

An Evaluation of Two Bug Pattern Tools for Java

Stefan Wagner, Florian Deissenboeck, Michael Aichner
Technische Universität München

Markus Schwalb, Johann Wimmer
softlab GmbH

An Evaluation of Two Bug Pattern Tools for Java

Stefan Wagner, Florian Deissenboeck, Technische Universität München

Michael Aichner, beck et al. projects GmbH

Markus Schwalb, Johann Wimmer, softlab GmbH

An Evaluation of Two Bug Pattern Tools for Java

Stefan Wagner, Florian Deissenboeck, Technische Universität München

Michael Aichner, beck et al. projects GmbH

Johann Wimmer, softlab GmbH

Markus Schwalb, mobileX AG

An Evaluation of Two Bug Pattern Tools for Java

Stefan Wagner, Florian Deissenboeck, Technische Universität München

Michael Aichner, beck et al. projects GmbH

Johann Wimmer, Cirquent GmbH

Markus Schwalb, mobileX AG

Bug Pattern Analysis – Hope

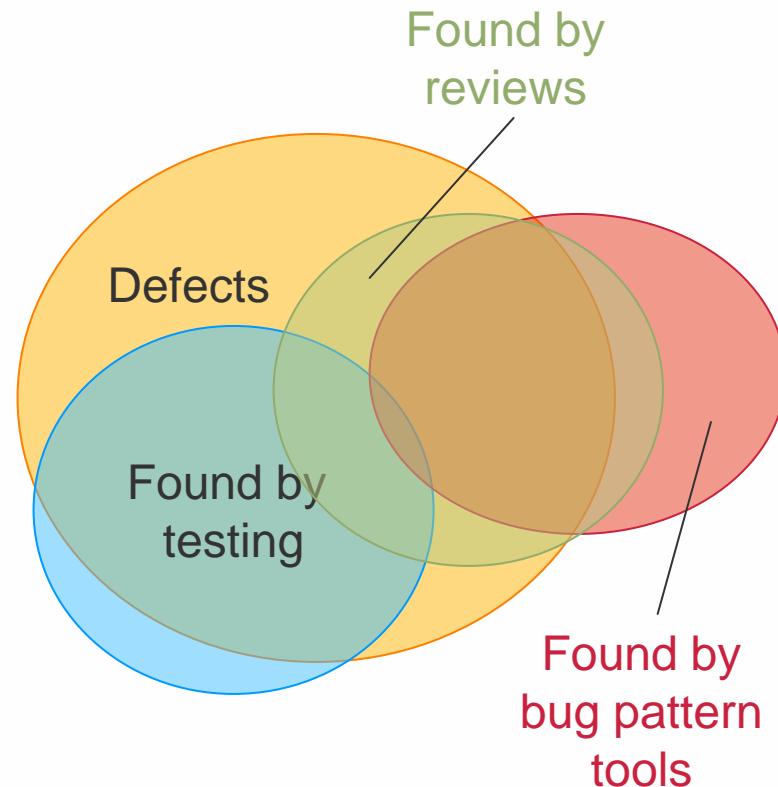
- Mistakes made in coding are often similar
- Many are very hard to find with testing
- Often expensive to detect with inspections
- Bug patterns can capture these mistakes
- Automatisation!
- Quick and cheap defect detection

Split Cleaner

```
Connection con;
String tableName;
String queryString =
    ("SELECT * FROM " +
     tableName);
Statement stmt =
    con.createStatement();
ResultSet rs =
    stmt.executeQuery(queryString
    );
while (rs.next())
    execute(rs);
```

Bug Pattern Analysis – Reality

- Different tools
 - Different programming languages supported
 - Different detectors
 - Only partly overlapping
- Usage scenarios
 - Quality check -> Dashboard
 - Preinspection defect detection
 - „Extended compiler“ -> included in daily build
- False positives
 - Warnings of issues that are not a defect
 - Can be up to 90%
- Found defects / found negatives
 - Can reveal many defects
 - Depends sometimes on programming style
 - Overlaps with defects found in reviews but rarely with defects found in testing



Problem Statement

- Preliminary results suggest that most identified defects are related to maintainability
- Are we really able to find potential field defects early?
- Can we handle the high false positive rates?
- Under what conditions is the use of the tools economically reasonable?

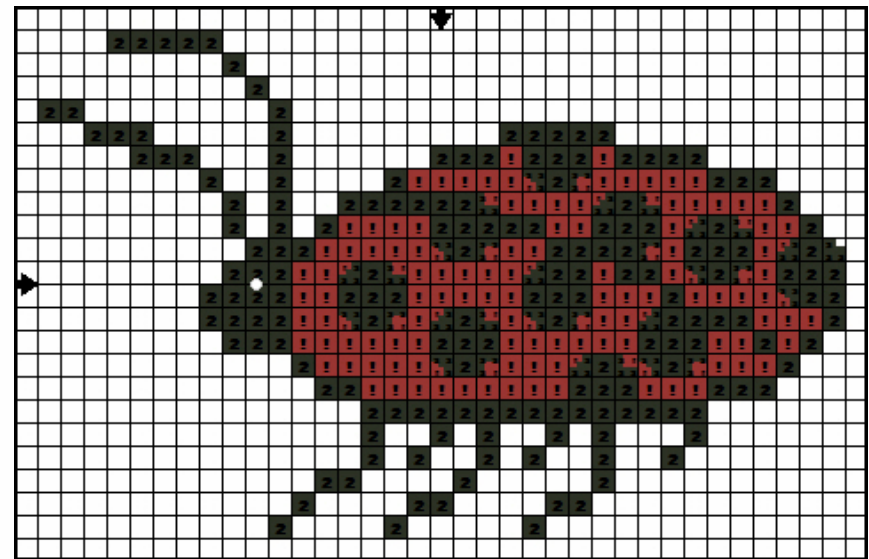


Outline

- Bug Pattern
- Research Questions
- Case Study
- Consequences
- Conclusions

Bug Pattern

- Common pitfalls of a programming language
- Example: Split cleaner
- Similar to anti-pattern, code smells
- Tools
 - FindBugs
 - PMD
 - Klocwork
 - PCLint
 - FxCop

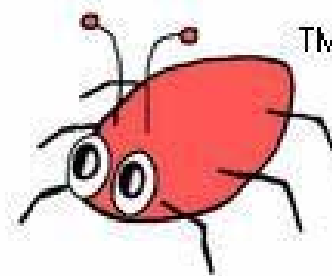


Research Questions

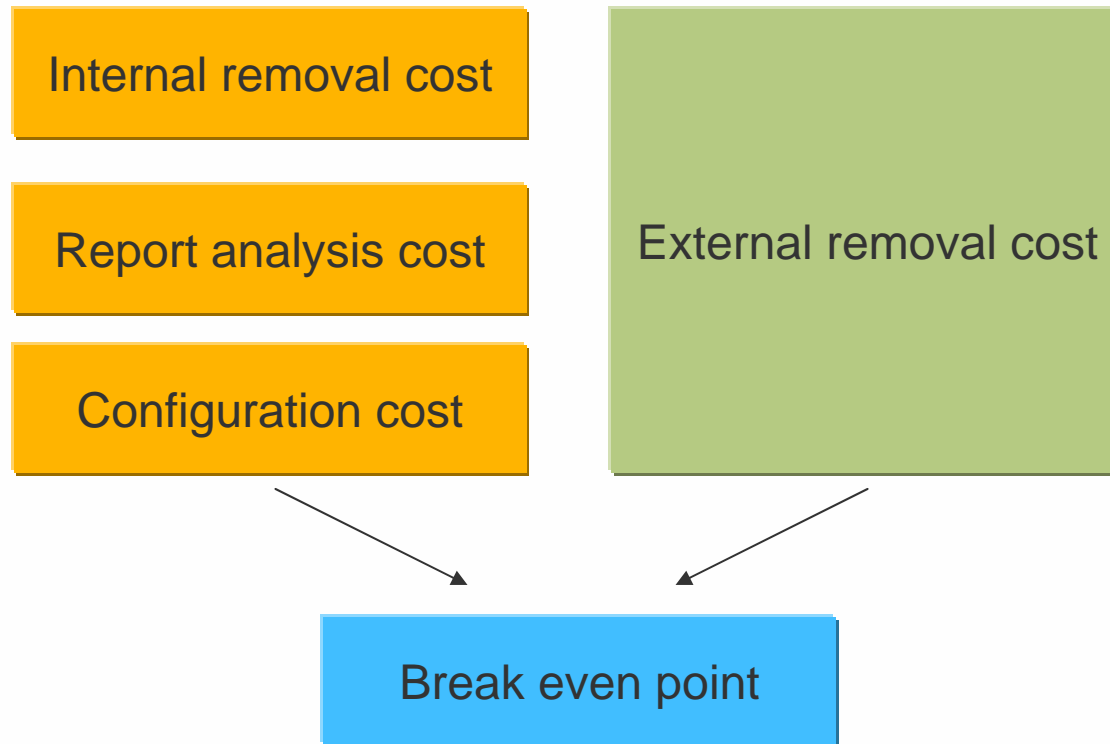
- Field defect
 - Defects after delivery of a version
 - Documented in the defect management system
- RQ 1: How many field defects need to be detected by the tools to be cost-efficient?
- RQ 2: How good are bug pattern tools in detecting field defects?

Case Study

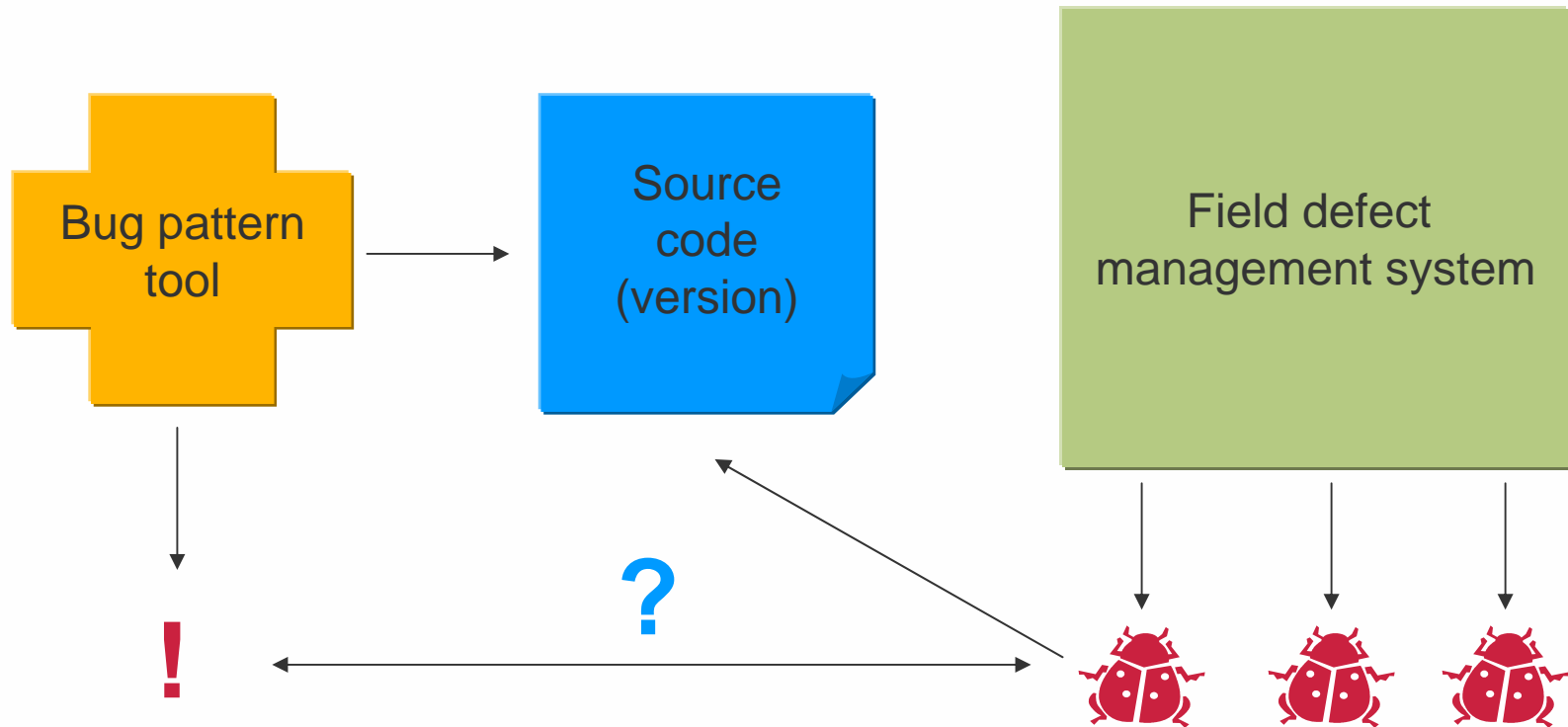
- Cirquent / softlab
- Two information systems
- Web-based
- Java (J2EE)
- FindBugs, PMD(, ConQAT)
- Standard and individual configuration
- Individual configuration
 - Iteratively developed
 - To reduce false positives
 - Excludes anomalies related to readability



Study Design: Cost-Efficiency



Study Design: Effectiveness



Threats to Validity

- Internal Validity
 - After the fact analysis -> no direct influence
 - Only a subset of failures analysed
 - Subset seems representative
- External Validity
 - Both projects at same company
 - Only two different tools (but widely used)
 - Only for Java
 - Other static analysis tools?

Cost Efficiency

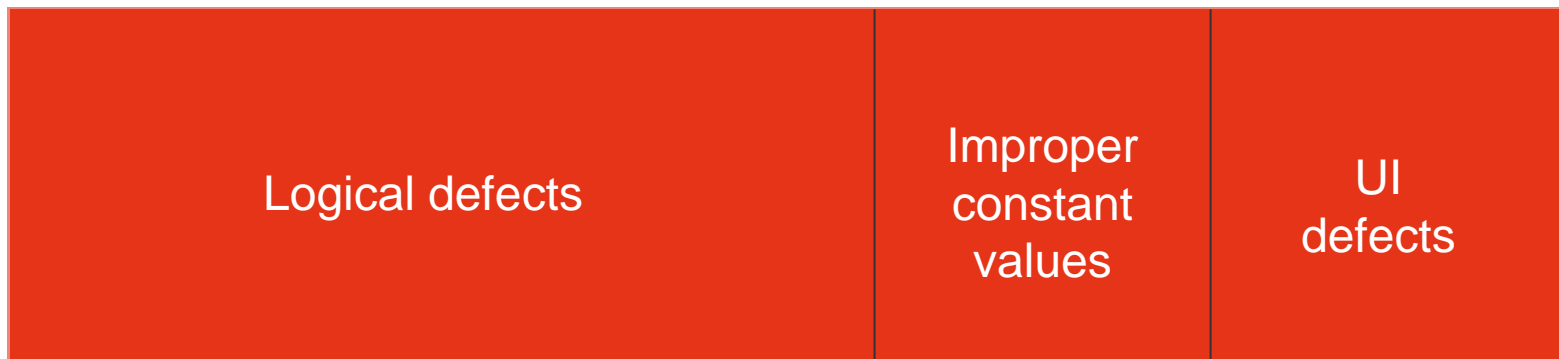
Monetary values in €

Cost Type	Project X		Project Y	
	StdConfig	IndConfig	StdConfig	IndConfig
Configuration Costs	0	420	0	420
Report analysis costs	2,240	178	402	141
Internal removal costs	1,120	155	201	122
Saved costs (per defect)	228	228	228	228
Break even (average)	14.7	3.3	2.7	3.0
Saved costs (severe defect)	2,388	2,388	2,388	2,388
Break even (severe defect)	1.4	0.3	0.3	0.3

Detection of Field Defects

- 615 reported defects
- Randomly selected 99 (16.1%)
- 27 discarded because fault location could not be determined

72 analysed defects



**None of the analysed field defects was found
by the bug pattern tools!**

Consequences and Lessons Learned

- Results
 - Tools can be economically reasonable
 - Need to detect a small number of potential field defects
 - Detected defects are more of a different type
- Learned along the way
 - Use the tools often and early in the development to keep the number of warnings small
 - Use different tools in combination
 - Create an individual configuration for each tool
 - Integrate the warnings into your IDE
 - Accumulate and analyse the results from the tools into a report

Conclusions

- Bug pattern tools not well-equipped to prevent field defects
- Only small number of detected (potential) field defects necessary
- Concentration on reliability and Java system
- Additionally
 - Comparison with other defect-detection: partially the same defects
 - Fault-proneness: number of tools that warn correlates with defect proneness
- Further questions
 - Different for quality attributes such as security?
 - What are the limits of static analysis?
 - When do specific static analysis make sense?
 - „Generic“ generation of tests for bugs that cannot be detected statically?
 - Possible automatic correction of bug patterns?

Effectiveness (different versions)

Versions		Fault	Change	Tool	Unknown	Σ
a	b	0	11	9	2	22
b	c	0	22	3	2	27
c	d	0	8	4	0	12
d	e	0	5	1	0	6
Σ		0	46	17	4	67
%		0.0	68.6	25.4	6.0	100.0