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.

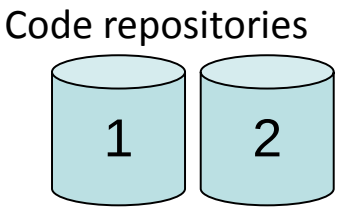
Motivation

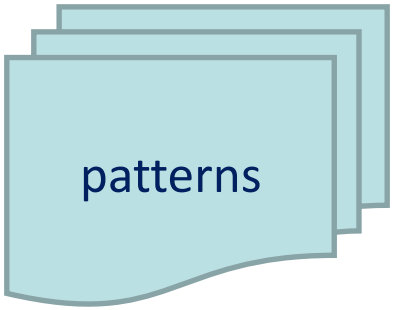
- Programmers commonly reuse APIs of existing frameworks or libraries

  - 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

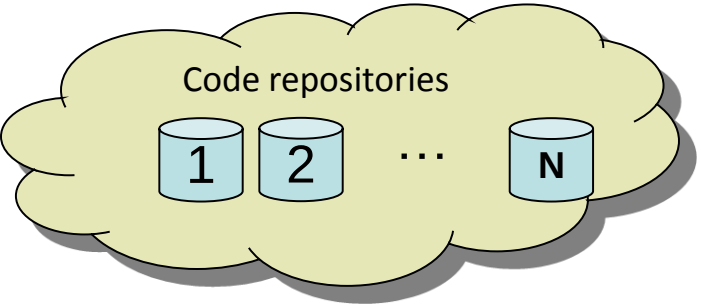


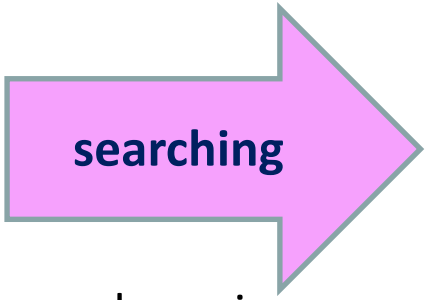


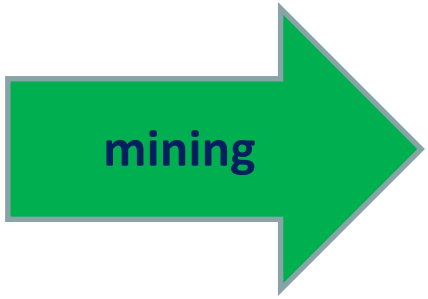
Eclipse, Linux, …

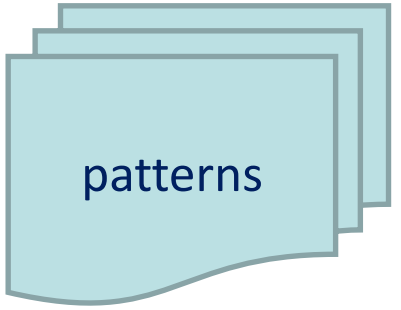
Often lack **sufficient** **relevant** data points (Eg. API call sites)

Our new approach









Open source code on the web

Code search engine e.g., Google /*Code Search*/ LABS

Agenda

- Problem
- Example
- CAR-Miner Approach
- Evaluation
- Conclusion

Exception Handling

- 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]

Problem Addressed by CAR-Miner

- 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)  {
1.13                 logger.error("Exception occurred");  }
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: }
```

Missing “conn.rollback()”

- 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*?

Example (contd)

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:                 if (conn != null)  { conn.rollback(); }
1.13:                 logger.error("Exception occurred");  }
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: }
```

Scenario 2

```
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: }
```

Specification: “Connection creation => Connection rollback”

- Satisfied by Scenario 1 but not by Scenario 2
- But Scenario 2 has no defect

Example (contd)

- Simple association rules of the form “FCa => FCe” are not expressive

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

(FCc1 FCc2) Λ FCa => FCe1, where

FCc1 -> Connection conn = OracleDataSource.getConnection()

FCc2 -> Statement stmt = Connection.createStatement()

FCa -> stmt.executeUpdate()

FCe1 -> conn.rollback()

Example (contd)

- Simple association rules of the form “FCa => FCe” are not expressive

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

(FCc1 FCc2) Λ FCa => FCe1, where

FCc1 -> Connection conn = OracleDataSource.getConnection()

FCc2 -> Statement stmt = Connection.createStatement()

FCa -> stmt.executeUpdate() //Triggering Action

FCe1 -> conn.rollback()

Example (contd)

- Simple association rules of the form “FCa => FCe” are not expressive

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

(FCc1 FCc2) Λ FCa => FCe1, where

FCc1 -> Connection conn = OracleDataSource.getConnection()

FCc2 -> Statement stmt = Connection.createStatement()

FCa -> stmt.executeUpdate()

FCe1 -> conn.rollback() //Recovery Action

Example (contd)

- Simple association rules of the form “FCa => FCe” are not expressive

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

(FCc1 FCc2) Λ FCa => FCe1, where

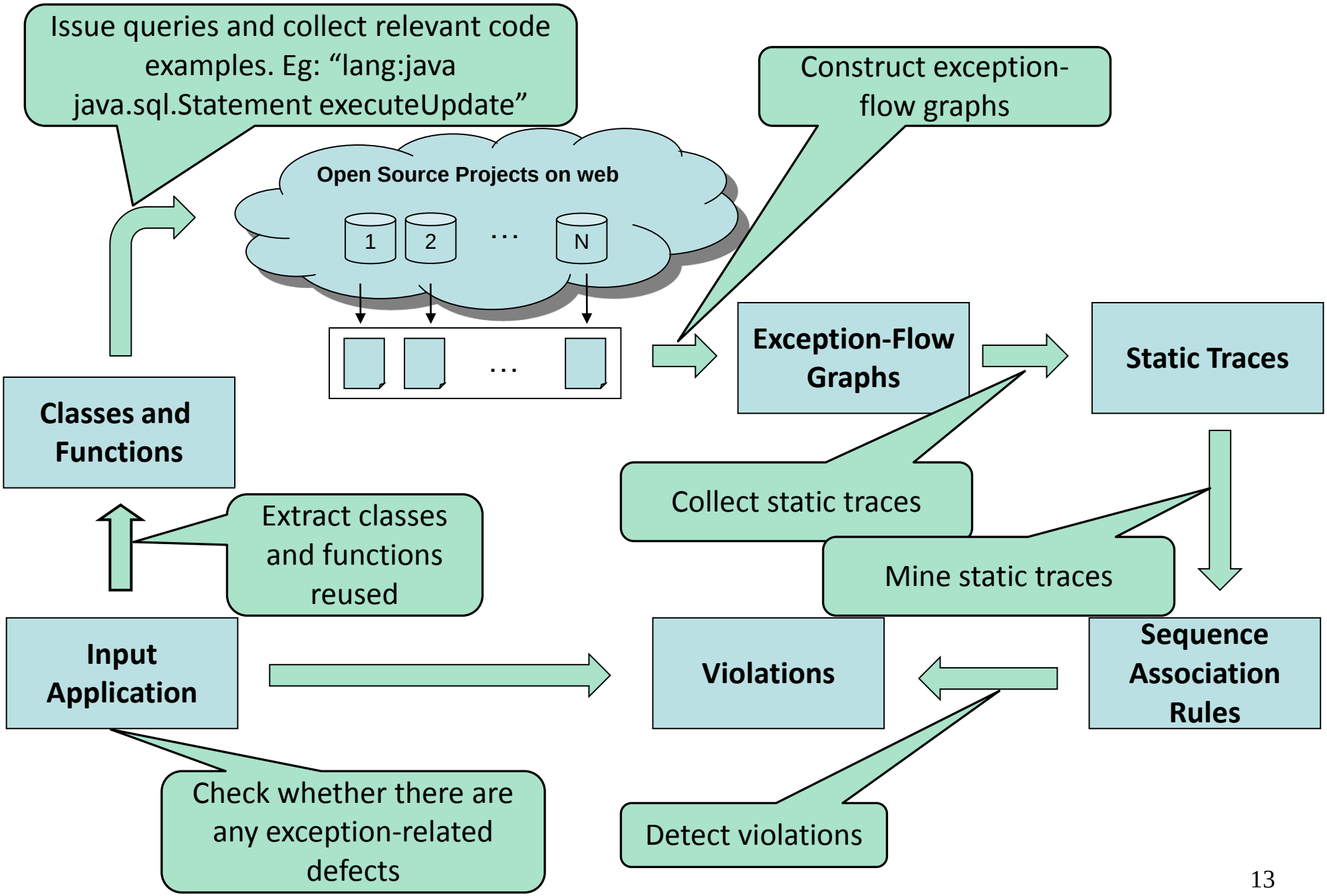
FCc1 -> Connection conn = OracleDataSource.getConnection()

FCc2 -> Statement stmt = conn.createStatement() //Context

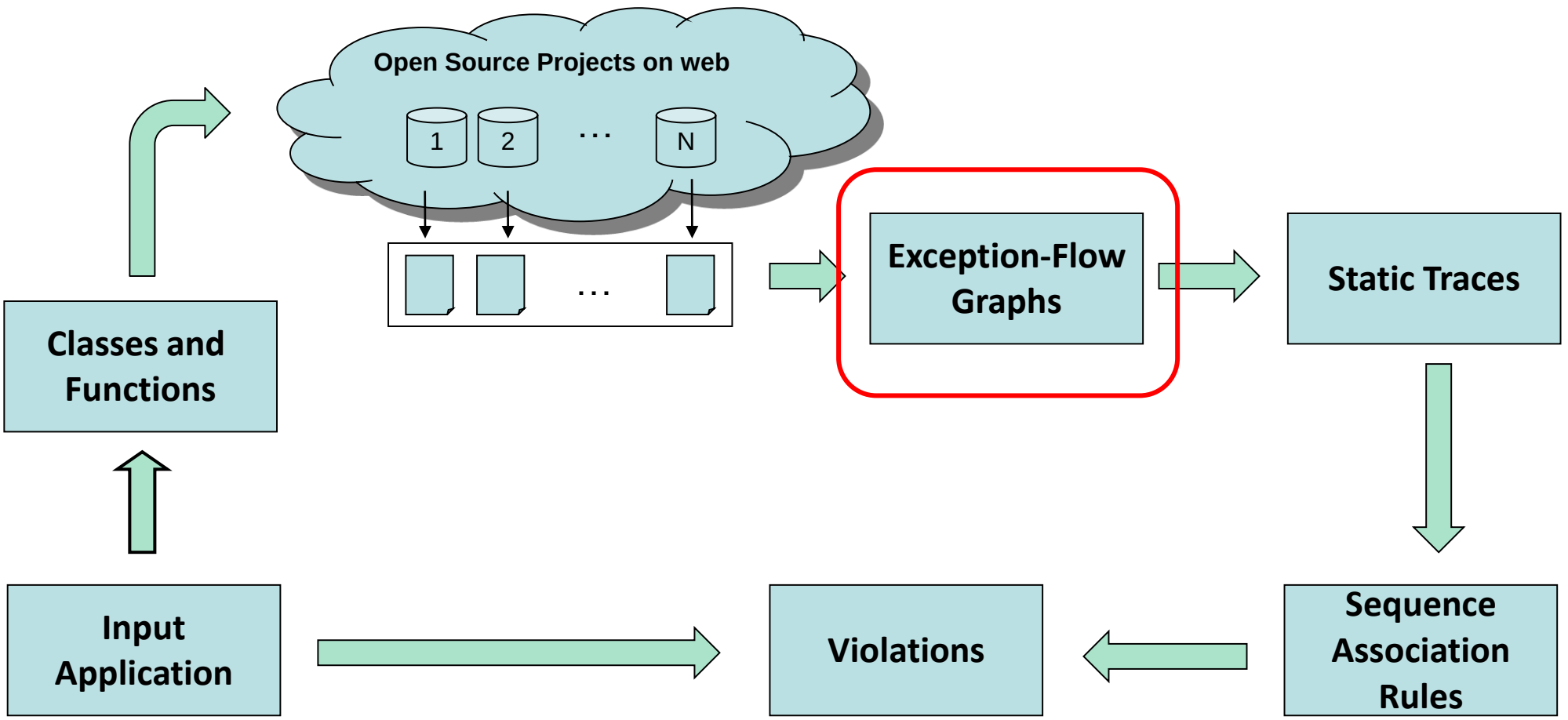
FCa -> stmt.executeUpdate()

FCe1 -> conn.rollback()

CAR-Miner Approach

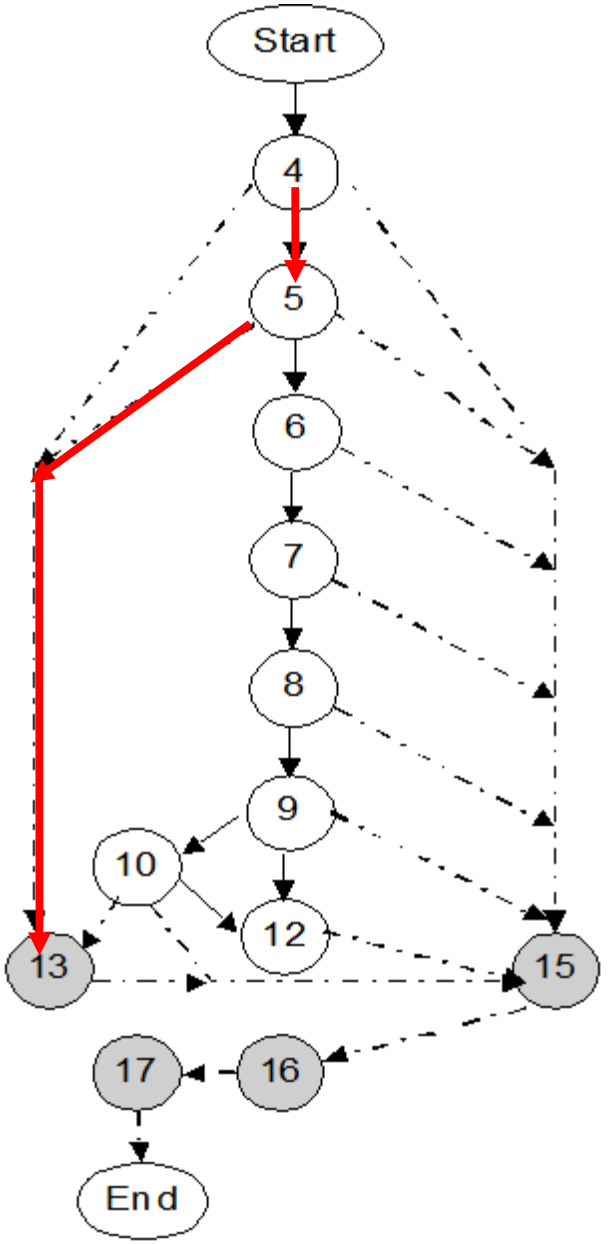


CAR-Miner Approach



Exception-Flow-Graph Construction

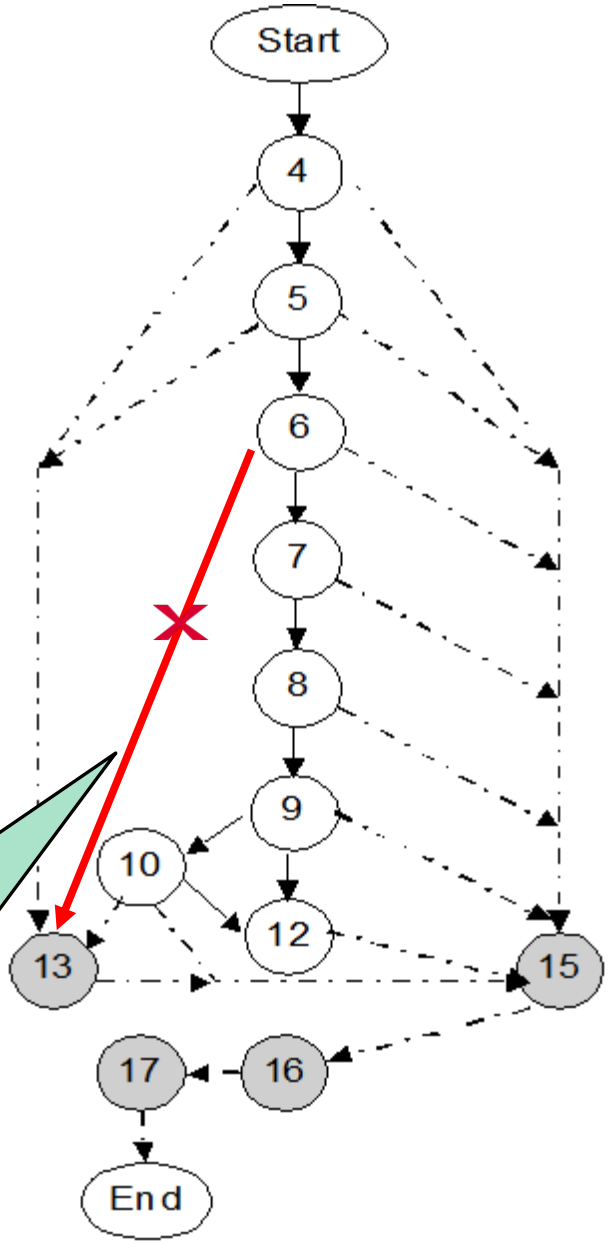
```
2.1: Connection conn = null;
2.2: Statement stmt = null;
2.3: BufferedWriter bw = null; FileWriter fw = null;
2.3: try {
2.4:     fw = new FileWriter("output.txt");
2.5:     bw = BufferedWriter(fw);
2.6:     conn = DriverManager.getConnection("jdbc:pl:db", "ps", "ps");
2.7:     Statement stmt = conn.createStatement();
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

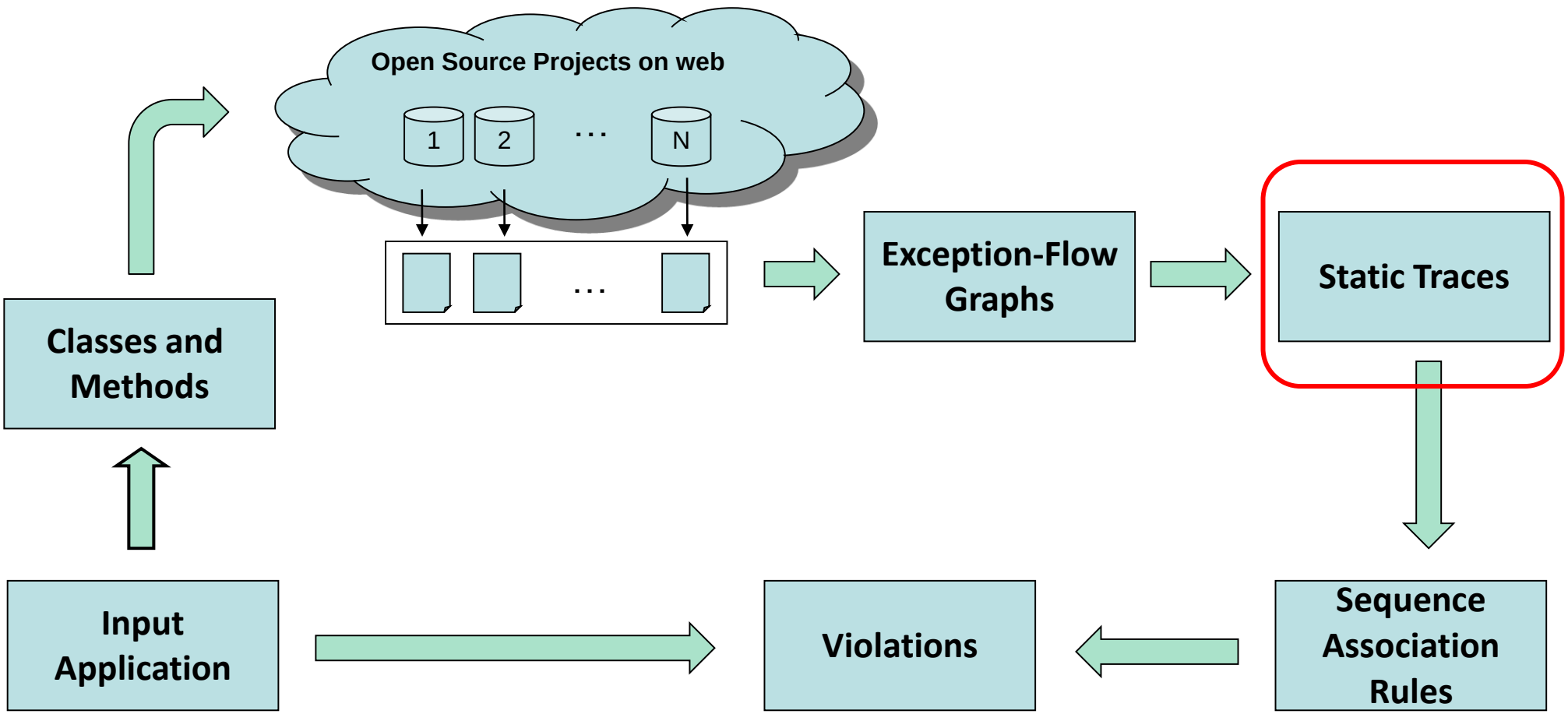
Exception-Flow-Graph Construction

```
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: }
```

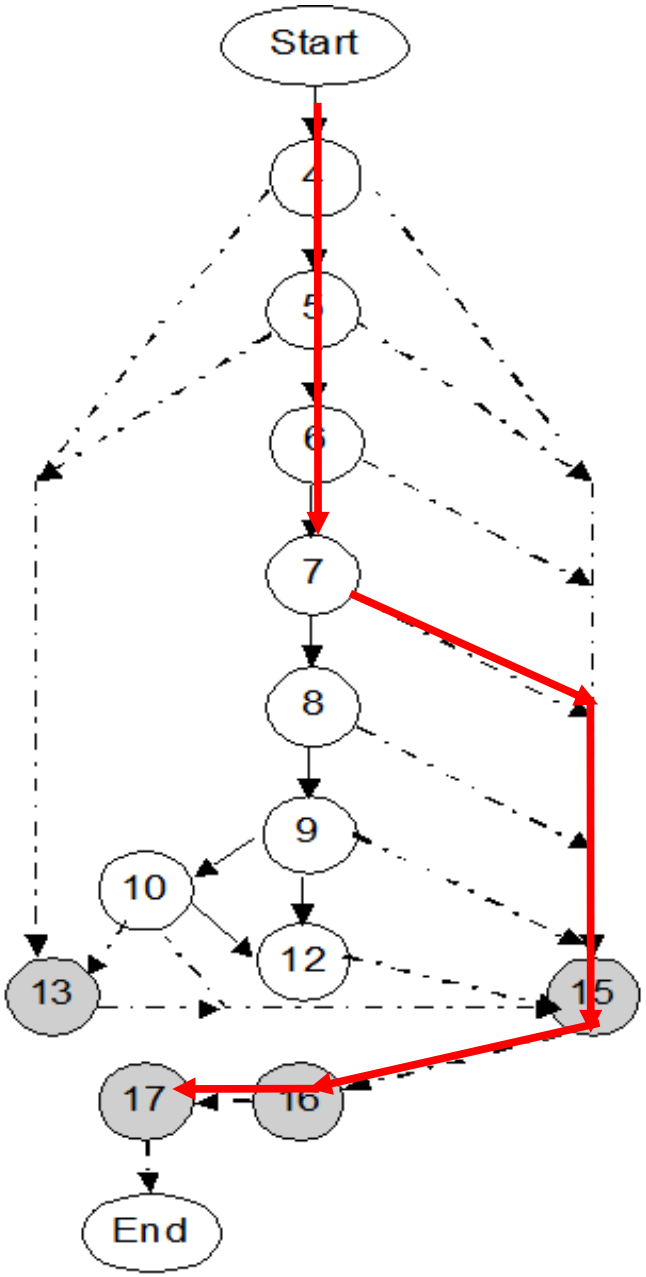


- Prevent infeasible edges using a sound-static analysis, called Jex [Robillard and Murphy (FSE 99)]
- Jex provides all potential exceptions thrown by a function call

CAR-Miner Approach



Static Trace Generation



- Collect static traces with the actions taken when exceptions occur
- A static trace for Node 7:
  "4 -> 5 -> 6 -> 7 -> 15 -> 16 -> 17"

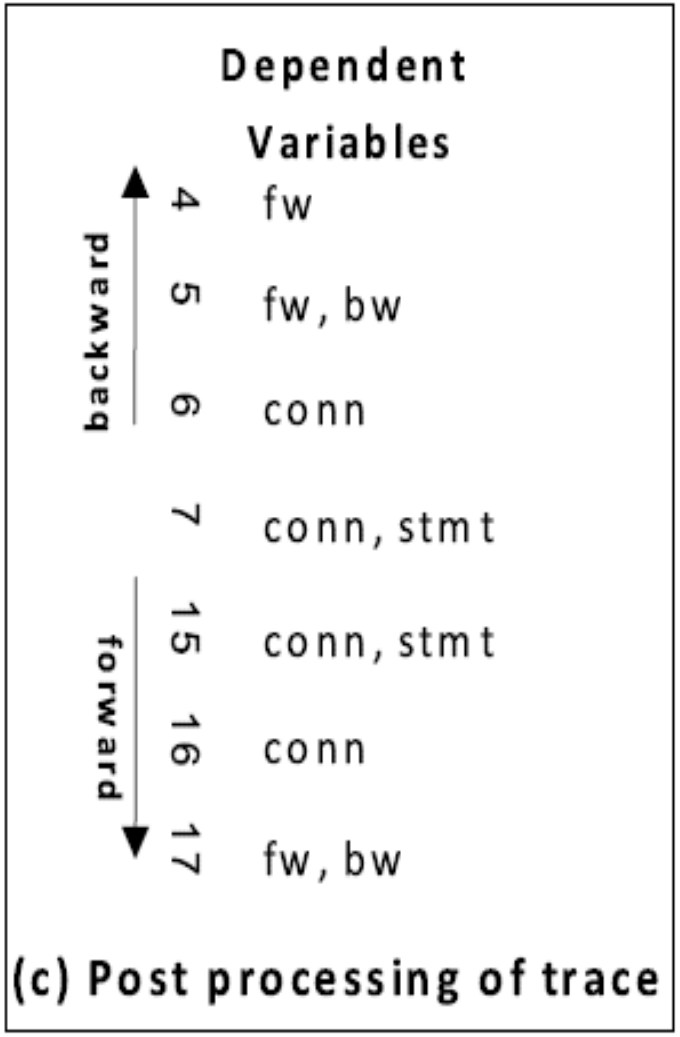
Static Trace Generation

```
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: }
```

- 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)

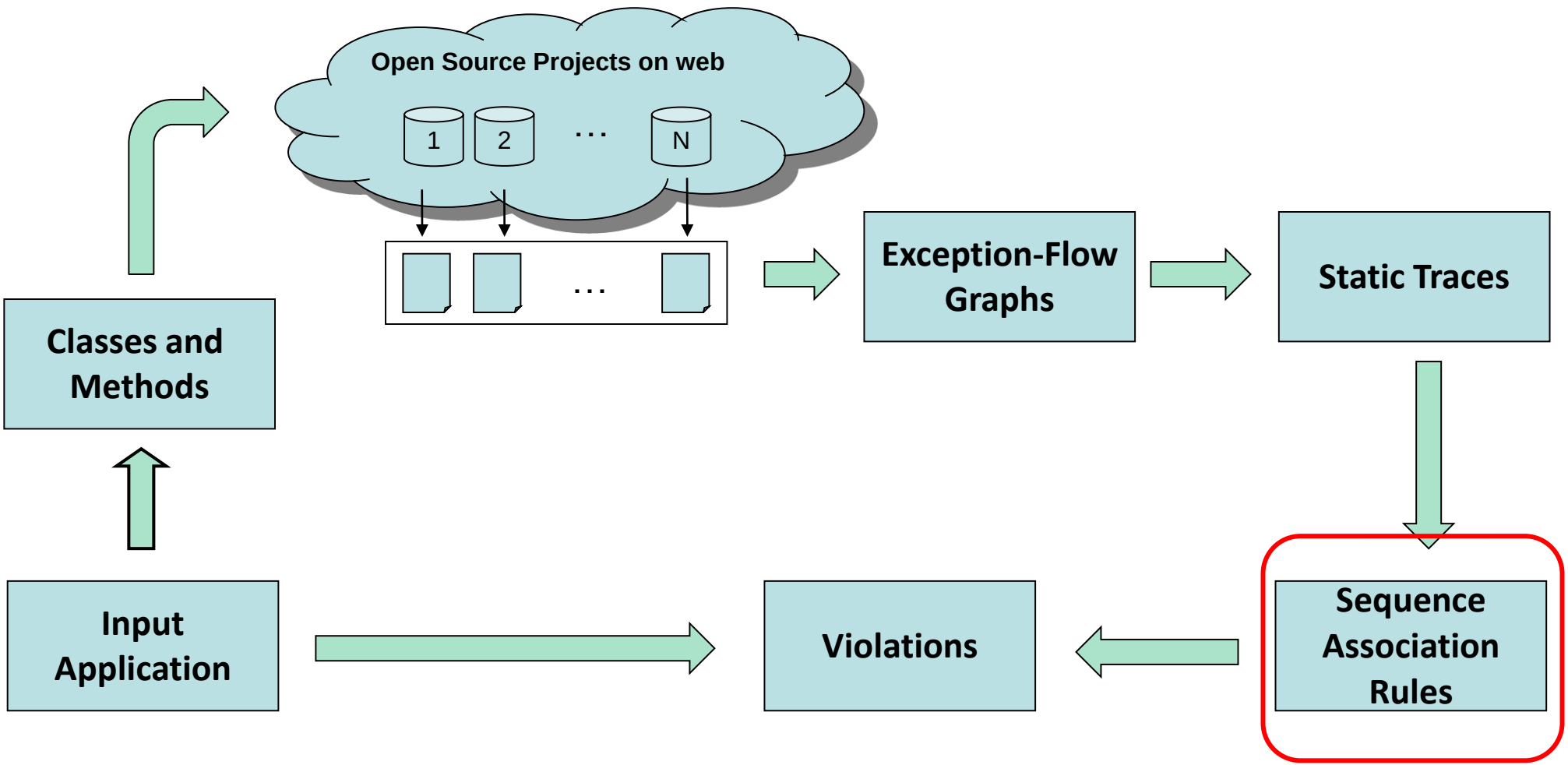
…

7: Statement stmt = conn.createStatement()

…

- Filtered sequence "6 -> 7 -> 15 -> 16"

CAR-Miner Approach



Static Trace Mining

- 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

$$(FCc1 \ldots FCcn) \wedge FCa \Rightarrow FCe1 \ldots FCen$$

Context *Trigger* *Recovery*

Mining Problem Definition

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

Context	*Recovery*
SDB1	**SDB2**
3,6,9,10	2,3,7,8
3,10,13	2,6,8
9,10,1,19	9,16,13

- Objective: To get association rules of the form

FC1 FC2 ... FCm -> FE1 FE2 ... FEn

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

$SDB_{1,2}$

$3^1,6^1,9^1,10^1,2^2,3^2,7^2,8^2$

$3^1,10^1,13^1,2^2,6^2,8^2$

$9^1,10^1,1^1,19^1,9^2,16^2,13^2$

- Apply frequent subsequence mining algorithm [Wang and Han, ICDE 04] to get frequent sequences

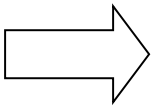
$SDB_{1,2}$

$3^1,10^1,2^2,8^2$

- Transform mined sequences into sequence association rules

Association Rule

3, 10 => 2,8

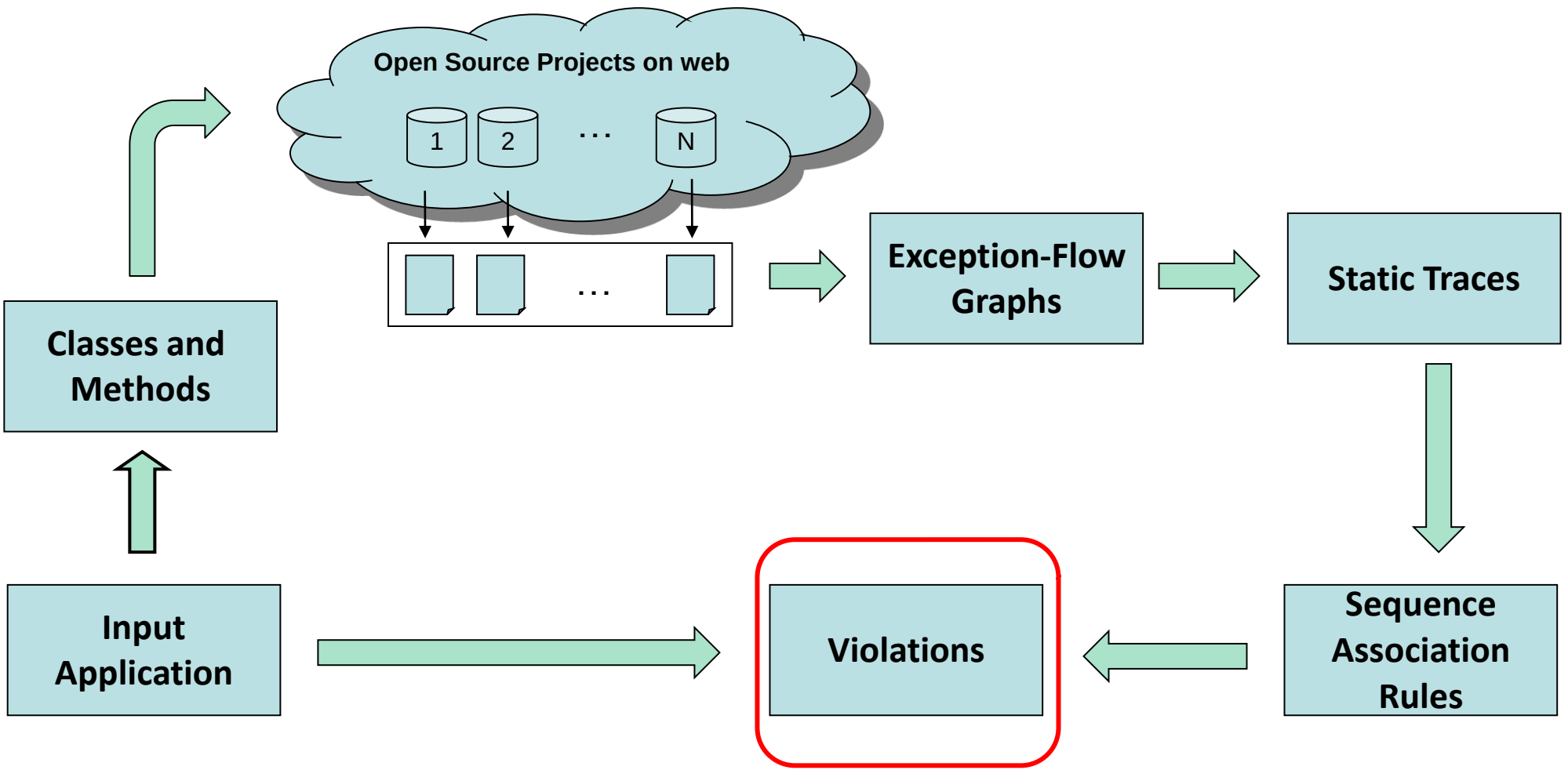


(3 10) Λ FCa => (2 8)

Context *Trigger* *Recovery*

- Rank rules based on the support assigned by frequent subsequence mining algorithm

CAR-Miner Approach



Violation Detection

- 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 ... FCen } in any exception path as violations

Evaluation

- 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?

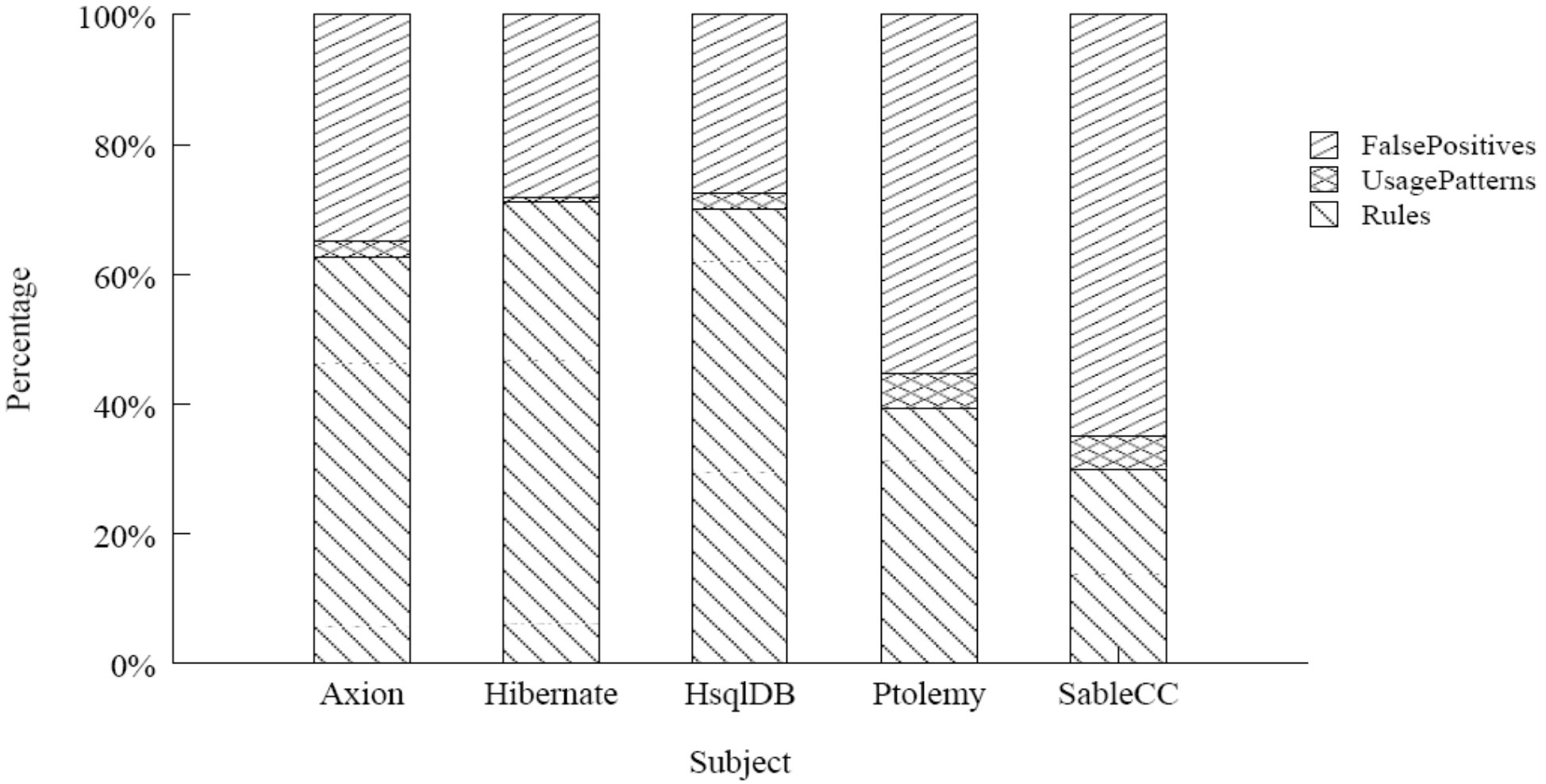
Subjects

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

- 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?



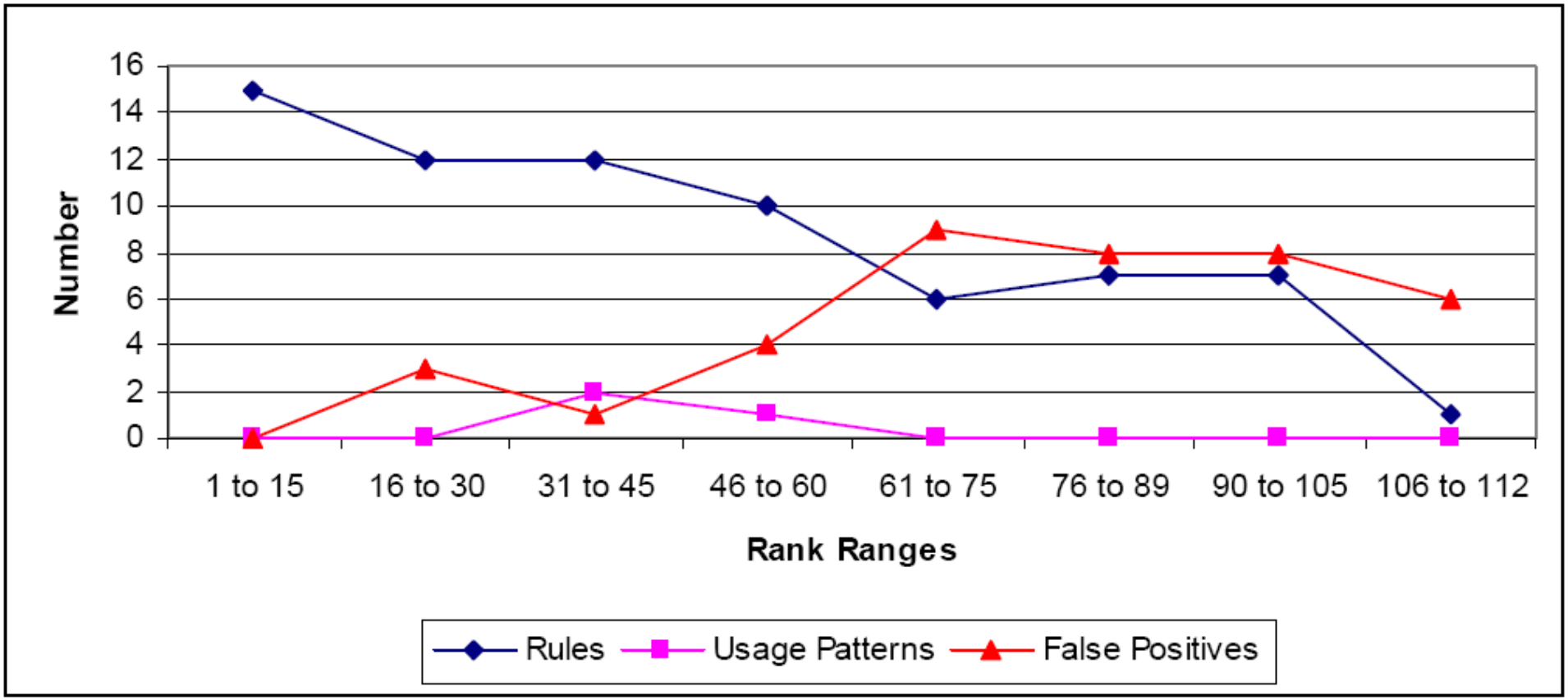
Real rules: 55% (Total: 294)

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?

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

- Total number of defects: 160
- New defects not found by WN-Miner approach: 87

RQ2: Status of Detected Violations

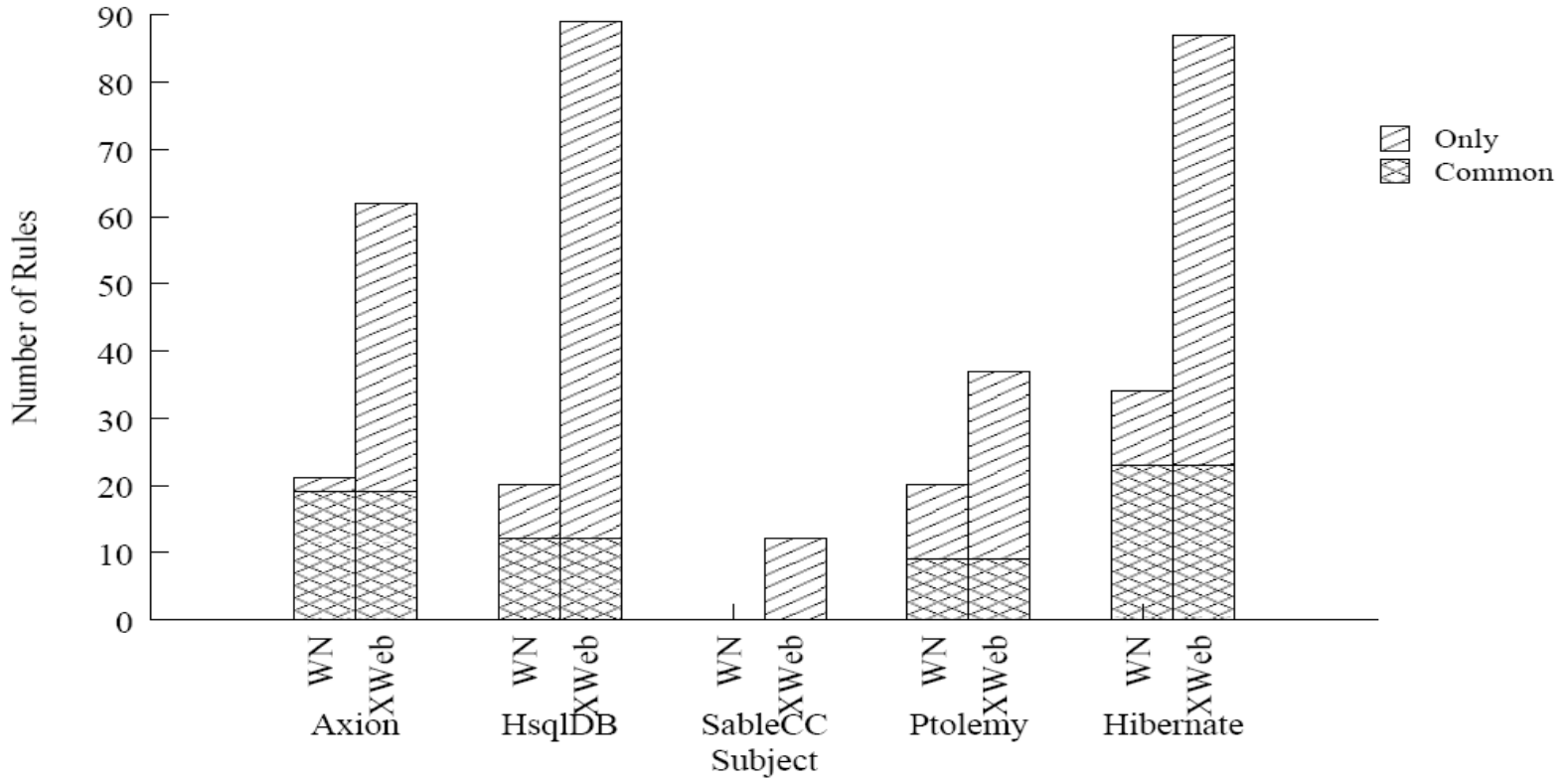
	# Defects	New Version	#Fixed	#Deleted	#Open
Axion 1.0M2	13	1.0M3	4	8	1
HsqlDB 1.7.1	51	1.8.0.9	2	9	40
Hibernate 2.0 b4	12	3.2.6	0	8	4
Sablecc 2.18.2	45	4-alpha.3	0	43	2
Ptolemy 3.0.2	39	3.0.2	0	0	39

- 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?

	# Rules	# Violations	# Defects	# Hints	# False Positives
Axion	3	6	4	0	2
HsqlDB	6	14	8	0	6
Hibernate	4	10	8	0	2
Sablecc	0	0	0	0	0
Ptolemy	1	1	1	0	0

- 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

$$(FCc1 \ldots FCcn) \wedge FCa \Rightarrow (FCe1 \ldots Fcen)$$

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