



東南大學

# Change Impact Analysis in the Context of Open Source Software

**Xiaobing Sun, Bixin Li**

School of Computer Science & Engineering  
Southeast University  
Nanjing, China

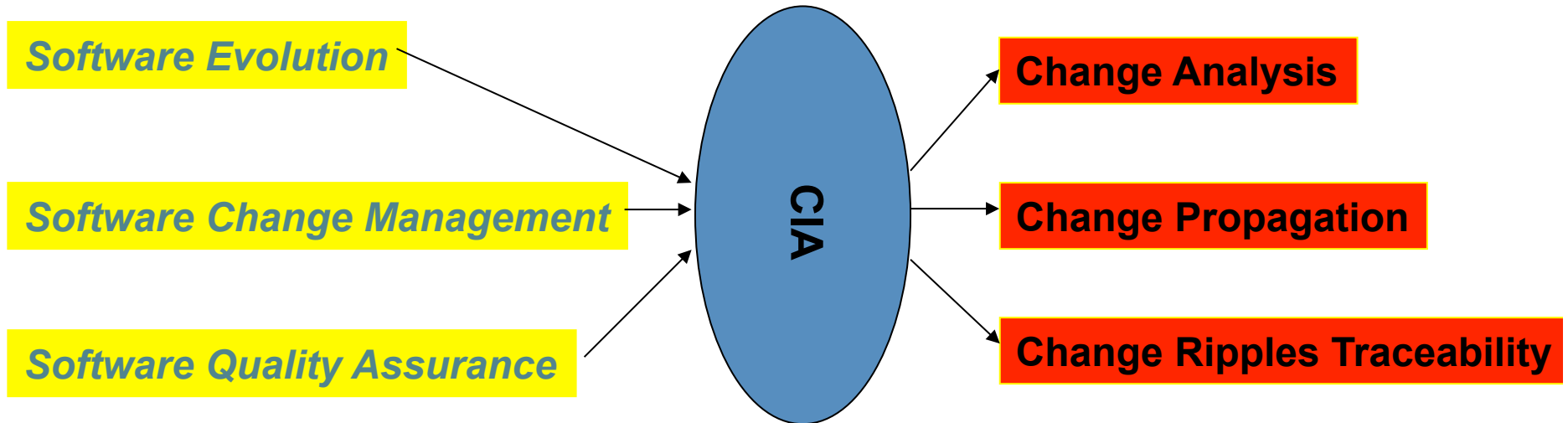
International Open Source Workshop in NUAA, 17 October 2011

# Contents

---

- ◆ **Introduction**
- ◆ **Change Impact Analysis**
- ◆ **Empirical Study**
- ◆ **Related Work**
- ◆ **Conclusion**

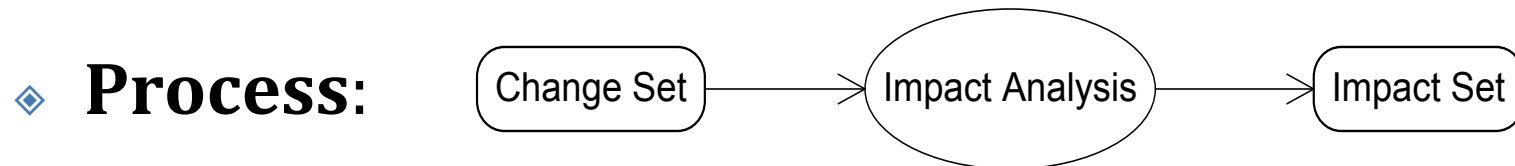
# Introduction (1/3)



**Better Software Evolution**

# Introduction (2/3)

- ◆ **Informal Definition:** To identify the potential effects caused by changes made to software (Bohner96'[52])



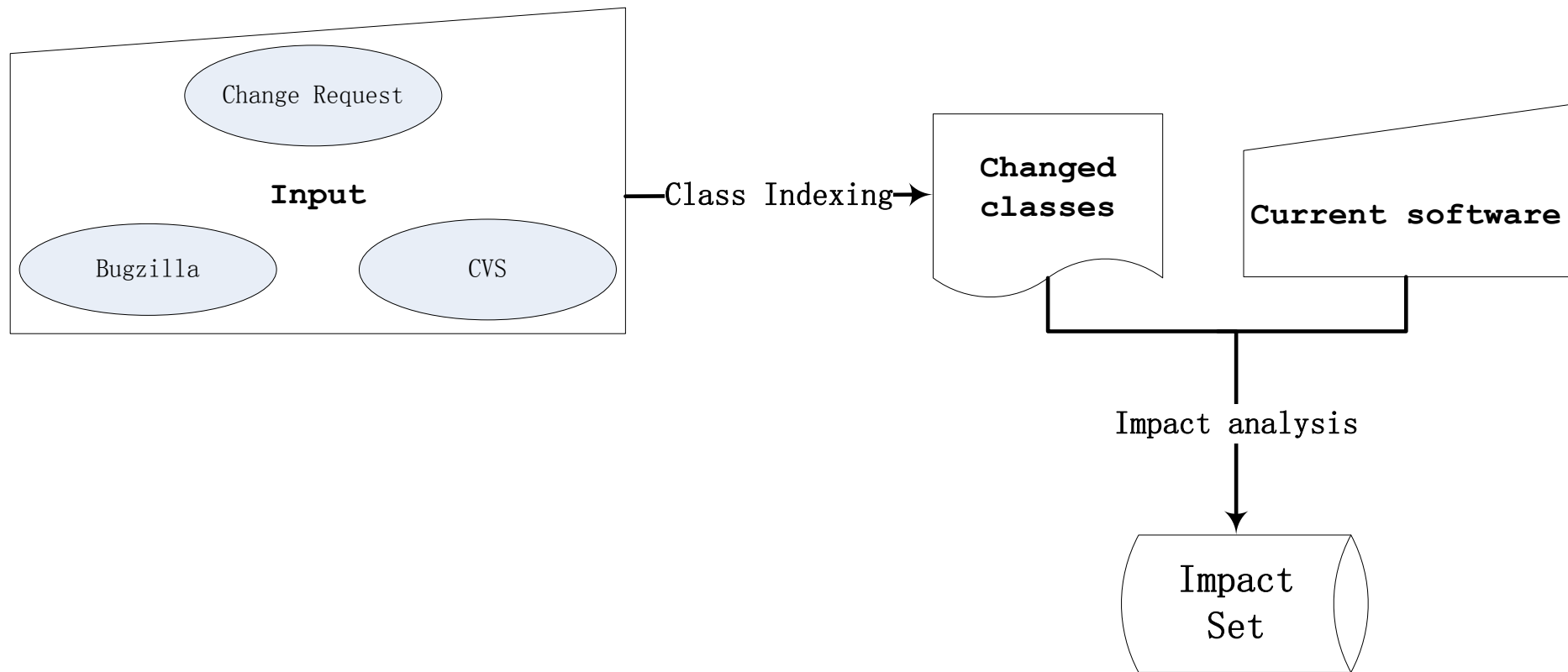
- ◆ **Evaluation:** Accuracy: Precision vs Recall (Tóth10'[71])
- ◆ **Classification:** Static vs Dynamic

# Introduction (3/3)

- ◆ Software engineering uses a systematic and strictly controlled approach, OSS projects use a more liberal approach.
- ◆ Issue (defect) management system & Version management system

- OSS Projects
  - Source Codes
  - Modification Record
    - Date, Time
    - Author
    - Change log
  - Bug-Archives
    - Creation-Time
    - End-Time
    - Description
    - Classification (Type)
    - Priority / Severity
    - Reporter / Assigned
  - Mailing Archives
    - Participants
    - Mail Contents

# Change Impact Analysis (1/7)



# Change Impact Analysis (2/7)

- ◆ Input: Version management system (CVS) and issue management system (Bugzilla), change request (Textual level).
- ◆ Output: Probable changed classes (source code level).
- ◆ Approach: Class Indexing
  - Identify similar historical change request in Bugzilla;
  - According to the change log in CVS, find all changed classes related to this change request.

# Change Impact Analysis (3/7)

- 
- ◆ Input: Current Software, changed classes.
  - ◆ Output: A ranked list of potentially impacted method.
  - ◆ Approach: Concept lattice based analysis
    - Lattice of class and method dependence
    - Impact set

# Change Impact Analysis (4/7)

---

- ◆ Formal objects: Classes
- ◆ Formal attributes: Methods
- ◆ Dependence between classes  $c$  and method  $m$ :
  1.  $m$  belongs to  $c$ ;
  2.  $m$  belongs to any superclass of  $c$ ;
  3.  $c$  depends on another method  $k$  calling  $m$ ;
  4.  $c$  depends on another method  $k$  called by  $m$ ;

# analysis (5/7)

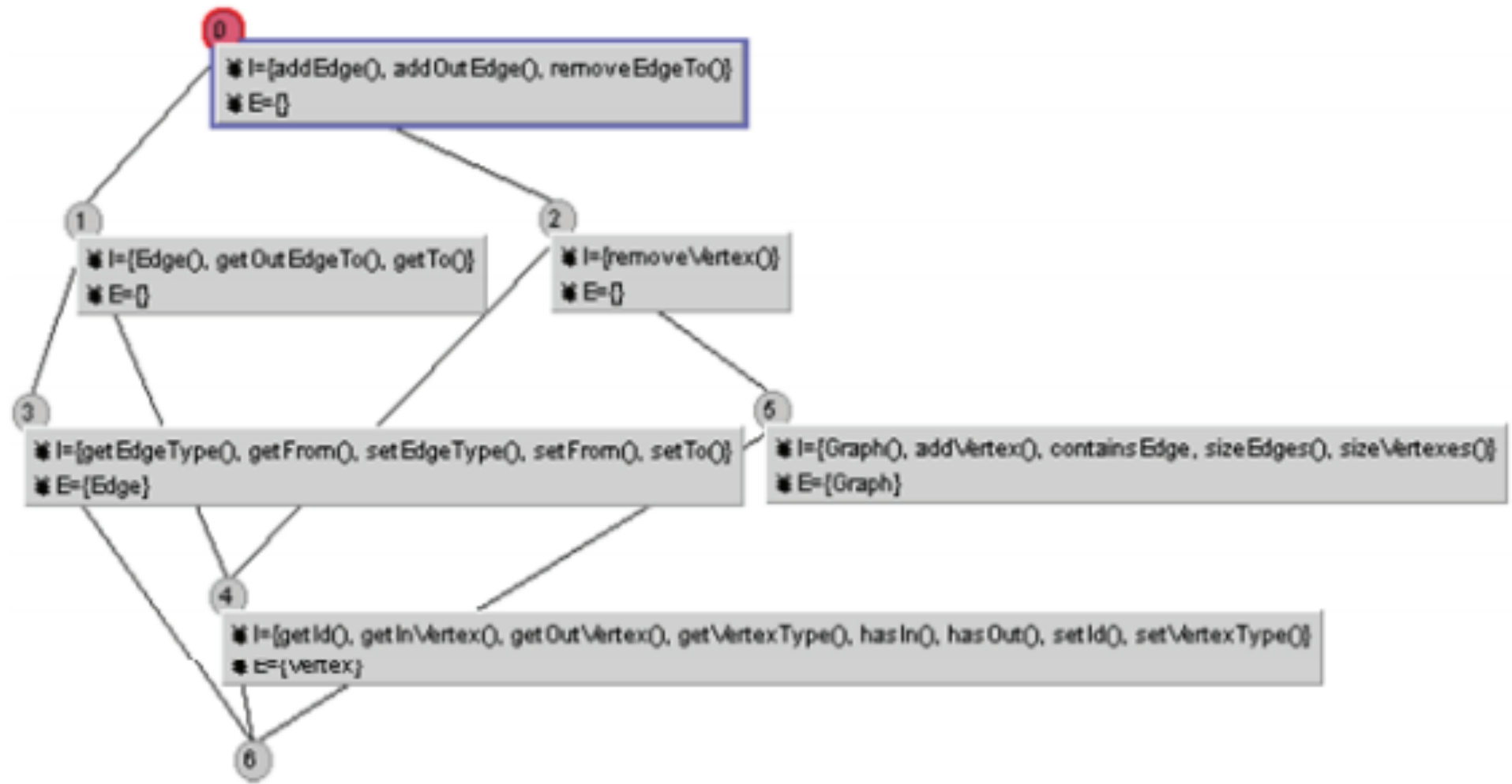
```

import java.util.*;
public class Vertex {
    private String id;
    private ArrayList out_Edges;
    private ArrayList in_Edges;
    private int vertexType;
    public int getVertexType() {
        return vertexType;
    }
    public void setVertexType(int vertexType) {
        vertexType = vertexType;
    }
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public ArrayList getOutEdges() {
        ArrayList in=new ArrayList();
        for(int i=0;i<in_edges.size();i++)
    
```

```

public class Edge {
    private Vertex from;
    private Vertex to;
    private int edgeType;
    public Edge(){
        this.from=null;
        this.to=null;
        this.edgeType=0;
    }
    public Vertex getFrom() {
        return from;
    }
    public void setFrom(Vertex from) {
        this.from = from;
    }
    public Vertex getTo() {
        return to;
    }
    public void setTo(Vertex to) {
        this.to = to;
    }
    public int getEdgeType() {
    
```

$V, E$



# Change Impact Analysis (6/7)

- ◆ **Change Set:** Classes
- ◆ **Impact Set:** Methods
- ◆ **Impact Analysis:** Two hypotheses

1) Upward reachable lattice nodes include methods that are shared by an increasing number of classes, hence they are expected to be **less and less** impacted by the change.

2) Lattice nodes upward reachable from the nodes labeled by an increasing number of changed classes (least common ancestors for these changed classes) are **more** probably affected by these joint classes changes.

Impact Factor

$$IF_j = m + \frac{1}{\sum_{i=1}^m \min(\text{dist}(i, j)) + 1}$$

# Change Impact Analysis (7/7)

- ◆ Determine the set of concept *lattice nodes*  $\{n_1, n_2, \dots, n_k\}$ , which are labeled by  $\{C_1, C_2, \dots, C_n\}$  on the *LoCMD*;
- ◆ Compute the set of lattice nodes *upward reachable* from  $\{n_1, n_2, \dots, n_k\}$ ;
- ◆ Calculate the *impact factor* of those lattice nodes obtained in the previous step;
- ◆ *Prioritize* the set of concepts according to their impact factor results, and provide the potentially impacted methods labeling these ordered concepts to maintainers for inspection stepwisely.

# Empirical Study (1/3)

---

## ◆ **Research Question**

- Usefulness of the Impact factor metric
- Effectiveness of the CIA technique

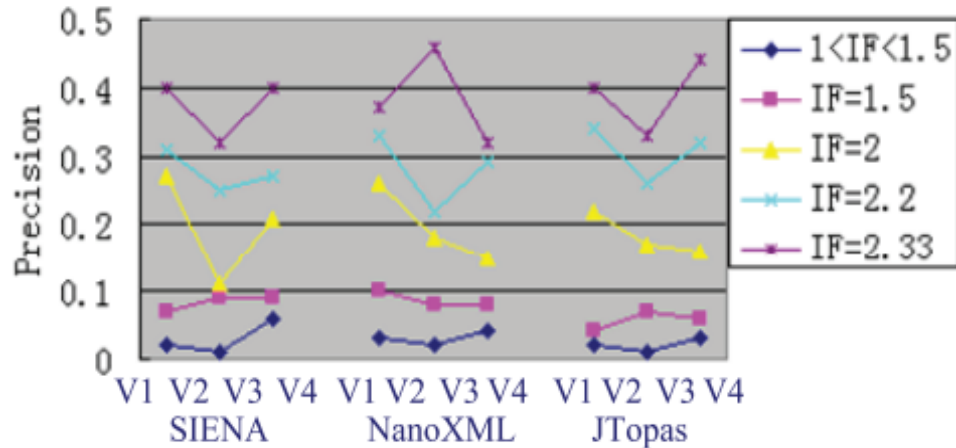
## ◆ **Measure**

- Precision (false positives)
- Recall (false negatives)

## ◆ **Three open source software systems**

- SIENA
- NanoXML
- JTopas

# Empirical Study (2/3)



Increase of the impact factor value also improves the precision of the impact set.

The Kruskal-Wallis test results for the precision of different *IF* values

| <i>Statistical results</i> | <i>Precision</i> |
|----------------------------|------------------|
| DF (degrees of freedom)    | 4                |
| Chi-square                 | 40.817           |
| Asymp. Sig.                | < 0.001          |
| Alpha                      | 0.05             |

Differences between precision values of different impact factor values are statistically significant.

# Empirical Study (3/3)

---

Precision and recall values of our CIA technique are higher than that of the conceptual coupling based CIA technique for different cut points of methods.

# Related Work

---

## ◆ **Current CIA**

- Traditional dependence analysis
- Dynamic execution information collection
- Coupling measure
- Software repository mining

## ◆ **Our CIA**

- Textual change request
- Cross-level CIA
- Consideration of the dependence between multiple changes

# Conclusion

---

- ◆ Textual change request
- ◆ A cross level CIA
- ◆ A ranked list of impact results
- ◆ Consideration of the dependence between multiple changes

