CAR-Miner: Mining Exception-Handling Rules as Sequence Association Rules

Suresh Thummalapenta and Tao Xie
Department of Computer Science
North Carolina State University
Raleigh, USA

This work is supported in part by NSF grant CCF-0725190 and ARO grant W911NF-08-1-0443.
Programmers commonly reuse APIs of existing frameworks or libraries

- Advantages: High productivity of development
- Challenges: Complexity and lack of documentation
- Consequences:
  - Programmers spend more efforts in understanding APIs
  - Defects in API client code
Background

Traditional approaches

Code repositories

Eclipse, Linux, ...

Often lack **sufficient relevant** data points (Eg. API call sites)

Our new approach

Open source code on the web

Code search engine e.g.,

**Code repositories**

1 2 3 ... N

**searching**

**mining**

patterns

patterns
Agenda

- Problem
- Example
- CAR-Miner Approach
- Evaluation
- Conclusion
APIs throw exceptions during runtime errors

Example: Session API of Hibernate framework throws HibernateException

APIs expect client applications to implement recovery actions after exceptions occur

Example: Session API of Hibernate expect client application to rollback open uncommitted transactions after HibernateException occurs

Failure to handle exceptions results in

Fatal issues: Database lock won’t be released if the transaction is not rolled back

Performance degradation due to resource leaks: 17% increase in the performance is found in a 34KLOC program after properly handling exceptions [Weimer and Necula, OOPSLA 04]
Use specification that describes exception-handling behavior and detect defects

**Problem:** Often specifications are not documented

**Solution:** Mine specifications from existing code bases using APIs

**Challenges:**
- **Limited data points:** Existing approaches mine specifications from a few code bases: lack of sufficient relevant data points may miss specifications
- **Limited expressiveness:** Simple specifications are not sufficient to characterize common exception-handling behaviors: why?
Example

Scenario 1

1.1: ...
1.2: OracleDataSource ods = null; Session session = null;
       Connection conn = null; Statement statement = null;
1.3: logger.debug("Starting update");
1.4: try {
1.5:     ods = new OracleDataSource();
1.6:     ods.setURL("jdbc:oracle:thin:scott/tiger@192.168.1.2:1521:catfish");
1.7:     conn = ods.getConnection();
1.8:     statement = conn.createStatement();
1.9:     statement.executeUpdate("DELETE FROM table1");
1.10:    connection.commit();
1.11:    catch (SQLException se) {
1.12:       logger.error("Exception occurred");
1.13:     }
1.14:    finally {
1.15:        if(statement != null) { statement.close(); }
1.16:        if(conn != null) { conn.close(); }
1.17:        if(ods != null) { ods.close(); }
1.18:    }

- Defect: No rollback done when SQLException occurs
- Requires specification such as “Connection should be rolled back when a connection is created and SQLException occurs”
- Q: Should every connection instance has to be rolled back when SQLException occurs?
<table>
<thead>
<tr>
<th>Scenario 1</th>
<th>Scenario 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1: ...</td>
<td>2.1: Connection conn = null;</td>
</tr>
<tr>
<td>1.2: OracleDataSource ods = null; Session session = null;</td>
<td>2.2: Statement stmt = null;</td>
</tr>
<tr>
<td>Connection conn = null; Statement statement = null;</td>
<td>2.3: BufferedWriter bw = null; FileWriter fw = null;</td>
</tr>
<tr>
<td>1.3: logger.debug(&quot;Starting update&quot;);</td>
<td>2.3: try {</td>
</tr>
<tr>
<td>1.4: try {</td>
<td>2.4:    fw = new FileWriter(&quot;output.txt&quot;);</td>
</tr>
<tr>
<td>1.5: ods = new OracleDataSource();</td>
<td>2.5:    bw = BufferedWriter(fw);</td>
</tr>
<tr>
<td>1.7: conn = ods.getConnection();</td>
<td>2.7:    Statement stmt = conn.createStatement();</td>
</tr>
<tr>
<td>1.8: statement = conn.createStatement();</td>
<td>2.8:    ResultSet res = stmt.executeQuery(&quot;SELECT Path FROM Files&quot;);</td>
</tr>
<tr>
<td>1.9: statement.executeUpdate(&quot;DELETE FROM table1&quot;);</td>
<td>2.9:    while (res.next()) {</td>
</tr>
<tr>
<td>1.10: connection.commit(); }</td>
<td>2.10:        bw.write(res.getString(1));</td>
</tr>
<tr>
<td>1.11: catch (SQLException se) {</td>
<td>2.11: }</td>
</tr>
<tr>
<td>1.12: if (conn != null) { conn.rollback(); }</td>
<td>2.12:    res.close();</td>
</tr>
<tr>
<td>1.13:    logger.error(&quot;Exception occurred&quot;); }</td>
<td>2.13: } catch (IOException ex) { logger.error(&quot;IOException occurred&quot;);</td>
</tr>
<tr>
<td>1.14: finally {</td>
<td>2.14: } finally {</td>
</tr>
<tr>
<td>1.15:    if (statement != null) { statement.close(); }</td>
<td>2.15:    if (stmt != null) stmt.close();</td>
</tr>
<tr>
<td>1.16:    if (conn != null) { conn.close(); }</td>
<td>2.16:    if (conn != null) conn.close();</td>
</tr>
<tr>
<td>1.17:    if (ods != null) { ods.close(); } }</td>
<td>2.17:    if (bw != null) bw.close();</td>
</tr>
<tr>
<td>1.18: }</td>
<td>2.18: }</td>
</tr>
</tbody>
</table>

**Specification:** “Connection creation => Connection rollback”

- Satisfied by Scenario 1 but not by Scenario 2
- But Scenario 2 has no defect
Simple association rules of the form “FCa => FCe” are not expressive

Requires more general association rules (sequence association rules) such as

\[(FCc_1 FCc_2) \land FCa \Rightarrow FCe_1, \text{ where}\]

FCc1 -> Connection conn = OracleDataSource.getConnection()
FCc2 -> Statement stmt = Connection.createStatement()
FCa -> stmt.executeUpdate()
FCe1 -> conn.rollback()
Simple association rules of the form “FCa => FCe” are not expressive

Requires more general association rules (sequence association rules) such as

\[(FCc_1 \, FCc_2) \land FCa \Rightarrow FCe_1, \text{ where}\]

\[FCc_1 \Rightarrow \text{Connection conn = OracleDataSource.getConnection()}\]

\[FCc_2 \Rightarrow \text{Statement stmt = Connection.createStatement()}\]

\[FCa \Rightarrow \text{stmt.executeUpdate()} //\text{Triggering Action}\]

\[FCe_1 \Rightarrow \text{conn.rollback()}\]
Simple association rules of the form “FCa => FCe” are not expressive

Requires more general association rules (sequence association rules) such as

\[(FCc_1 FCc_2) \land FCa \Rightarrow FCe_1, \text{ where}\]

FCc1 -> Connection conn = OracleDataSource.getConnection()

FCc2 -> Statement stmt = Connection.createStatement()

FCa -> stmt.executeUpdate()

FCe1 -> conn.rollback() //Recovery Action
Simple association rules of the form “FCa => FCe” are not expressive

Requires more general association rules (sequence association rules) such as

\[(FCc_1 \land FCc_2) \land FCa \Rightarrow FCe_1, \text{ where} \]

\[FCc_1 \Rightarrow \text{Connection conn} = \text{OracleDataSource.getConnection()}\]
\[FCc_2 \Rightarrow \text{Statement stmt} = \text{conn.createStatement()} //\text{Context}\]
\[FCa \Rightarrow \text{stmt.executeUpdate()}\]
\[FCe_1 \Rightarrow \text{conn.rollback()}\]
CAR-Miner Approach

**Input Application**
- Check whether there are any exception-related defects

**Classes and Functions**
- Extract classes and functions reused

**Open Source Projects on web**
- Issue queries and collect relevant code examples. Eg: “lang:java java.sql.Statement executeUpdate”

**Exception-Flow Graphs**
- Construct exception-flow graphs

**Static Traces**
- Collect static traces
- Mine static traces

**Violations**
- Detect violations

**Sequence Association Rules**
CAR-Miner Approach

Open Source Projects on web

Classes and Functions

Input Application

Exception-Flow Graphs

Static Traces

Sequence Association Rules

Violations
Exception-Flow-Graph Construction

2.1: Connection conn = null;
2.2: Statement stmt = null;
2.3: BufferedWriter bw = null; FileWriter fw = null;
2.3: try {
2.4:     fw = new FileWriter("output.txt");
2.5:     bw = BufferedWriter(fw);
2.6:     conn = DriverManager.getConnection("jdbc:pl:db", "ps", "ps");
2.7:     Statement stmt = conn.createStatement();
2.8:     ResultSet res = stmt.executeQuery("SELECT Path FROM Files");
2.9:     while (res.next()) {
2.10:         bw.write(res.getString(1));
2.11:     }
2.12:     res.close();
2.13: } catch (IOException ex) { logger.error("IOException occurred");
2.14: } finally {
2.15:     if (stmt != null) stmt.close();
2.16:     if (conn != null) conn.close();
2.17:     if (bw != null) bw.close();
2.18: }

- Based on algorithm by Sinha and Harrold (TSE 00)
- Solid: normal execution path, Dotted: exceptional execution path
Exception-Flow-Graph Construction

2.1: Connection conn = null;
2.2: Statement stmt = null;
2.3: BufferedWriter bw = null; FileWriter fw = null;
2.3: try {
2.4:    fw = new FileWriter("output.txt");
2.5:    bw = BufferedWriter(fw);
2.6:    conn = DriverManager.getConnection("jdbc:pl:db", "ps", "ps");
2.7:    Statement stmt = conn.createStatement();
2.8:    ResultSet res = stmt.executeQuery("SELECT Path FROM Files");
2.9:    while (res.next()) {
2.10:       bw.write(res.getString(1));
2.11:    }
2.12:    res.close();
2.13: } catch (IOException ex) { logger.error("IOException occurred");
2.14: } finally {
2.15:    if (stmt != null) stmt.close();
2.16:    if (conn != null) conn.close();
2.17:    if (bw != null) bw.close();
2.18: }

- Prevent infeasible edges using a sound-static analysis, called Jex [Robillard and Murphy (FSE 99)]
- Jex provides all potential exceptions thrown by a function call
CAR-Miner Approach

Input Application → Classes and Methods → Open Source Projects on web

Exception-Flow Graphs → Static Traces

Violations → Sequence Association Rules
Collect static traces with the actions taken when exceptions occur

A static trace for Node 7: “4 -> 5 -> 6 -> 7 -> 15 -> 16 -> 17”
Static Trace Generation

- Includes 3 sections:
  - Normal function-call sequence (4 -> 5 -> 6)
  - Function call (7)
  - Exception function-call sequence (15 -> 16 -> 17)

A static trace for Node 7: “4 -> 5 -> 6 -> 7 -> 15 -> 16 -> 17”
Trace Post-Processing

- Identify and remove unrelated function calls using data-dependency

- “4 -> 5 -> 6 -> 7 -> 15 -> 16 -> 17”

  4: FileWriter fw = new FileWriter("output.txt")
  5: BufferedWriter bw = new BufferedWriter(fw)
  ...
  7: Statement stmt = conn.createStatement()
  ...

- Filtered sequence “6 -> 7 -> 15 -> 16”
CAR-Miner Approach

Open Source Projects on web

Classes and Methods

Input Application

Exception-Flow Graphs

Static Traces

Violations

Sequence Association Rules
Handle traces of each function call (triggering function call) individually

Input: Two sequence databases with a one-to-one mapping
- normal function-call sequences (context)
- exception function-call sequences (recovery)

Objective: Generate sequence association rules of the form

$$(FCc_1 \ldots FCc_n) \land FCa \Rightarrow FCe_1 \ldots FCen$$

Context \hspace{1cm} Trigger \hspace{2cm} Recovery
Mining Problem Definition

- **Input**: Two sequence databases with a one-to-one mapping

<table>
<thead>
<tr>
<th>Context</th>
<th>Recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDB₁</td>
<td>SDB₂</td>
</tr>
<tr>
<td>3, 6, 9, 10</td>
<td>2, 3, 7, 8</td>
</tr>
<tr>
<td>3, 10, 13</td>
<td>2, 6, 8</td>
</tr>
<tr>
<td>9, 10, 1, 19</td>
<td>9, 16, 13</td>
</tr>
</tbody>
</table>

- **Objective**: To get association rules of the form
  \[ \text{FC}₁ \text{ FC}₂ \ldots \text{ FC}_m \rightarrow \text{FE}₁ \text{ FE}₂ \ldots \text{ FE}_n \]

  where \{FC₁, FC₂, ..., FCₘ\} ∈ SDB₁ and \{FE₁, FE₂, ..., FEₙ\} ∈ SDB₂

- **Existing association rule mining algorithms cannot be directly applied on multiple sequence databases**
Mining Problem Solution

- Annotate the sequences to get a single combined database
  
  \[
  SDB_{1,2} = \begin{align*}
  &3^1, 6^1, 9^1, 10^1, 2^2, 3^2, 7^2, 8^2 \\
  &3^1, 10^1, 13^1, 2^2, 6^2, 8^2 \\
  &9^1, 10^1, 1^1, 19^1, 9^2, 16^2, 13^2
  \end{align*}
  \]

- Apply frequent subsequence mining algorithm [Wang and Han, ICDE 04] to get frequent sequences
  
  \[
  SDB_{1,2} = \begin{align*}
  &3^1, 10^1, 2^2, 8^2
  \end{align*}
  \]

- Transform mined sequences into sequence association rules
  
  \[3, 10 \Rightarrow 2, 8\]

- Rank rules based on the support assigned by frequent subsequence mining algorithm
  
  \[\text{Context} \quad \text{Trigger} \quad \text{Recovery} \]
  
  \[(3, 10) \land \text{FCa} \Rightarrow (2, 8)\]
CAR-Miner Approach

Open Source Projects on web

Classes and Methods

Input Application

Exception-Flow Graphs

Static Traces

Sequence Association Rules

Violations
Analyse each call site of triggering function call in input application to detect potential violations.

Extract context function call sequence from the beginning of the function to the call site, say “CC1 CC2 ... CCn”.

If FCc1 ... FCcn is a sub-sequence of CC1 CC2 ... CCn

Report any missing function calls of { FCe1 ... FCe{n} } in any exception path as violations.
Research Questions:

1) Do the mined rules represent real rules?
2) Do the detected violations represent real defects?
- Does CAR-Miner perform better than WN-miner [Weimer and Necula, TACAS 05]?
1) Do the sequence association rules help detect new defects?
## Subjects

<table>
<thead>
<tr>
<th>Subject</th>
<th>Lines of code</th>
<th>Internal Info</th>
<th>External Info</th>
<th># Code Examples</th>
<th>Time (in sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axion 1.0M2</td>
<td>24k</td>
<td>219/2405</td>
<td>58/217</td>
<td>47783 (7M)</td>
<td>1381</td>
</tr>
<tr>
<td>Hsqldb 1.7.1</td>
<td>30k</td>
<td>98/1179</td>
<td>80/264</td>
<td>78826 (26M)</td>
<td>2547</td>
</tr>
<tr>
<td>Hibernate 2.0 b4</td>
<td>39k</td>
<td>452/4321</td>
<td>174/883</td>
<td>88153 (27M)</td>
<td>1125</td>
</tr>
<tr>
<td>SableCC 2.18.2</td>
<td>22k</td>
<td>183/1551</td>
<td>21/76</td>
<td>47594 (15M)</td>
<td>1220</td>
</tr>
<tr>
<td>Ptolemy 3.0.2</td>
<td>170k</td>
<td>1505/9617</td>
<td>477/2595</td>
<td>70977 (21M)</td>
<td>1126</td>
</tr>
</tbody>
</table>

- **Internal Info**: classes and methods belonging to the application
- **External Info**: classes and methods used by the application
- **Code examples**: amount of code collected through code search engine
RQ1: Real Rules

Do the mined rules represent real rules?

- Real rules: 55% (Total: 294)
- Usage patterns: 3%
- False positives: 43%
RQ1: Distribution of Real Rules for Axion

- Distribution of rules based on ranks assigned by CAR-Miner

- Number of false positives is quite low between 1 to 60 rules
RQ2: Detected Violations

- Do the detected violations represent real defects?

<table>
<thead>
<tr>
<th>Subject</th>
<th>#Total Violations</th>
<th>#Violations of first 10 rules</th>
<th>#Defects</th>
<th>#Hints</th>
<th>#FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axion 1.0M2</td>
<td>257</td>
<td>19</td>
<td>13</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>HsqlDB 1.7.1</td>
<td>394</td>
<td>62</td>
<td>51</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Hibernate 2.0 b4</td>
<td>136</td>
<td>22</td>
<td>12</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Sablecc 2.18.2</td>
<td>168</td>
<td>66</td>
<td>45</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>Ptolemy 3.0.2</td>
<td>665</td>
<td>95</td>
<td>39</td>
<td>1</td>
<td>55</td>
</tr>
</tbody>
</table>

- Total number of defects: 160
- New defects not found by WN-Miner approach: 87
### RQ2: Status of Detected Violations

<table>
<thead>
<tr>
<th>Package</th>
<th># Defects</th>
<th>New Version</th>
<th>#Fixed</th>
<th>#Deleted</th>
<th>#Open</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axion 1.0M2</td>
<td>13</td>
<td>1.0M3</td>
<td>4</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>HsqlDB 1.7.1</td>
<td>51</td>
<td>1.8.0.9</td>
<td>2</td>
<td>9</td>
<td>40</td>
</tr>
<tr>
<td>Hibernate 2.0 b4</td>
<td>12</td>
<td>3.2.6</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>Sablecc 2.18.2</td>
<td>45</td>
<td>4-alpha.3</td>
<td>0</td>
<td>43</td>
<td>2</td>
</tr>
<tr>
<td>Ptolemy 3.0.2</td>
<td>39</td>
<td>3.0.2</td>
<td>0</td>
<td>0</td>
<td>39</td>
</tr>
</tbody>
</table>

- HsqlDB developers responded on the first 10 reported defects
  - Accepted 7 defects
  - Rejected 3 defects
- Reason given by HsqlDB developers for rejected defects:
  
  "Although it can throw exceptions in general, it should not throw with HsqlDB, So it is fine"
RQ3: Comparison with WN-miner

- Does CAR-Miner performs better than WN-miner?

- Found 224 new rules and missed 32 rules
- CAR-Miner detected most of the rules mined by WN-miner
- Two major factors:
  - sequence association rules
  - Increase in the data scope
RQ4: New defects by sequence association rules

- Do the sequence association rules detect new defects?

<table>
<thead>
<tr>
<th></th>
<th># Rules</th>
<th># Violations</th>
<th># Defects</th>
<th># Hints</th>
<th># False Positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axion</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>HsqlDB</td>
<td>6</td>
<td>14</td>
<td>8</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Hibernate</td>
<td>4</td>
<td>10</td>
<td>8</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Sablecc</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ptolemy</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Detected 21 new real defects among all applications
Conclusion

- Problem-driven methodology for advancing mining software engineering data by identifying
  - new problems, patterns
  - mining algorithms, defects
- CAR-Miner mines sequence association rules of the form

\[(\text{FCc}1 \ldots \text{FCcn}) \land \text{FCa} \Rightarrow (\text{FCe}1 \ldots \text{Fcen})\]

  Context   Trigger   Recovery

- Future work: Exploit synergy between mining and testing
  - Test generation to dynamically confirm violations
  - Mine method-call sequences for test generation
Thank You