

# Maintainability metrics for embedded SQL

## Authors:

- Martin van der Vlist
- Serguei Roubtsov
- Mark van den Brand
- Alexander Serebrenik

# The Problem

```
//running example
public int getCustomerCredit(int id) throws SQLException {
    String sql = "SELECT cd.credit ";
    sql += "FROM CustomerDetails cd" +
           "WHERE cd.category = " + this.getCategory();
    if (this.restrict) {
        sql += " AND cd.restriction = 1";
    }
    sql += " AND cd.id = ?";
    PreparedStatement s = this.con.prepareStatement(sql);
    s.setInt(1, id);
    ResultSet rs = s.executeQuery();
    rs.next();
    return rs.getInt("credit");
}
```

# Approach

```
public int getCustomerCredit(int id) throws SQLException {
    String sql = "SELECT cd.credit ";
    sql += "FROM CustomerDetails cd" +
           "WHERE cd.category = " +
           this.getCategory();
    if (this.restrict) {
        sql += " AND cd.restriction = 1";
    }
    sql += " AND cd.id = ?";
    PreparedStatement s =
        this.con.prepareStatement(sql);
    s.setInt(1, id);
    ResultSet rs = s.executeQuery();
    rs.next();
    return rs.getInt("credit");
}
```

Starting point

# Approach

```
public int getCustomerCredit(int id) throws SQLException {  
    String sql = "SELECT cd.credit ";  
    sql += "FROM CustomerDetails cd" +  
        "WHERE cd.category = " +  
            this.getCategory();  
    if (this.restrict) {  
        sql += " AND cd.restriction = 1";  
    }  
    sql += " AND cd.id = ?";  
    PreparedStatement s =  
        this.con.prepareStatement(sql);  
    s.setInt(1, id);  
    ResultSet rs = s.executeQuery();  
    rs.next();  
    return rs.getInt("credit");  
}
```

# of String Parts = 6

# Of Decisions = 1

# Of Alternatives = 2

# Unknown Parts = 1

% syntactically Correct Alt.  
= 100%

- Determine starting point
- Slicing to determine fragment
- Compute metrics

# Categories

	NUP = 0	NUP > 0
0 = DON	Category I	Category III
0 > DON	Category II	Category IV

# Categories & Conjectures

	NUP = 0	NUP > 0
0 = DON	Category I	Category III
0 > DON	Category II	Category IV

## Conjectures:

- **C1:** Percentage of code fragments  $s$  with  $PCA(s) > 0$  is decreasing from categories I and II to categories III and IV.
- **C2:** Category I is larger than all other categories together.
- **C3:** Category I is becoming smaller during the evolution.

# Case studies

**Compiere**<sup>®</sup>



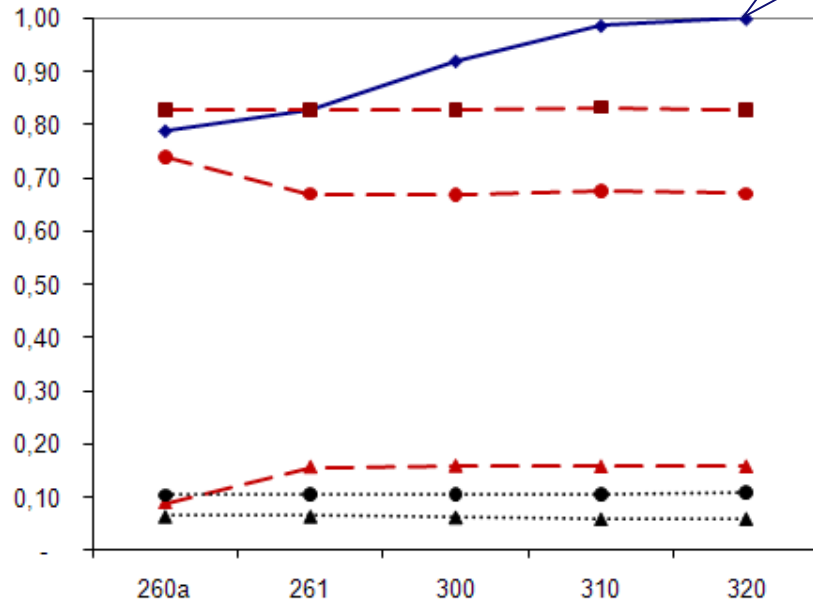
- ERP system
- GPL license
- Company-maintained, little community feedback used
- 0.7 million lines of code
- 2056 fragments detected
- 5 versions analyzed between 2006 and 2008

- ERP system
- GPL license
- Community-driven fork of Compiere
- 1.1 million lines of code
- 2631 fragments detected
- 16 versions analyzed between 2006 and Nov. 2009

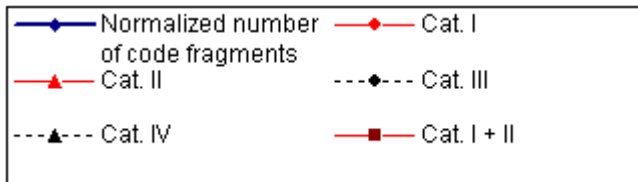
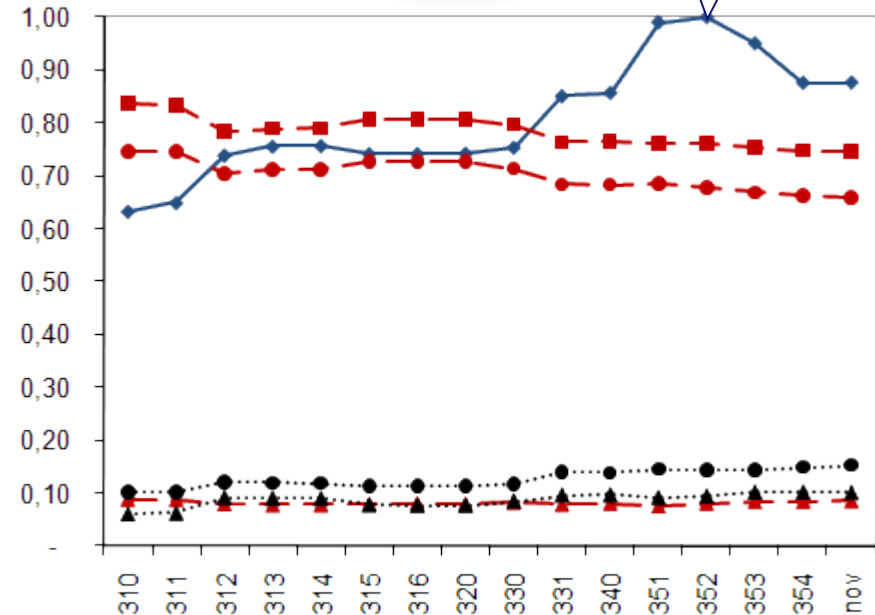
# Case studies: Categories

Compiere®

2056 fragments



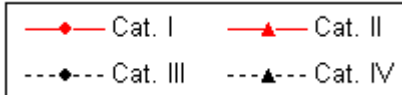
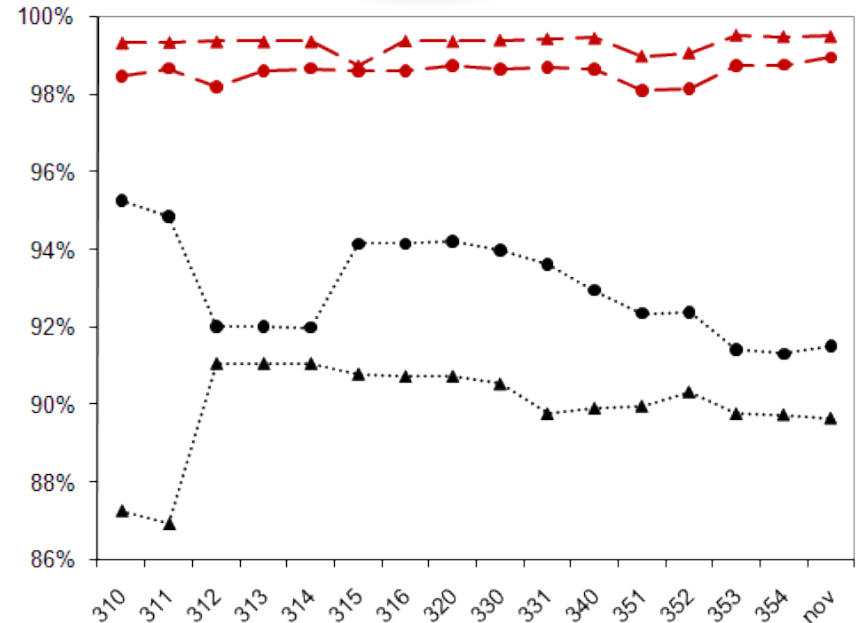
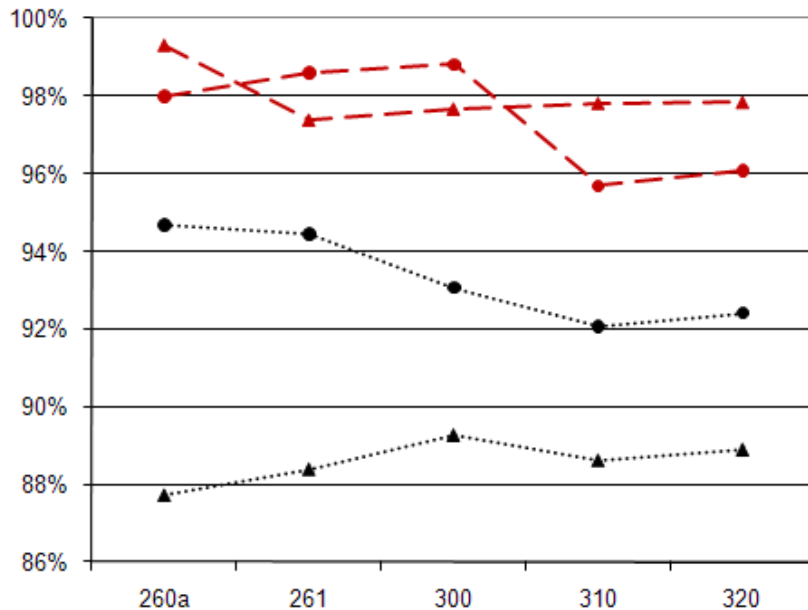
2631 fragments



# Case studies:

% syntactically Correct Alternatives > 0

Compiere®



# Conjectures on case studies

- **C1:** Percentage of code fragments  $s$  with  $PCA(s) > 0$  is decreasing from categories I and II to categories III and IV.
  - Not completely true, due to incomplete parser (no DDL support)
- **C2:** Category I is larger than all other categories together.
  - True for both case studies
- **C3:** Category I is becoming smaller during the evolution.
  - True for ADempiere, no change in Compiere

# Outcome

- **Analysis of either Java or SQL not enough**
- **Introduced 5 metrics for embedded SQL:  
NSP, NOA, NOD, NUP, PCA**
- **Introduced categorization**
- **Using these:**
  - **Insight in evolution of case studies**
  - **Insight in complexity of case studies**
- **Formulated and checked conjectures**