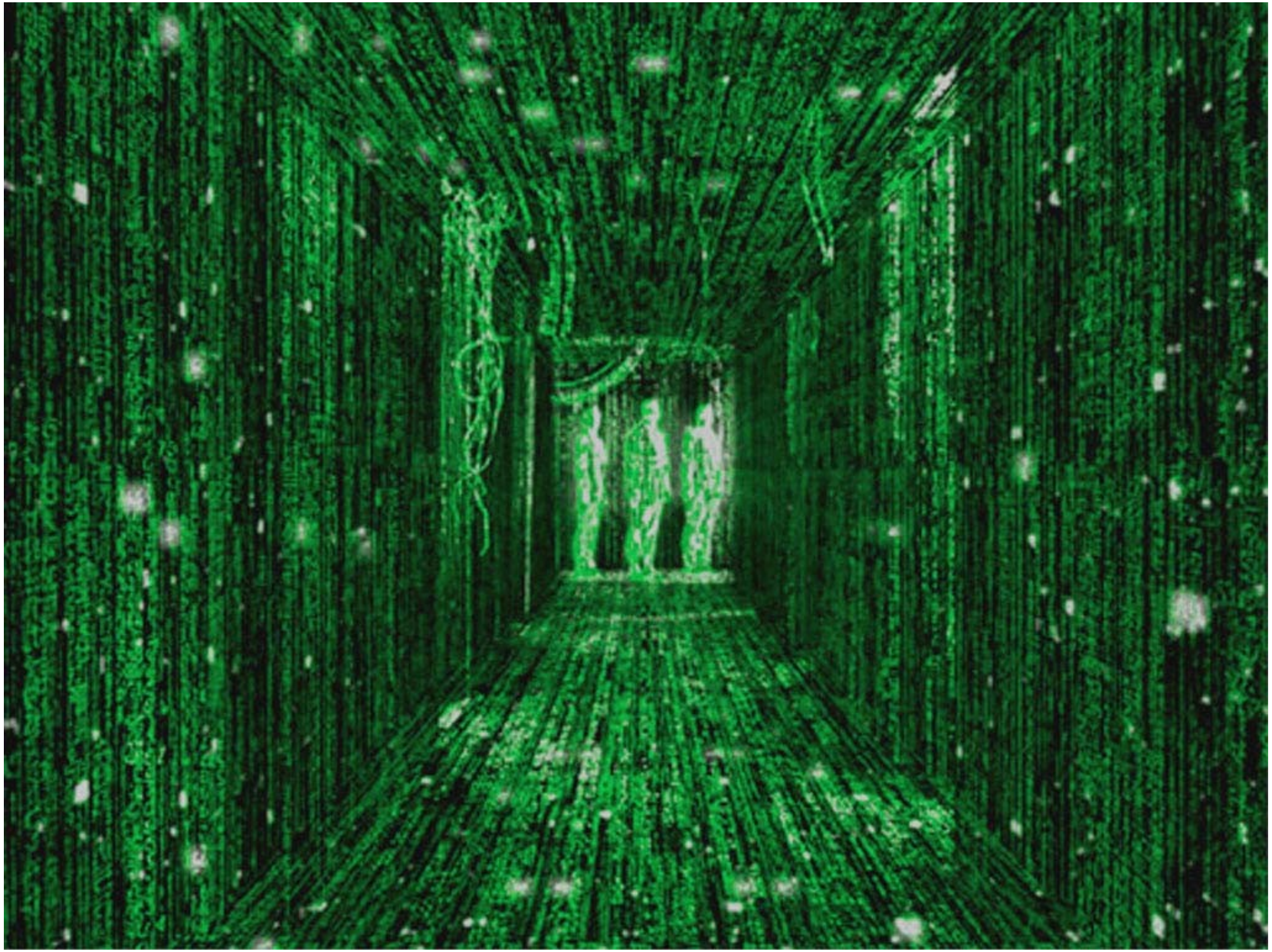
Empirical Analysis of Fault-Proneness in Methods by Focusing on Their Comment Lines

Hirohisa Aman, Sousuke Amasaki, Takashi Sasaki
and Minoru Kawahara

SWC Software Construction
RWTH AACHEN UNIVERSITY

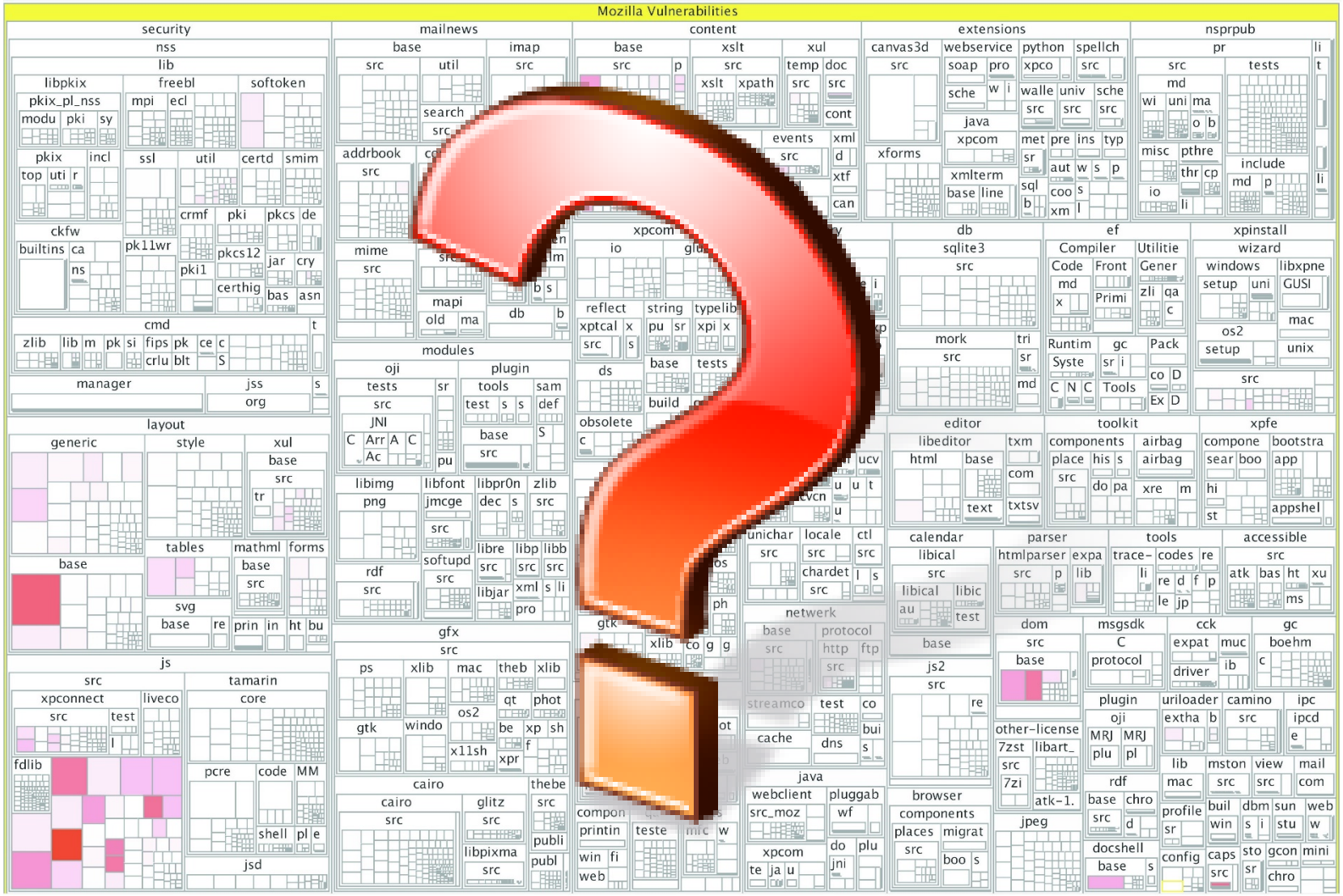
Introduction

Introduction



Introduction

Introduction



S. Neuhaus, T. Zimmermann, C. Holler, A. Zeller.
Predicting Vulnerable Software Components

Lines of comments / documentation (LOD, LOI)

```java
/**
 * This method calculates the number of risks for a given EOM and a given risk filter.
 */
public DataSeriesDTO<Double, String> getCurrentRiskCount(
        String eom, String metricName, RiskFilter riskFilter) {
    DataSeriesDTO<Double, String> result = new DataSeriesDTO<Double, String>(metricName);

    try {
        HashedMap<LocalDate, Integer> resultMap = new HashedMap<LocalDate, Integer>();

        // Get eom from persistence
        Eom eomObj = crudFacade.getEomByName(eom);

        // Iterate all the risks of the eom and count the number
        // of risks per history interval
        for (Risk risk : eomObj.getRisks()) {
        }

        // Sort List by date
        List<java.util.Map.Entry<LocalDate, Integer>> elements = new ArrayList(resultMap.entrySet());
        Collections.sort(elements, new Comparator<java.util.Map.Entry<LocalDate, Integer>>() {
        });

        // Convert HashMap to List
        for (java.util.Map.Entry<LocalDate, Integer> resultEntry : elements) {
            // Add new entry from hash set entry. During this
            // convert the LocalDate to string which become the
            // categories of the measurement values.
            result.add(new DataValueDTO<Double, String>(new Double(
                    resultEntry.getValue()), resultEntry.getKey().toString(
                    "MMM")));
        }
```

Lines of comments / documentation (LOD, LOI)

```java
/**
 * This method calculates the number of risks for a given EOM and a given risk filter.
 */
public DataSeriesDTO<Double, String> getCurrentRiskCount(
        String eom, String metricName, RiskFilter riskFilter) {
    DataSeriesDTO<Double, String> result = new DataSeriesDTO<Double, String>(metricName);

    try {
        HashedMap<LocalDate, Integer> resultMap = new HashedMap<LocalDate, Integer>();

        // Get eom from persistence
        Eom eomObj = crudFacade.getEomByName(eom);

        // Iterate all the risks of the eom and count the number
        // of risks per history interval
        for (Risk risk : eomObj.getRisks()) {
        }

        // Sort List by date
        List<java.util.Map.Entry<LocalDate, Integer>> elements = new ArrayList(resultMap.entrySet());
        Collections.sort(elements, new Comparator<java.util.Map.Entry<LocalDate, Integer>>() {
        });

        // Convert HashMap to List
        for (java.util.Map.Entry<LocalDate, Integer> resultEntry : elements) {
            // Add new entry from hash set entry. During this
            // convert the LocalDate to string which become the
            // categories of the measurement values.
            result.add(new DataValueDTO<Double, String>(new Double(
                    resultEntry.getValue()), resultEntry.getKey().toString(
                    "MMM")));
        }
```

Lines of comments / documentation (LOD, LOI)

```java
/**
 * This method calculates the number of risks for a given EOM and a given risk filter.
 */
public DataSeriesDTO<Double, String> getCurrentRiskCount(
        String eom, String metricName, RiskFilter riskFilter) {
    DataSeriesDTO<Double, String> result = new DataSeriesDTO<Double, String>(metricName);

    try {
        HashedMap<LocalDate, Integer> resultMap = new HashedMap<LocalDate, Integer>();

        // Get eom from persistence
        Eom eomObj = crudFacade.getEomByName(eom);

        // Iterate all the risks of the eom and count the number
        // of risks per history interval
        for (Risk risk : eomObj.getRisks()) {
        }

        // Sort List by date
        List<java.util.Map.Entry<LocalDate, Integer>> elements = new ArrayList(resultMap.entrySet());
        Collections.sort(elements, new Comparator<java.util.Map.Entry<LocalDate, Integer>>() {
        });

        // Convert HashMap to List
        for (java.util.Map.Entry<LocalDate, Integer> resultEntry : elements) {
            // Add new entry from hash set entry. During this
            // convert the LocalDate to string which become the
            // categories of the measurement values.
            result.add(new DataValueDTO<Double, String>(new Double(
                    resultEntry.getValue()), resultEntry.getKey().toString(
                    "MMM")));
        }
```

Research Questions

1) Do the fault-proneness of methods significantly differ by the commenting style?

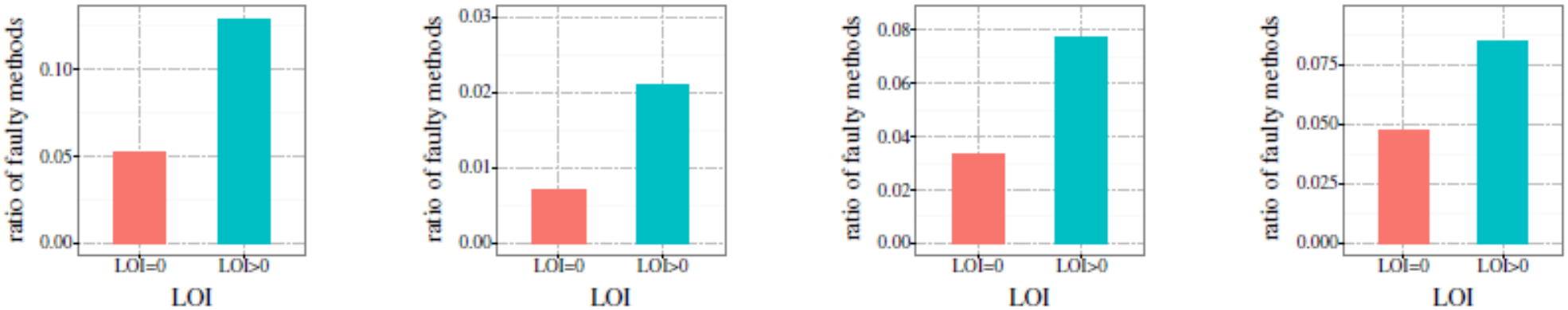
2) Does the amount of comments have any relationships on the fault-proneness?

Methodology

1. Investigate open source applications
2. Calculate LOI and LOD metrics
3. Investigate errors
   - Error = Keyword matching in commit log
4. Calculate correlation between the metrics and errors

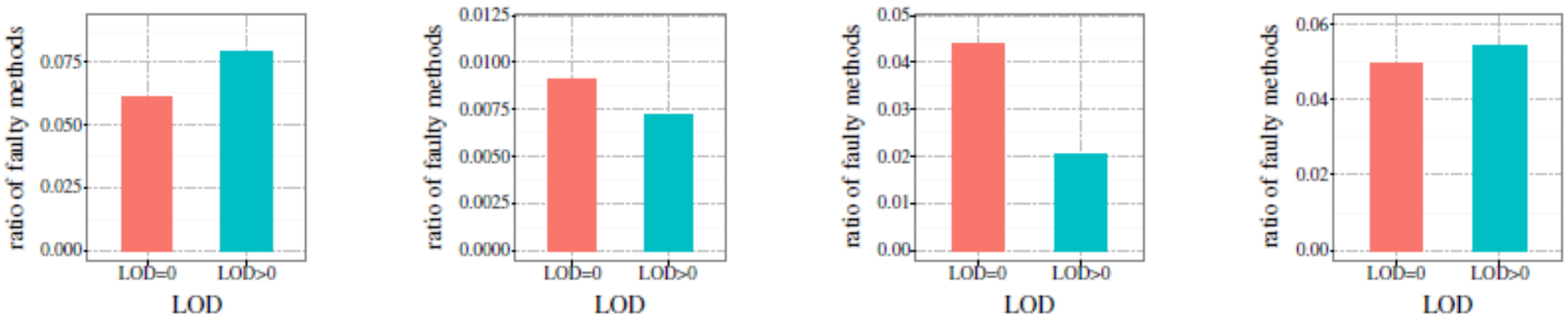
Field Study Results – ratio of faulty methods

- LOI = 0 vs. LOI > 1



(a) Eclipse CheckStyle (b) Hibernate (c) PMD (d) SQuirreL

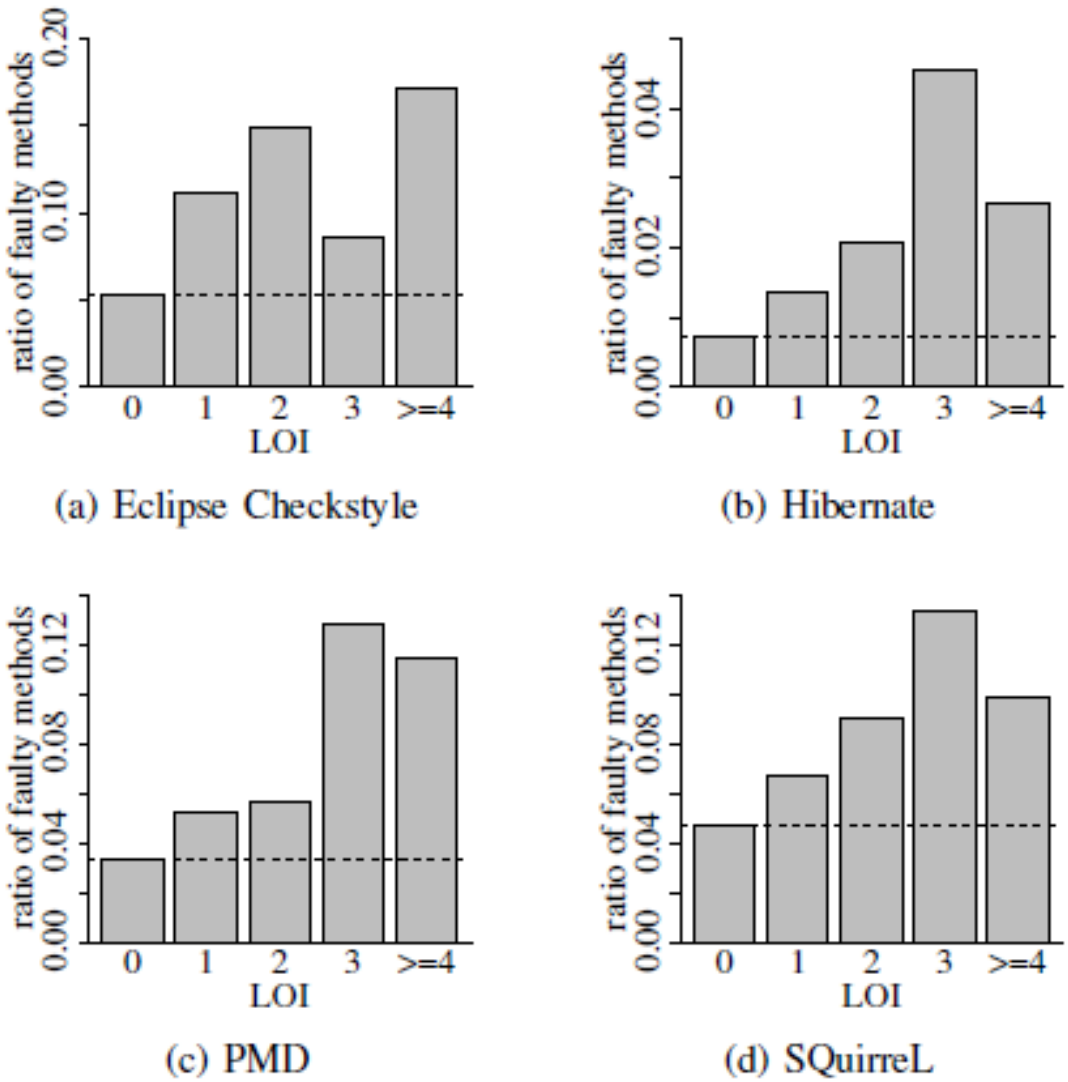
- LOD = 0 vs. LOD > 1



(a) Eclipse CheckStyle (b) Hibernate (c) PMD (d) SQuirreL

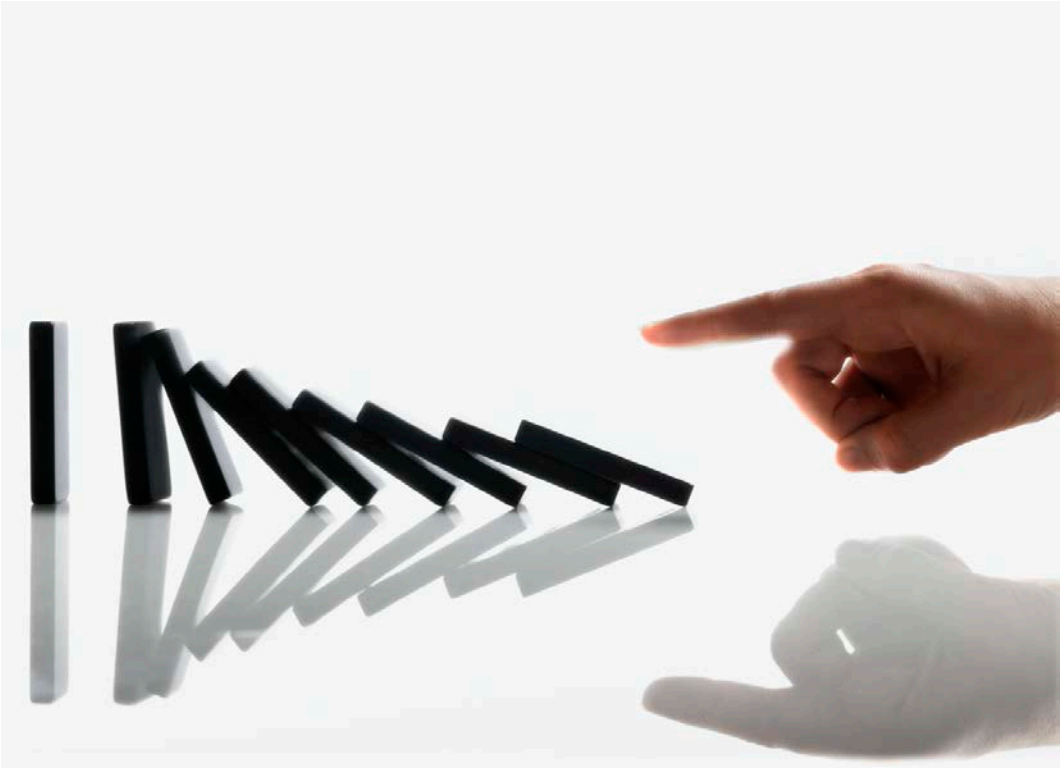
Field Study Results

- Ratio of faulty methods changes with the LOI



Discussion

- Result: Yes there is a correlation!



T. Holschuh, A. Zeller, et. Al.
Predicting Defects in SAP Java Code: An Experience Report

- How to interpret this finding?
  - How to use it (proactively)?

J. Sliwerski, T. Zimmermann, A. Zeller.
Don't Program on Fridays! How to Locate Fix-Inducing Changes