DOCUMENTING DATABASE USAGES AND SCHEMA CONSTRAINTS IN DATABASE-CENTRIC APPLICATIONS

ISSTA’16

Mario Linares-Vásquez, Boyang Li, Christopher Vendome, Denys Poshyvanyk

WILLIAM & MARY

Universidad de los Andes
Colombia
Database-Centric Applications (DCAs)

Source code

- High level features require different database operations
- SQL-style operations
- API calls using de-facto libraries or ORM frameworks
Database-Centric Applications (DCAs)

**Source code**
- High level features require different database operations
- SQL-style operations
- API calls using de-facto libraries or ORM frameworks

**Database**
- Tables, columns, types, constraints
- Underlying domain model of DCAs, including business rules and terms
Some challenges in DCAs

- DB schema and source code evolves collaterally (i.e., asynchronously)

- Schema changes have significant impact on DCAs’ code
Some challenges in DCAs

- Lack of usage of referential integrity in schemas impact the understanding of the schemas

- Tracing DB schema constraints along source code method call-chains is a “moderate” or a “very hard” challenge
Example: Xinco

- Document Management System
- In active development since 2004
- J2EE Architecture + Web Services
- 23 tables and 135 attributes
- Referential integrity

https://sourceforge.net/p/xinco/
public void deleteFromDB(boolean delete_this, XincoDBManager DBM, int userID) throws XincoException {
    int i = 0;
    try {
        Statement stmt;
        fillXincoCoreNodes(DBM);
        fillXincoCoreData(DBM);
        for (i = 0; i < getXinco_core_nodes().size(); i++) {
            ((XincoCoreNodeServer)getXinco_core_nodes().elementAt(i)).deleteFromDB(true, DBM, userID);
        }
        for (i = 0; i < getXinco_core_data().size(); i++) {
            XincoIndexer.removeXincoCoreData(...);
            XincoCoreDataServer.removeFromDB(...);
            [...]
        }
        if (delete_this) {
            XincoCoreAuditServer audit = new XincoCoreAuditServer();
            stmt = DBM.con.createStatement();
            stmt.executeUpdate("DELETE FROM xinco_core_ace WHERE xinco_core_node_id=" + getId());
            stmt.close();
            audit.updateAuditTrail("xinco_core_node", new String [] {
                "id =" + getId()}, DBM, "audit.general.delete", userID);
            stmt = DBM.con.createStatement();
            stmt.executeUpdate("DELETE FROM xinco_core_node WHERE id=" + getId());
            stmt.close();
            [...]
        }
        DBM.con.commit();
    } catch (Exception e) {
        [...]
    }
}
public void deleteFromDB(boolean delete_this, XincoDBManager DBM, int userID)
    throws XincoException {
    int i=0;
    try {
        Statement stmt;
        fillXincoCoreNodes(DBM);
        fillXincoCoreData(DBM);
        for (i=0;i<getXinco_core_nodes().size();i++) {
            ((XincoCoreNodeServer)getXinco_core_nodes().elementAt(i)).
                deleteFromDB(true, DBM,userID);
        }
        for (i=0;i<getXinco_core_data().size();i++) {
            XincoIndexer.removeXincoCoreData(...);
            XincoCoreDataServer.removeFromDB(...);
        }
        if (delete_this) {
            XincoCoreAuditServer audit= new XincoCoreAuditServer();
            stmt = DBM.con.createStatement();
            stmt.executeUpdate("DELETE FROM xinco_core_ace WHERE 
                xinco_core_node_id=" + getId());
            stmt.close();
            audit.updateAuditTrail("xinco_core_node",new String [] {"id ="+getId()},
                DBM,"audit.general.delete",userID);
            stmt = DBM.con.createStatement();
            stmt.executeUpdate("DELETE FROM xinco_core_node WHERE id=" + getId());
            stmt.close();
        }
        DBM.con.commit();
    } catch (Exception e) {
        [...]
    }
}
```java
public void deleteFromDB(boolean delete_this, XincoDBManager DBM, int userID) throws XincoException {
    int i=0;
    try {
        Statement stmt;
        fillXincoCoreNodes(DBM);
        fillXincoCoreData(DBM);
        for (i=0; i<getXinco_core_nodes().size(); i++) {
            ((XincoCoreNodeServer)getXinco_core_nodes().elementAt(i)).deleteFromDB(true, DBM, userID);
        }
        for (i=0; i<getXinco_core_data().size(); i++) {
            XincoIndexer.removeXincoCoreData(...);
            XincoCoreDataServer.removeFromDB(...);
        }
        if (delete_this) {
            XincoCoreAuditServer audit= new XincoCoreAuditServer();
            stmt = DBM.con.createStatement();
            stmt.executeUpdate("DELETE FROM xinco_core_ace WHERE xinco_core_node_id=" + getId());
            stmt.close();
            audit.updateAuditTrail("xinco_core_node",new String []{"id ="+getId(),
                "DBM","audit.general.delete",userID});
            stmt = DBM.con.createStatement();
            stmt.executeUpdate("DELETE FROM xinco_core_node WHERE id=" + getId());
            stmt.close();
        }
        DBM.con.commit();
    } catch (Exception e) {
        [...] 
    }
}
```
Example: com.bluecubs.xinco.core.server.XincoCoreNodeServer.deleteFromDB

```java
public void deleteFromDB(boolean delete_this, XincoDBManager DBM, int userID)
    throws XincoException {
    int i = 0;
    try {
        Statement stmt;
        fillXincoCoreNodes(DBM);
        fillXincoCoreData(DBM);
        for (i = 0; i < getXinco_core_nodes().size(); i++) {
            ((XincoCoreNodeServer)getXinco_core_nodes().elementAt(i)).
deleteFromDB(true, DBM, userID);
        }
        for (i = 0; i < getXinco_core_data().size(); i++) {
            XincoIndexer.removeXincoCoreData(...);
            XincoCoreDataServer.removeFromDB(...);
        }
        if (delete_this) {
            XincoCoreAuditServer audit = new XincoCoreAuditServer();
            stmt = DBM.con.createStatement();
            stmt.executeUpdate("DELETE FROM xinco_core_ace WHERE
oxinco_core_node_id=" + getId());
            stmt.close();
            audit.updateAuditTrail("xinco_core_node", new String [] {"id ="+getId()},
                DBM, "audit.general.delete", userID);
            stmt = DBM.con.createStatement();
            stmt.executeUpdate("DELETE FROM xinco_core_node WHERE id=" + getId());
            stmt.close();
        }
        DBM.con.commit();
    } catch (Exception e) {
        [...]
    }
}
```
Motivation

1. Understanding DB usages and schema constraints is crucial to understand how features are implemented.

2. Manually documenting/investigating DB usages is time consuming.

3. Less information about the DB can be inferred from source code methods at higher levels of the call-chains.
1. Updated documentation at method-level

3. Textual description of db-usages and related constraints

3. Local and delegated db operations

4. Automatic documentation for different layers in a DCA

5. Combines static-analysis and summarization techniques
1. SQL queries/statements detection

DBScribe finds API calls that execute SQL-statements, and the corresponding SQL literals

```
public CourseSchedule(int offerID){
    try{
        Connection conn = Database.getConnection();
        String selectpart = "Select *";
        String fromPart = " FROM courseschedule"
        String wherePart = " WHERE offerID= ?";
        String scheduleSelect = selectpart + fromPart + wherePart;
        [...] // code to prepare and execute SQL statement
        PreparedStatement statement = conn.prepareStatement(scheduleSelect);
        [...] // code to execute SQL statement
    } catch(SQLException e){
        System.out.println("Error retrieving schedule");
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}
```
1. SQL queries/statements detection

DBScribe finds API calls that execute SQL-statements, and the corresponding SQL literals

```java
public CourseSchedule(int offerID){
    try{
        Connection conn = Database.getConnection();
        String selectpart = "Select *";
        String fromPart = " FROM courseschedule"
        String wherePart = " WHERE offerID= ?";
        String scheduleSelect = selectpart + fromPart + wherePart;
        ...[
        PreparedStatement statement = conn.prepareStatement(scheduleSelect);
        ...[
    }catch(SQLException e){
        System.out.println("Error retrieving schedule");
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}
```
1. SQL queries/statements detection

DBScribe finds API calls that execute SQL-statements, and the corresponding SQL literals

```java
public CourseSchedule(int offerID){
    try{
        Connection conn = Database.getConnection();
        String selectpart = "Select *";
        String fromPart = " FROM courseschedule";
        String wherePart = " WHERE offerID= ?";
        String scheduleSelect = selectpart + fromPart + wherePart;
        [..]
        PreparedStatement statement = conn.prepareStatement(scheduleSelect);
        [..]
    }catch(SQLException e){
        System.out.println("Error retrieving schedule");
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}
```
1. SQL queries/statements detection

DBScribe finds API calls that execute SQL-statements, and the corresponding SQL literals

```java
public CourseSchedule(int offerID){
    try{
        Connection conn = Database.getConnection();
        String selectpart = "Select *";
        String fromPart = " FROM courseschedule"
        String wherePart = " WHERE offerID= ?";
        String scheduleSelect = selectpart + fromPart + wherePart;
        [...]
        PreparedStatement statement = conn.prepareStatement(scheduleSelect);
        [...]
    }catch(SQLException e){
        System.out.println("Error retrieving schedule");
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}
```

Variables Map

- **selectpart** -> "Select *
- **fromPart** -> " FROM courseschedule"
- **wherePart** -> " WHERE offerID= ?"

SQL-related calls
1. SQL queries/statements detection

DBScribe finds API calls that execute SQL-statements, and the corresponding SQL literals

```java
public CourseSchedule(int offerID){
    try{
        Connection conn = Database.getConnection();
        String selectpart = "Select *";
        String fromPart = " FROM courseschedule"
        String wherePart = " WHERE offerID= ?";
        String scheduleSelect = selectpart + fromPart + wherePart;
        PreparedStatement statement = conn.prepareStatement(scheduleSelect);
    }catch(SQLException e){
        System.out.println("Error retrieving schedule");
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}
```

**VariablesMap**

- `selectpart` -> "Select *"
- `fromPart` -> " FROM courseschedule"
- `wherePart` -> " WHERE offerID= ?"
- `scheduleSelect` -> "Select * FROM courseschedule WHERE offerID= ?"
1. SQL queries/statements detection

DBScribe finds API calls that execute SQL-statements, and the corresponding SQL literals

```java
public CourseSchedule(int offerID){
    try{
        Connection conn = Database.getConnection();
        String selectpart = "Select *";
        String fromPart = " FROM courseschedule";
        String wherePart = " WHERE offerID= ?";
        String scheduleSelect = selectpart + fromPart + wherePart;
        PreparedStatement statement = conn.prepareStatement(scheduleSelect);
        [...] 
    } catch(SQLException e){
        System.out.println("Error retrieving schedule");
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}
```

**VariablesMap**

- `selectpart` -> “Select *”
- `fromPart` -> “ FROM courseschedule”
- `wherePart` -> “ WHERE offerID= ?”

**SQL-related calls**

```
Select * FROM course schedule WHERE offerID= ?
```
1. SQL queries/statements detection

SQL-related calls in method $m$

```
Select * FROM course schedule WHERE offerID= ?

INSERT INTO investment VALUES (ssn,capitalGains,capitalLosses, stockDividend)
```

![JSqlParser](http://jsqlparser.sourceforge.net/)

\[
< \text{literal}_1, \text{operation-type}_1, \text{tables}_1, \text{fields}_1 >
\]

\[
\cdot
\]

\[
< \text{literal}_n, \text{operation-type}_n, \text{tables}_n, \text{fields}_n >
\]
Given a set of methods $M$ invoking SQL statements/queries, DBScribe finds the set of call-chains that end at any method in $M$, based on the callers sets.

$M=\{1,2,3\}$
Given a set of methods $M$ invoking SQL statements/queries, DBScribe finds the set of call-chains that end at any method in $M$, based on the callers sets.

**Call chains:**

$4 \rightarrow 1$

$M = \{1, 2, 3\}$
Given a set of methods $M$ invoking SQL statements/queries, DBScribe finds the set of call-chains that end at any method in $M$, based on the callers sets.

**Call chains:**

- $4 \rightarrow 1$
- $5 \rightarrow 1$

$M = \{1, 2, 3\}$
Given a set of methods $M$ invoking SQL statements/queries, DBScribe finds the set of call-chains that end at any method in $M$, based on the callers sets.

**Call chains:**

$6 \rightarrow 4 \rightarrow 1$

$5 \rightarrow 1$

$M=\{1,2,3\}$
2. Partial call graph extraction

Given a set of methods \( M \) invoking SQL statements/queries, DBScribe finds the set of call-chains that end at any method in \( M \), based on the callers sets

**Call chains:**

8 -> 6 -> 4 -> 1
5 -> 1

\( M = \{1,2,3\} \)
Given a set of methods $M$ invoking SQL statements/queries, DBScribe finds the set of call-chains that end at any method in $M$, based on the callers sets.

**Call chains:**
- $12 \rightarrow 8 \rightarrow 6 \rightarrow 4 \rightarrow 1$
- $5 \rightarrow 1$

$M=\{1,2,3\}$
Given a set of methods $M$ invoking SQL statements/queries, DBScribe finds the set of call-chains that end at any method in $M$, based on the callers sets.

**Call chains:**

12 -> 8 -> 6 -> 4 -> 1
7 -> 5 -> 1

$M=\{1,2,3\}$
2. Partial call graph extraction

Given a set of methods $M$ invoking SQL statements/queries, DBScribe finds the set of call-chains that end at any method in $M$, based on the callers sets.

**Call chains:**

12 -> 8 -> 6 -> 4 -> 1
11 -> 7 -> 5 -> 1

$M=\{1,2,3\}$
3. DB schema constraints extraction

- Auto-numeric columns
- Non-null columns
- Foreign keys
- Varchar limits
- Columns that should contain unique values
4. DB usages and constraints propagation

Given a set of *constraints* \((C)\) and *db operations* \((DBO)\) invoked by each method, DBScribe propagates (iteratively) them through the call chains, from the bottom.
Given a set of constraints (C) and db operations (DBO) invoked by each method, DBScribe propagates (iteratively) them through the call chains, from the bottom.
Given a set of constraints \( (C) \) and db operations \( (DBO) \) invoked by each method, DBScribe propagates (iteratively) them through the call chains, from the bottom.
Given a set of **constraints** \((C)\) and **db operations** \((DBO)\) invoked by each method, DBScribe propagates (iteratively) them through the call chains, from the bottom.

**Diagram:**

- **Level 0:**
  - Node 1: \(\{C_1, DBO_1\}\)
  - Node 2: \(\{C_2, DBO_2\}\)

- **Level 1:**
  - Node 3: \(\{C_1 \cup C_2 \cup C_3, DBO_1 \cup DBO_2 \cup DBO_3\}\)
  - Node 4: \(\{C_2, DBO_2\}\)

- **Level 2:**
  - Node 5: \(\{\}\)
  - Node 6: \(\{\}\)

- **Level 3:**
  - Node 7: \(\{\}\)
Given a set of *constraints* (*C*) and *db operations* (*DBO*) invoked by each method, DBScribe propagates (iteratively) them through the call chains, from the bottom.
5. HTML descriptions generation

\{C_1, DBO_1\}

\{C_2, DBO_2\}

\{C_1 \cup C_2 \cup C_3, DBO_1 \cup DBO_2 \cup DBO_3\}

\ldots

\ldots

\ldots
com.umas.code.CourseOffered.addOneSeatFilledToCourseOffered()

This method implements the following db-related operations:

- It queries the table(s) COURSEOFFERED

- It updates the SeatsFilled attribute(s) in table COURSEOFFERED

This method invokes db-related operations via delegation:

- It queries the table(s) SEMESTER via the chain-call `com.umas.code.CourseOffered.checkIfCurrent`
  `com.umas.code.CourseOffered.getCurrentSemesterID`

Some constraints that should be taken into the account are the following:

- Make sure the values in COURSEOFFERED.SeatsFilled are not null
Example

com.umas.code.CourseOffered.addOneSeatFilledToCourseOffered()

This method implements the following db-related operations:

- It queries the table(s) COURSEOFFERED

- It updates the SeatsFilled attribute(s) in table COURSEOFFERED

This method invokes db-related operations via delegation:

- It queries the table(s) SEMESTER via the chain-call com.umas.code.CourseOffered.checkIfCurrent
  com.umas.code.CourseOffered.getCurrentSemesterID

Some constraints that should be taken into the account are the following:

- Make sure the values in COURSEOFFERED.SeatsFilled are not null
Example

com.umas.code.CourseOffered.addOneSeatFilledToCourseOffered()

This method implements the following db-related operations:

- It queries the table(s) COURSESOFFERED
- It updates the SeatsFilled attribute(s) in table COURSESOFFERED

This method invokes db-related operations via delegation:

- It queries the table(s) SEMESTER via the chain-call com.umas.code.CourseOffered.checkIfCurrent com.umas.code.CourseOffered.getCurrentSemesterID

Some constraints that should be taken into the account are the following:

- Make sure the values in COURSESOFFERED.SeatsFilled are not null
This method implements the following db-related operations:

- It queries the table(s) **COURSESOFFERED**

- It updates the **SeatsFilled** attribute(s) in table **COURSESOFFERED**

This method invokes db-related operations via delegation:

- It queries the table(s) **SEMESTER** via the chain-call `com.umas.code.CourseOffered.checkIfCurrent` → `com.umas.code.CourseOffered.get_CurrentSemesterID`

**Some constraints that should be taken into the account are the following:**

- Make sure the values in **COURSESOFFERED.SeatsFilled** are not null
Example

`com.umas.code.Admin.addAdmin(String, Department)`

This method invokes db-related operations via delegation:

- It inserts the `UIN`, `Salary`, `OfficeAddress`, `OfficeHours` attributes into table `EMPLOYEE` via a call to the `com.umas.code.Employee.addEmployee` method

- It queries the table(s) `EMPLOYEE` via the chain-call `com.umas.code.Employee.addEmployee ➔ com.umas.code.Employee.addEmployeeCheck`

Some constraints that should be taken into the account are the following:

- Make sure the strings to be stored in `EMPLOYEE` do not overflow the varchar limits: `45` (`OfficeAddress`, `OfficeHours`)

- Make sure the values in `EMPLOYEE.Salary` are not null

- Make sure the values in `EMPLOYEE.UIN` are not null

- Make sure the values of attribute `EMPLOYEE.UIN` are unique because there is a UNIQUENESS constraint

- When inserting into table `EMPLOYEE`, make sure the referential integrity imposed by attribute(s) `UIN` is accomplished. The foreign keys in the table are the following: `(UIN ➔ people.UIN)`
5. HTML descriptions generation

\( \{C_1, DBO_1\} \)

\( \{C_2, DBO_2\} \)

\( \{C_1 \cup C_2 \cup C_3, DBO_1 \cup DBO_2 \cup DBO_3\} \)

\( \ldots \)

\( \ldots \)

\( \ldots \)

\( \ldots \)

\( m \)
### Methods with local invocations:

1. **com.bluecubs.xinco.add.server.XincoAddAttributeServer.XincoAddAttributeServer(int, int, XincoDBManager)**
   - **This method implements the following db-related operations:**
     - It queries the table(s) `XINCO_ADD_ATTRIBUTE`

2. **com.bluecubs.xinco.add.server.XincoAddAttributeServer.getXincoAddAttributes(int, XincoDBManager)**
   - **This method implements the following db-related operations:**
     - It queries the table(s) `XINCO_ADD_ATTRIBUTE`

3. **com.bluecubs.xinco.add.server.XincoAddAttributeServer.write2DB(XincoDBManager)**
   - **This method implements the following db-related operations:**
     - It inserts values for the first 8 columns into table `XINCO_ADD_ATTRIBUTE`
   - **Some constraints that should be taken into the account are the following:**
     - Make sure the strings to be stored in `XINCO_ADD_ATTRIBUTE` do not overflow the varchar limits: 65535 (attrib_text), 255 (attrib_varchar)
     - When inserting into table `XINCO_ADD_ATTRIBUTE`, make sure the referential integrity imposed by attribute(s) `xinco.core.data_id` is accomplished. The foreign keys in the table are the following: `(xinco.core.data_id ➔ xinco.core.data.id)`
Limitations

- Current implementation supports **MySQL Server, JDBC, and Hibernate**

- Call graph extraction is **path insensitive** (over-approximation)

- **No inter-procedural analysis** for strings concatenation/replacement in SQL literals
Empirical Study

5+2 Systems

52 Participants

2 Original developers
Empirical Study

5+2 Systems
52 Participants
2 Original developers

1. Quality of the descriptions: completeness, conciseness, expressiveness
2. Usefulness and user preferences
3. Industrial applicability
Quality of the descriptions

30 randomly selected **methods** and their **descriptions** (6 per system)

OS systems → **DBScribe**

Participant vs DBScribe
- Completeness
- Conciseness
- Expressiveness
- User preferences
- Usefulness

Descriptions (method level)

6 methods per participant

Participants
Quality of the descriptions

**Completeness**

- 66% Does not miss any important info
- 29% Misses some important info
- 5% Misses most important info

**Conciseness**

- 71% Contains no redundant info
- 25% Contains some redundant info
- 4% Contains a lot of redundant info
Quality of the descriptions

Expressiveness

77% Is easy to read
19% Is somewhat readable
4% Is hard to read/understand
Usefulness

<table>
<thead>
<tr>
<th>Usefulness</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>48/52</td>
<td></td>
</tr>
</tbody>
</table>

Useful for understanding the database usages in source code methods

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental change</td>
<td>21</td>
</tr>
<tr>
<td>Maintenance</td>
<td>10</td>
</tr>
<tr>
<td>Bugs fixing</td>
<td>10</td>
</tr>
<tr>
<td>Others (e.g. test cases design)</td>
<td>15</td>
</tr>
</tbody>
</table>
Industrial applicability

- Completeness
- Conciseness
- Expressiveness
- User preferences
- Usefulness

2 industrial systems

DBScribe

HTML reports

Original developers/maintainers
Industrial applicability

DBScribe is useful for incremental change and maintenance

“Based on the descriptions you can be aware all dependencies a table could have. It would let you estimate in a better way the impact due to future changes.”

“It helps you create a quick vision of the system with the basic method and code structure without looking at actual source code”
“The link system for call-chains works only in one way, one could get lost navigating a complex system [...] A navigation tree might be useful in this case.”

“you should extend the approach to include JPA”, “it would be better to have it in the IDE, something like right click->generate”.
Database-Centric Applications (DCAs)
**Database-Centric Applications (DCAs)**

Users → High-level features → GUI/API → Source Code → Source Code → DB

**DBScribe**

Source Code → 1. SQL queries/statements detection → 2. Partial call graph extraction → 3. DB schema constraints extraction → 4. DB usage and constraints propagation → 5. HTML description generation → Templates

**Empirical Study**

- **5+2 Systems**
- **52 Participants**
- **2 Original devs.**

1. Method description quality: completeness, conciseness, expressiveness
2. Usefulness and user preferences
3. Industrial applicability
**Database-Centric Applications (DCAs)**

**Users** → High-level features → Source Code → Source Code → DB

**DBScribe**

1. SQL queries/statements detection
2. Partial call graph extraction
3. DB schema constraints extraction
4. DB usages and constraints propagation
5. HTML description generation

**Empirical Study**

- **5+2** Systems
- **52** Participants
- **2** Original devs.

1. Method description quality: completeness, conciseness, expressiveness
2. Usefulness and user preferences
3. Industrial applicability

**Summary**

- Mostly complete, concise, and readable
- Useful for software engineering tasks
- Can be applied in industry
THANKS !!

http://www.cs.wm.edu/semeru/data/ISSTA16-DBScribe/
Table 3: Systems’ statistics: Lines Of Code, Tables in the DB schema, # of JDBC API calls involving SQL-Statements, # of SQL statements that DB-Scribe was Not able to Parse, # of Methods declaring SQL-statements Locally (ML), via Delegation (MD), Locally + Delegation (MLD), execution Time in sec.

<table>
<thead>
<tr>
<th>System</th>
<th>LOC</th>
<th>TB</th>
<th>S</th>
<th>NP</th>
<th>ML</th>
<th>MD</th>
<th>MLD</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMAS [8]</td>
<td>32K</td>
<td>122</td>
<td>211</td>
<td>4</td>
<td>125</td>
<td>431</td>
<td>67</td>
<td>29.53</td>
</tr>
<tr>
<td>Riskit rev.96 [7]</td>
<td>12.7K</td>
<td>13</td>
<td>111</td>
<td>2</td>
<td>35</td>
<td>9</td>
<td>44</td>
<td>15.02</td>
</tr>
<tr>
<td>FINA 3.4.2 [2]</td>
<td>139.5K</td>
<td>52</td>
<td>710</td>
<td>26</td>
<td>312</td>
<td>118</td>
<td>99</td>
<td>130.78</td>
</tr>
<tr>
<td>Xinco rev.700 [9]</td>
<td>25.6K</td>
<td>23</td>
<td>76</td>
<td>15</td>
<td>26</td>
<td>22</td>
<td>21</td>
<td>31.41</td>
</tr>
<tr>
<td>OpenEmm 6.0 [5]</td>
<td>102.4K</td>
<td>68</td>
<td>200</td>
<td>110</td>
<td>73</td>
<td>12</td>
<td>1</td>
<td>104.78</td>
</tr>
<tr>
<td>System 1*</td>
<td>73.2K</td>
<td>53</td>
<td>398</td>
<td>27</td>
<td>262</td>
<td>660</td>
<td>24</td>
<td>71.07</td>
</tr>
<tr>
<td>System 2*</td>
<td>28.4K</td>
<td>24</td>
<td>164</td>
<td>8</td>
<td>106</td>
<td>247</td>
<td>44</td>
<td>40.13</td>
</tr>
</tbody>
</table>
Table 5: Answers to “What software engineering tasks will you use this type of summary for?”

<table>
<thead>
<tr>
<th>Category</th>
<th>Subcategories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental change (21)</td>
<td>Program comprehension (11), Add new features (4), Impact analysis (4), Concept location (1), Change database schema (1)</td>
</tr>
<tr>
<td>Bugs (10)</td>
<td>Debugging (6), Bug fixing (4)</td>
</tr>
<tr>
<td>Maintenance (10)</td>
<td>Maintenance (4), Refactoring (2), Re-modularization (2), Re-engineering (2)</td>
</tr>
<tr>
<td>Others (15)</td>
<td>Documentation (9), Change db-related code (3), Test cases design (2), Systems integration (1)</td>
</tr>
</tbody>
</table>