code were I can add it [menu bar]”
- **Which-A-In-B** (Experimental:6, Control:6)
  - “Any of these are really a possibility of where it might have all the positions of all the pieces. I guess I should be looking for some sort of a data structure”
Observation: Control group used more time consuming strategies to answer questions

How-To-Get-A

Directly: Is-In-Tier + Is-Owned-By/Is-Part-Of

Directly: Has-A + Is-A

Fields in classes
Inheritance trees of all possible subtypes of A

Indirectly: Has-Label

(a) Types, fields, methods, comments.

(b) Object hierarchy

Control group:
Is-A (type hierarchy) 35
Has-Label (file search) 25
Has-A (java search) 72
Is-in-layer (packages) 18
Is in tier (java doc) 15

Experimental group:
Is in tier 26
Points-To 41
Is-Owned-By/Is-Part-Of 18

● Motivation ● Theory ● Experiment ● Results ● Discussion ● Related Work ● Conclusion
Threats to Validity

• Construct
  • Small sample
  • Task design
    • Not many Which-A-In-B questions
    • Learning effect from T1 to T2

• External
  • One subject system
  • Not real maintenance tasks
  • Not enough professional or Java developers
Threats to Validity

- Internal
  - Longer tutorial on OOG using MiniDraw
  - Heterogeneity of experience (ANCOVA)

<table>
<thead>
<tr>
<th>Task</th>
<th>covariate</th>
<th>time spent</th>
<th>code explored</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Java experience</td>
<td>P=0.48</td>
<td>P=0.52</td>
</tr>
<tr>
<td></td>
<td>Industry experience</td>
<td>P=0.45</td>
<td>P=0.78</td>
</tr>
<tr>
<td></td>
<td>Programming experience</td>
<td>P=0.46</td>
<td>P=0.83</td>
</tr>
<tr>
<td></td>
<td>Eclipse experience</td>
<td>P=0.47</td>
<td>P=0.71</td>
</tr>
<tr>
<td>T2</td>
<td>Java experience</td>
<td>P=0.39</td>
<td>P=0.92</td>
</tr>
<tr>
<td></td>
<td>Industry experience</td>
<td>P=0.17</td>
<td>P=0.20</td>
</tr>
<tr>
<td></td>
<td>Programming experience</td>
<td>P=0.14</td>
<td>P=0.28</td>
</tr>
<tr>
<td></td>
<td>Eclipse experience</td>
<td>P=0.64</td>
<td>P=0.59</td>
</tr>
<tr>
<td>T3</td>
<td>Java experience</td>
<td>P=0.80</td>
<td>P=0.15</td>
</tr>
<tr>
<td></td>
<td>Industry experience</td>
<td>P=0.50</td>
<td>P=0.63</td>
</tr>
<tr>
<td></td>
<td>Programming experience</td>
<td>P=0.79</td>
<td>P=0.01</td>
</tr>
<tr>
<td></td>
<td>Eclipse experience</td>
<td>P=0.08</td>
<td>&lt;0.05</td>
</tr>
</tbody>
</table>

- Motivation ● Theory ● Experiment ● Results ● Discussion ● Related Work ● Conclusion
Related Work

• Theories of comprehension
  • Bottom-up / Top-down [Storey, IWPC’05]
  • Multiple levels of abstraction [Pacione et al., WCRE’04]
  • Developers questions about code [Silitto et al., TSE’08, LaToza and Myers, PLATEAU’10]

• Evaluation of tools and diagrams for comprehension
  • Controlled experiments – using code modifications
    • Dynamically extracted information/diagrams [Quante et al., ICPC’08, Rothlisberger et al., TSE’11]
  • Controlled experiments – using questionnaires/interviews
    • Manually created diagrams: sequence diagrams, object diagrams [Scaniello et al., IET Seminar digests’11]
  • Case studies
    • Hierarchical Instance Models [Torchiano et al., CIT’99]
    • Dynamic vs. static object diagrams [Tonella and Potrich, ICSM’02]
    • Statically extracted (automatically) [Jackson and Waingold, TSE’01, O’Callahan’01, Spiegel’02]
    • Statically extracted (annotations) [Lam and Rinard, ECOOP’03]
Future Work

• Promising results, but more to do…
  • Add richer information to OOG
    • Evaluated OOG with points-to edges
    • More edges possible, e.g., dataflow edges: [Vanciu and Abi-Antoun, WCRE'12]
  • Reduce effort of OOG extraction
  • Address tool usability issues
  • Replicate results with larger sample
Conclusion

• Theory: *type+group+hierarchy*
  • describe role of an object more precisely than *type*
  • answer *key comprehension questions* related to an object's role and the run-time structure

• Controlled experiment:
  • First to evaluate statically extracted, global, hierarchical object graphs for code modifications
  • OOG helped answer questions about run-time structure more easily than DCS tools
  • On average, OOGs reduced effort:
    • Time spent by 22% -- 60%
    • Irrelevant code explored by 10% -- 60%