SOFTWARE TESTING IN INDUSTRY

BACHELOR THESIS – MARKUS EGGIMANN

2ND PRESENTATION

UNIVERSITÄT BERN

SWISS POST

CONTENT

- Motivation
- Research questions
- Case study
- Research methods
- Results
- Conclusions
- Discussion

MOTIVATION


Testing?
Find Bugs!
Fast Deployment!
Quality!
Short Release Cycles!

MOTIVATION

RESEARCH QUESTIONS

RQ1: “Does the discovery of bugs push the writing of tests?”

RQ2: “Do existing tests prevent the occurrence of bugs?”

RQ3: “Is the system’s architecture designed in a way, that addition of new tests is easy?”

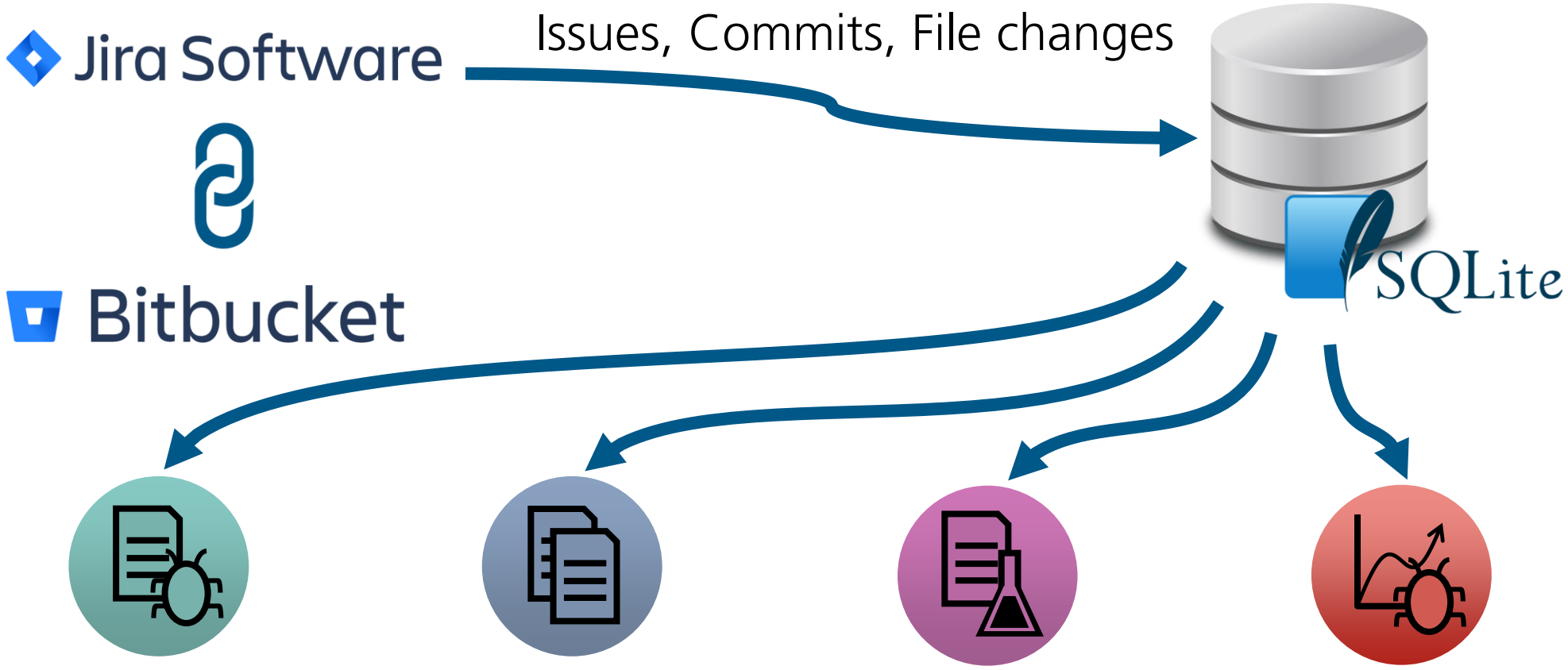
CASE STUDY

Project ePostOffice

- Digital platform for receiving and sending physical and electronic mail
- Mixed technologies
- Ongoing for 6 years

- Confluence (Wiki, Requirements)
- Jira (Issue-Tracker, SCRUM)
- Git (Version control)

RESEARCH METHODS 1/3



RESEARCH METHODS 2/3

Defective classes not covered by Tests

- Why are they not covered by tests? => Assess testability of code
- How could we test them? => Suggest improvements

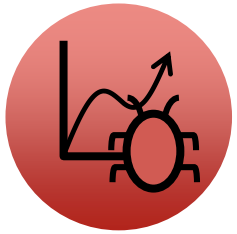
Possibly recurring Bugs

- Which ones are really recurring bugs? => Manual inspection
- Why did they occur again? => Study history

RESEARCH METHODS 3/3

Defective classes covered by Tests

- How good are those tests? => Apply mutation testing to them



Bug statistics

- Any anomalies in development history? => Ask developers
- Proneness to bugs of different components? => Ask developers

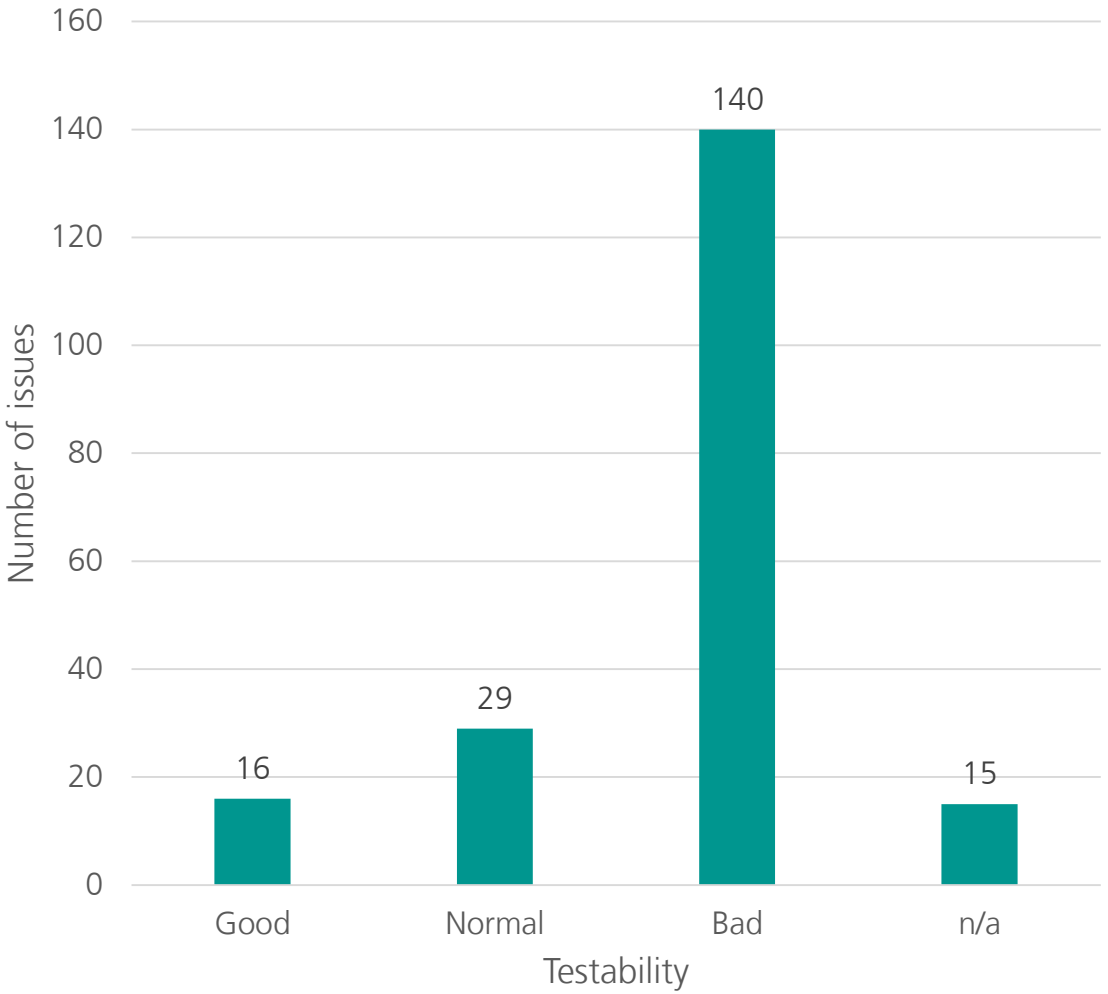
RESULTS 1/5
TESTABILITY OF CODE AFFECTED BY BUGS

Method

- Manual inspection of 200 issues/bugfixes
- Classification of Testability

Results

- Testability is bad in most cases
- Reasons:
  - Violation of Single Responsibility Principle
  - Too many dependencies



RESULTS 2/5
RECURRING BUGS

Method

- Tool to compare diffs on method level
- List of commits which changed same method
- Manual inspection of suggestions to find real recurring bugs

Results

- C# files: 20 possibly recurring bugs, no really recurring bugs
- Java files: 135 possibly recurring bugs, 7 similar bugs
- No tests added after first and also not after second bug

RESULTS 3/5
MUTATION TESTING

Method

- Apply mutation testing to bugs which are possibly covered by tests
- Mutate focal methods under test

Results

- Newer components achieve higher scores
- If only main control flow is covered by tests, many mutations are not detected

Component	# Classes	# Mut	# Kill	Score
DelOrd	2	11	9	81.8%
GKPortal	2	24	21	87.5%
PkPortal	3	66	44	66.7%
RecPref	2	31	26	83.9%
Transfer	1	2	2	100.0%
Total	10	134	102	76.1%

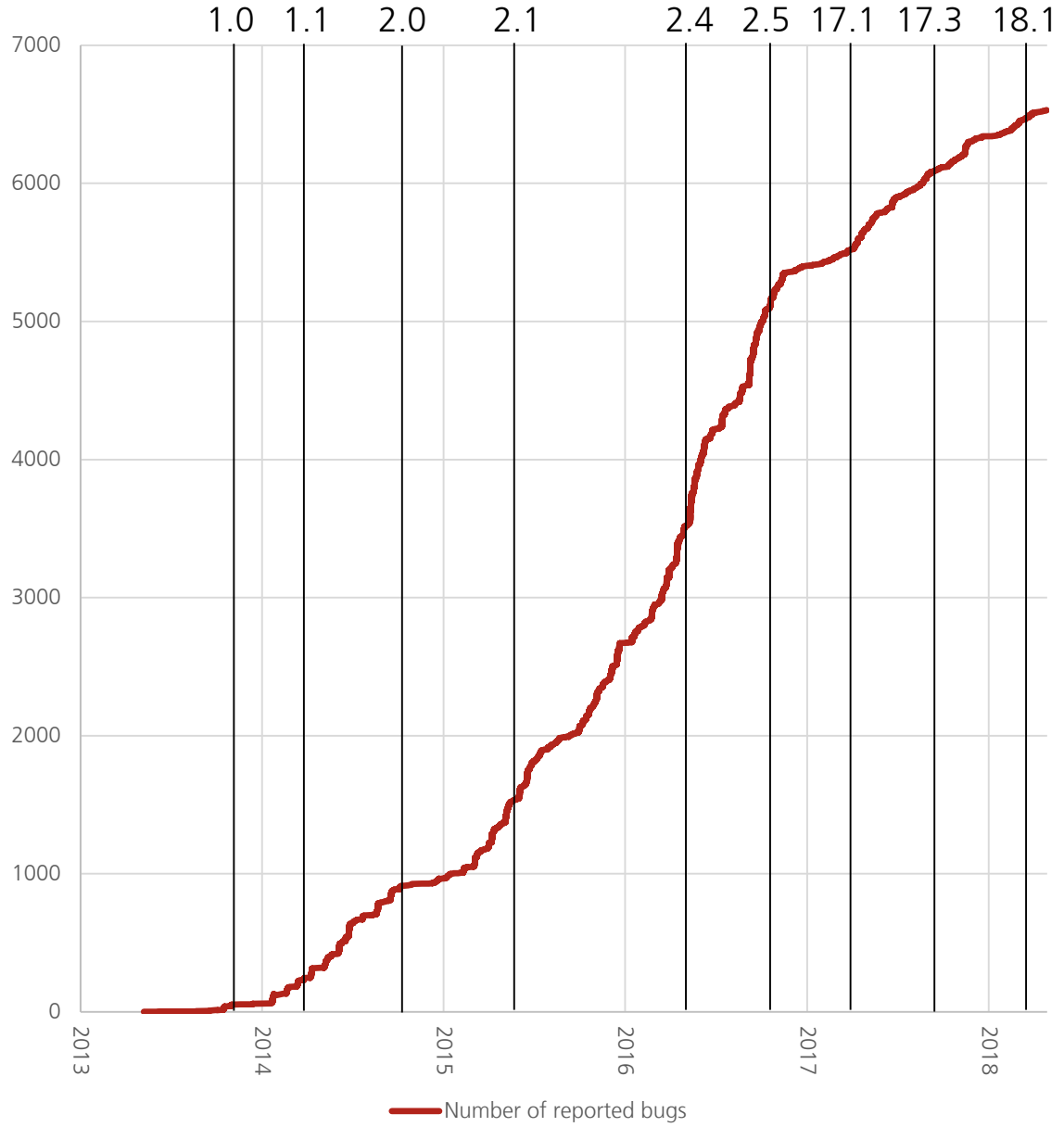
RESULTS 4/5
BUG REPORT HISTORY

Method

- Retrieve bugs from database
- Plot and inspect timeline
- Approach developers

Results

- Ascents right before and right after releases
- Clear “bend” in Nov 2016



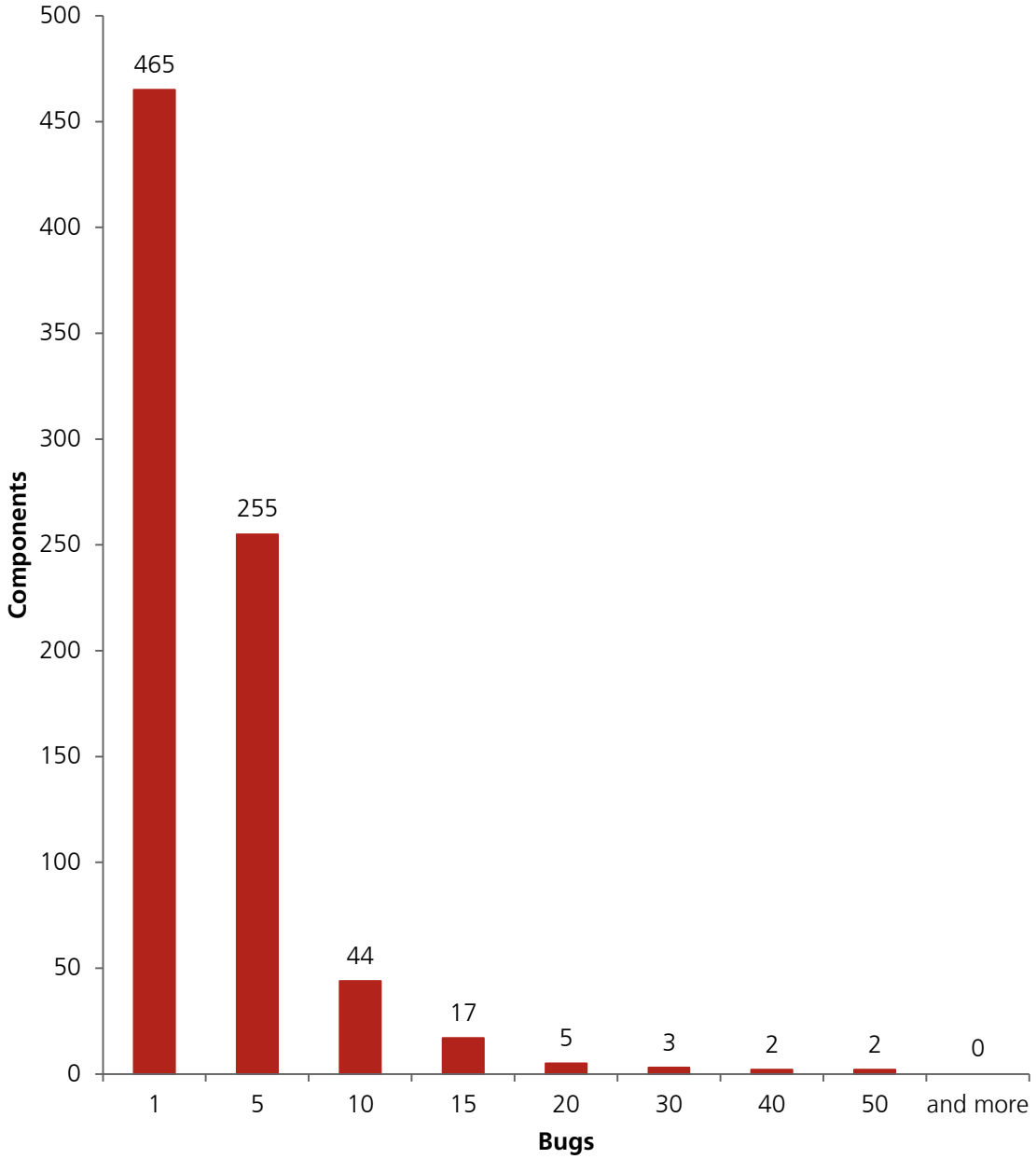
RESULTS 5/5
PRONENESS TO BUGS

Method

- Use query to get number of bugs per component
- Approach developers for “worst 15” components

Results

- 58.6% of buggy components are only affected by one bug
- Buggy components are usually large and complex



CONCLUSIONS

RQ1: “Does the discovery of bugs push the writing of tests?”

RQ2: “Do existing tests prevent the occurrence of bugs?”

RQ3: “Is the system’s architecture designed in a way, that addition of new tests is easy?”

«It is all about testability!»

DISCUSSION