Detecting Problems in the Database Access Code of Large Scale Systems

An industrial Experience Report

Existing static analysis tools focus on language-related problems

Coverity

PMD

Google error-prone

FindBugs

Facebook Infer

However, many problems are related to how developers use different frameworks

Over 67% of Java developers use Object-Relational Mapping (Hibernate) to access databases

HIBERNATE

$\boxed{67\%}$

Java JDBC

$\boxed{22\%}$

Existing static analysis tools provide mostly rudimentary support for JDBC!

Over 40% of Java web application developers use Spring

Developers use Spring to manage database transactions in web applications

None of the static analysis tools support Spring!

There is a huge need for framework-specific tools

Developers leverage MANY frameworks, but existing tools only support detecting language-related problems.

An example class with Java ORM code

User class is mapped to *"user"* table in DB

Performance-related configs

id is mapped to the column *"id"* in the user table

A user can belong to multiple teams

Eagerly retrieve associated teams when retrieving a user object

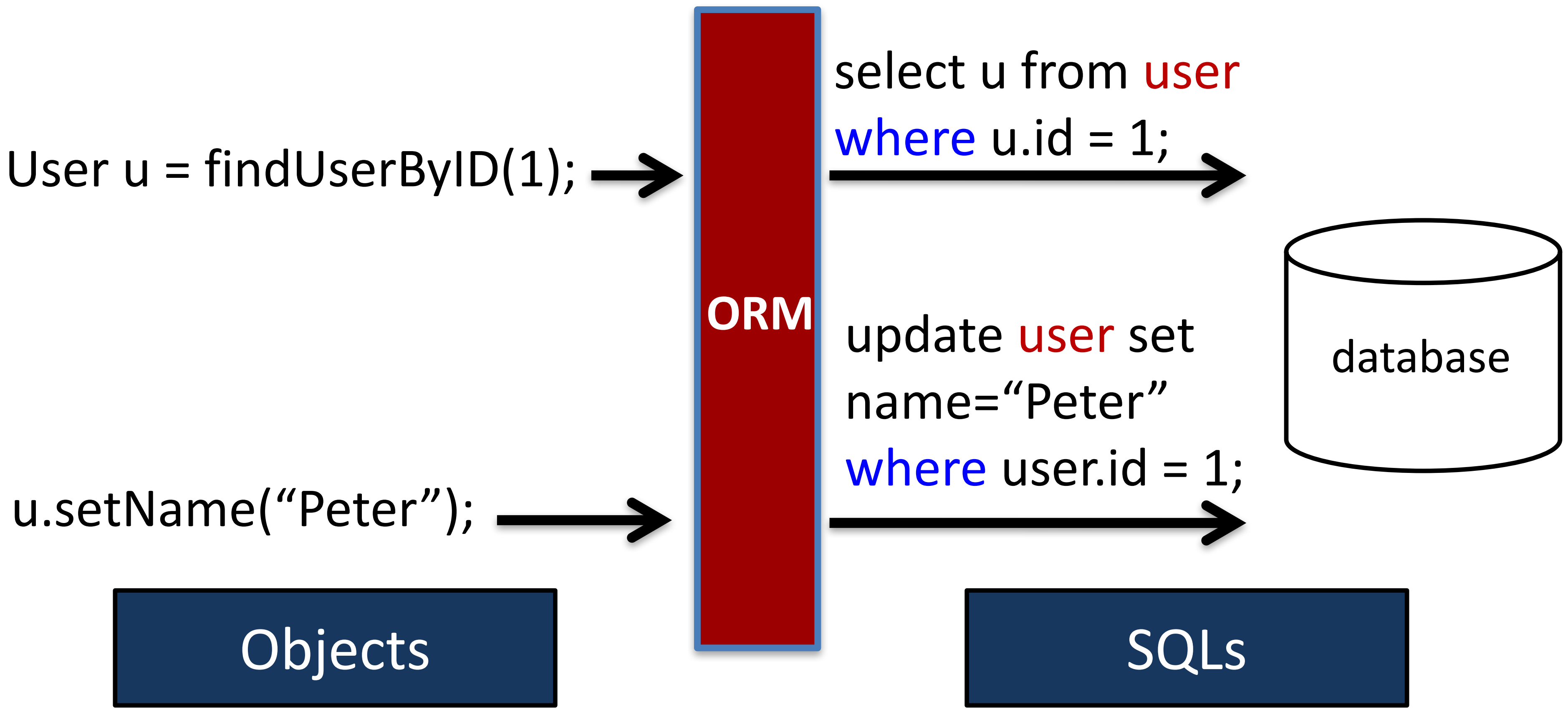
User.java

```
@Entity
@Table(name = "user")
@DynamicUpdate
public class User{
    @Column(name="id")
    private int id;

    @Column(name="name")
    String userName;

@OneToMany(fetch=FetchType.EAGER)
    List<Team> teams;
    public void setName(String n){
        userName = n;
    }
... other getter and setter methods
```

Accessing the database using ORM



User u = findUserByID(1);
u.setName("Peter");
ORM
select u from user where u.id = 1;
update user set name="Peter" where user.id = 1;
database
Objects
SQLs

Transaction management using Spring

```
@Transaction(Propogation.REQUIRED)
getUser(){
    …
    updateUserGroup(u)
    …
}
```

Create a DB transaction

Entire business logic will be executed with the same DB transaction

By using ORM and Spring, developers can focus more on the business logic and functionality

Implementing DBChecker

- ***DBChecker*** looks for both *functional* and *performance* bug patterns

- ***DBChecker*** is integrated in industrial practice

Overview of the presentation

Bug patterns

Lessons learned when adopting the tool in practice

Overview of the presentation

```
import org.webmacro.util.
import org.webmacro.*;
import java.io.*;
import java.sql.*;
import java.util.*;
import javax.servlet.
```

Bug patterns

Lessons learned when adopting the tool in practice

More patterns and learned lessons in the paper

ORM excessive data bug pattern

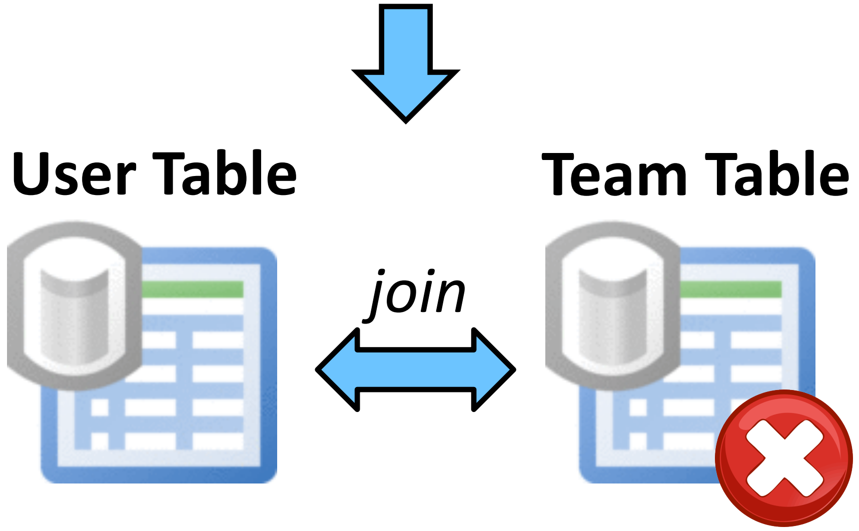
```
Class User{
        @EAGER
        List<Team> teams;
}
```

Eagerly retrieve teams from DB

Objects

```
User u = findUserById(1);
u.getName();
EOF
```

SQL



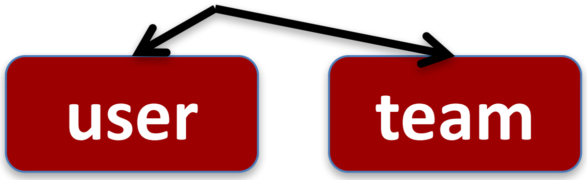
Team data is never used!

Detecting excessive data using static analysis

```
Class User{
        @EAGER
        List<Team> teams;
}
```

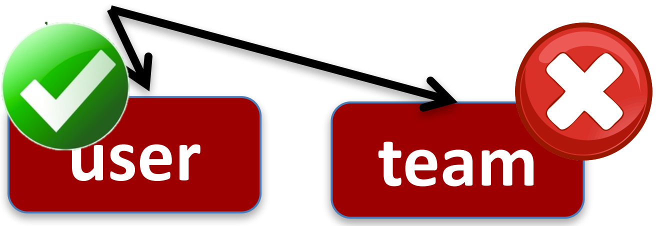
First find all the objects that eagerly retrieve data from DB

```
User user = findUserByID(1);
```



Identify all the data usages of ORM-managed objects

```
user.getName();
```



Check if the eagerly retrieved data is ever used

Nested transaction bug pattern

```
@Transaction(Propogation.
REQUIRED)
getUser(){
    updateUserGroup(u)
    …
}
```

$\leftarrow$ Create a DB transaction

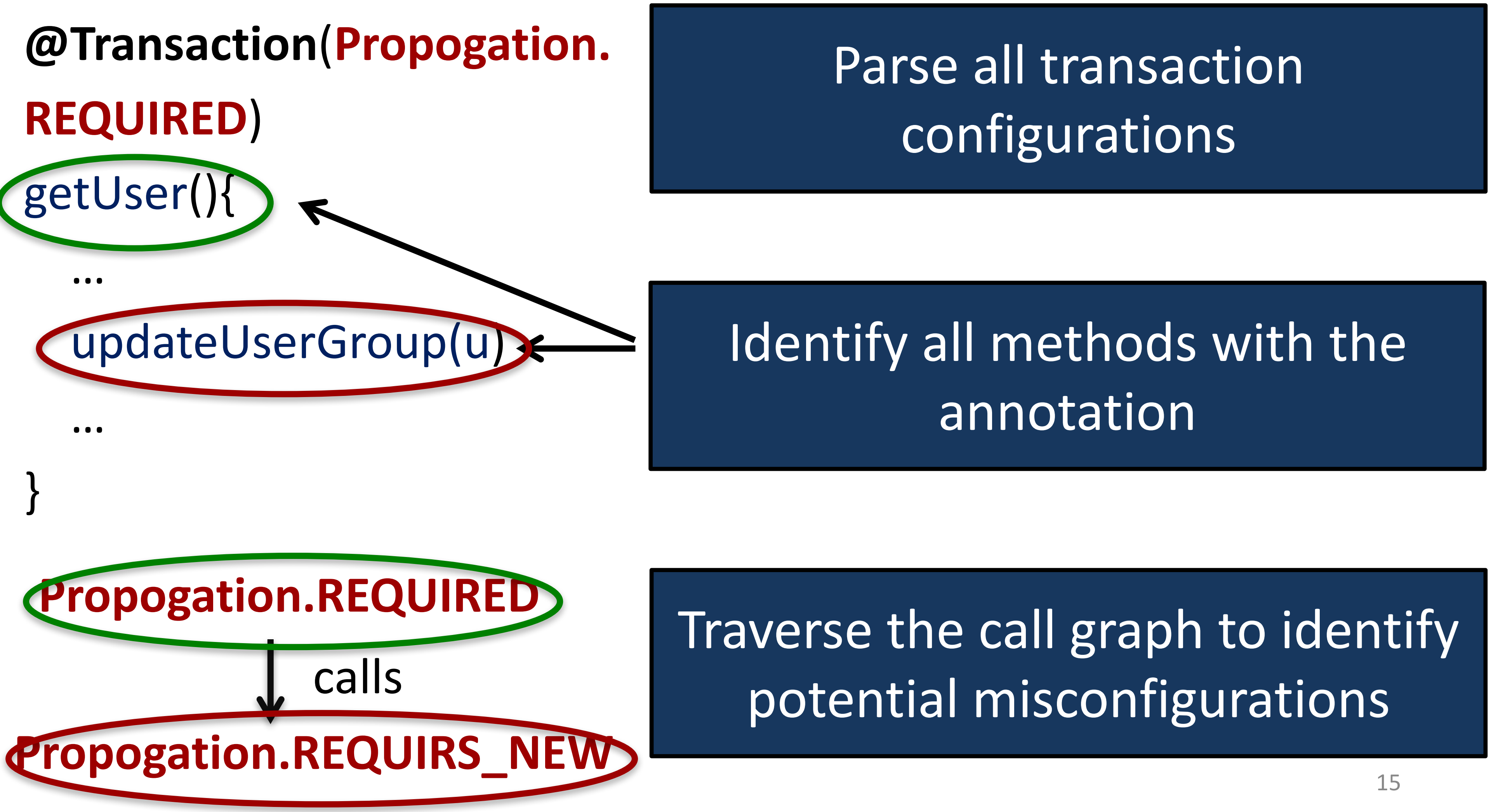
$\leftarrow$

```
@Transaction(Propogation.
REQUIRES_NEW)
```

Create a child transaction, and suspend parent transaction until child is finished

Misconfigurations can cause unexpected transaction timeout, deadlock, or other performance-related problems

Detecting nested transaction bug pattern



```
@Transaction(Propogation.
REQUIRED)
getUser(){
    …
  updateUserGroup(u)
    …
}
```

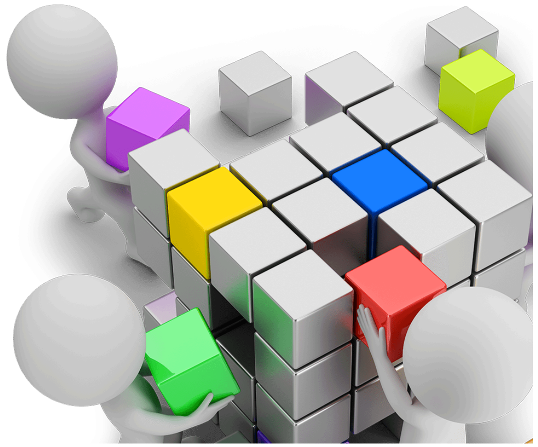
$$\begin{tikzcd}
\text{Propogation.REQUIRED} \arrow[d, "\text{calls}"] \\
\text{Propogation.REQUIRS\_NEW}
\end{tikzcd}$$

Parse all transaction configurations

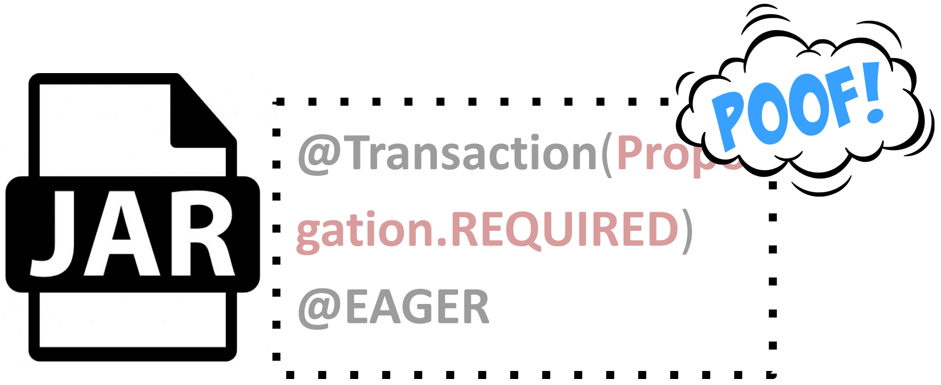
Identify all methods with the annotation

Traverse the call graph to identify potential misconfigurations

Limitation of current static analysis tools



Do not consider how developers configure frameworks



Annotations are lost when converting source code to byte code

Many problems are related to framework configurations

Many configurations are set through annotations

Overview of the presentation

Bug patterns

Lessons learned when adopting the tool in practice

Most discussed bug patterns are related to incorrect usage of frameworks

Overview of the presentation

```
import org.webmacro.se
import org.webmacro.util.
import java.io.*;
import java.sql.*;
import java.util.*;
import javax.servlet.
```

Bug patterns

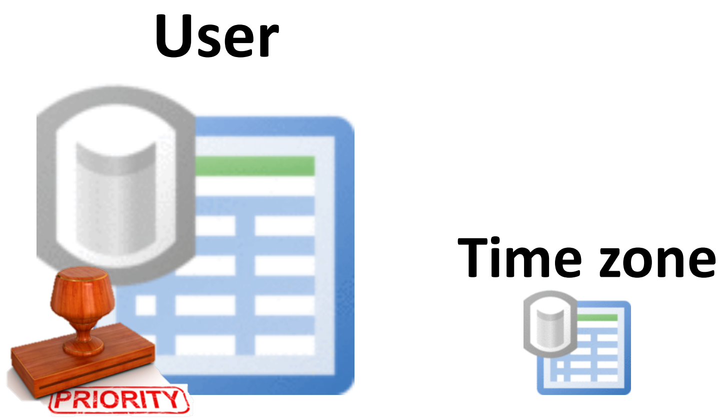
Lessons learned when adopting the tool in practice

Most discussed bug patterns are related to incorrect usage of frameworks

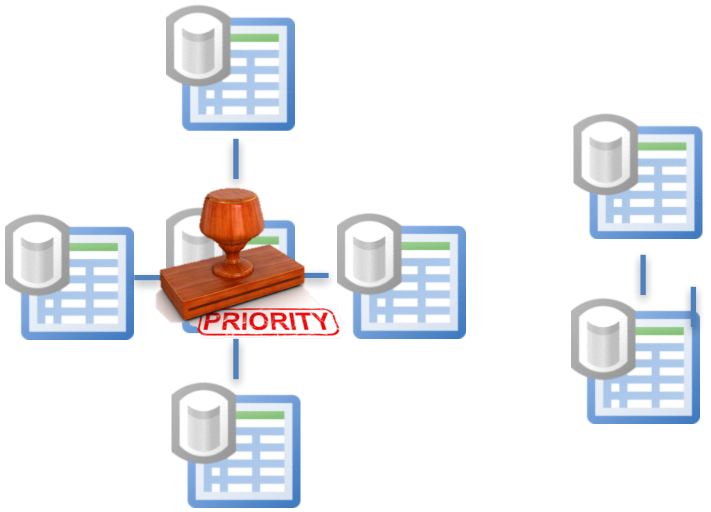
Handling a large number of detection results

- Developers have ***limited time*** to fix detected problems
- Most existing static analysis frameworks do not prioritize the detected instances for ***the same bug pattern***

Prioritizing based on DB tables



- Problems related to *large* or *frequently-accessed* tables are ranked higher (more likely to be performance bottlenecks)



- Problems related to highly dependable tables are ranked higher

Developers have different backgrounds

- Not all developers are familiar with these frameworks and databases
- Developers may not take the problems seriously if they don’t understand the impact

Educating developers about the detected problems

- We hosted several workshops to educate developers about the impact and cause of the problems
- Walk developers through examples of detected problems
- May learn new bug patterns from developers

Overview of the presentation

```
import org.webmacro.se
import org.webmacro.util.
import java.io.*;
import java.sql.*;
import java.util.*;
import javax.servlet.
```

Bug patterns

Lessons learned when adopting the tool in practice

Most discussed bug patterns are related to incorrect usage of frameworks

We prioritize problems based on DB tables, and educate developers about the problems

Existing static analysis tools focus on language-specific problems

Coverity

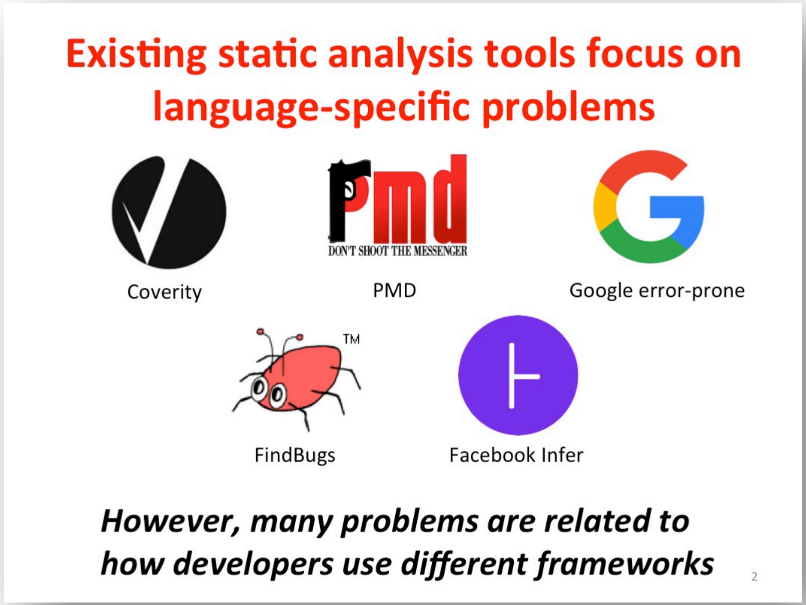
PMD

Google error-prone

FindBugs

Facebook Infer

However, many problems are related to how developers use different frameworks



Existing static analysis tools focus on language-specific problems

Coverity

PMD

Google error-prone

FindBugs

Facebook Infer

However, many problems are related to how developers use different frameworks

2

Over 67% of Java developers use Object-Relational Mapping (Hibernate) to access databases

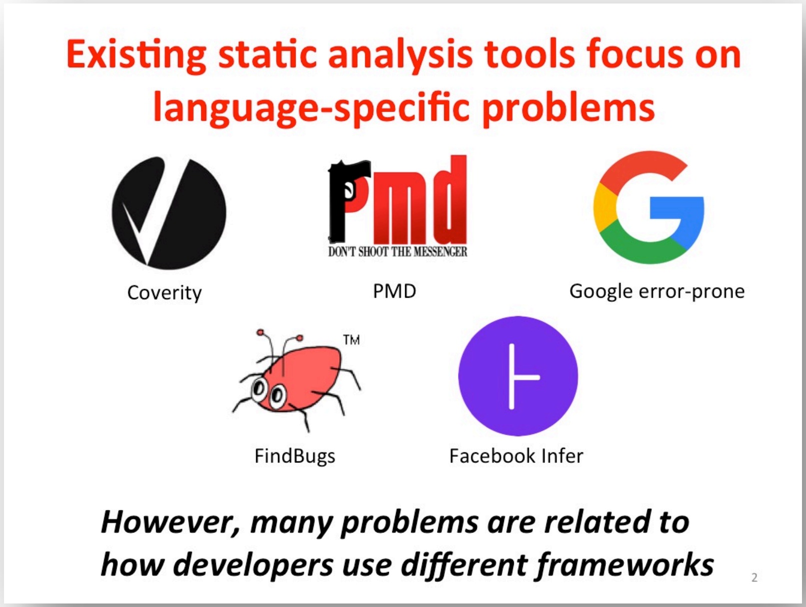
HIBERNATE

67%

Java JDBC

22%

Existing static analysis tools provide mostly rudimentary support for JDBC!

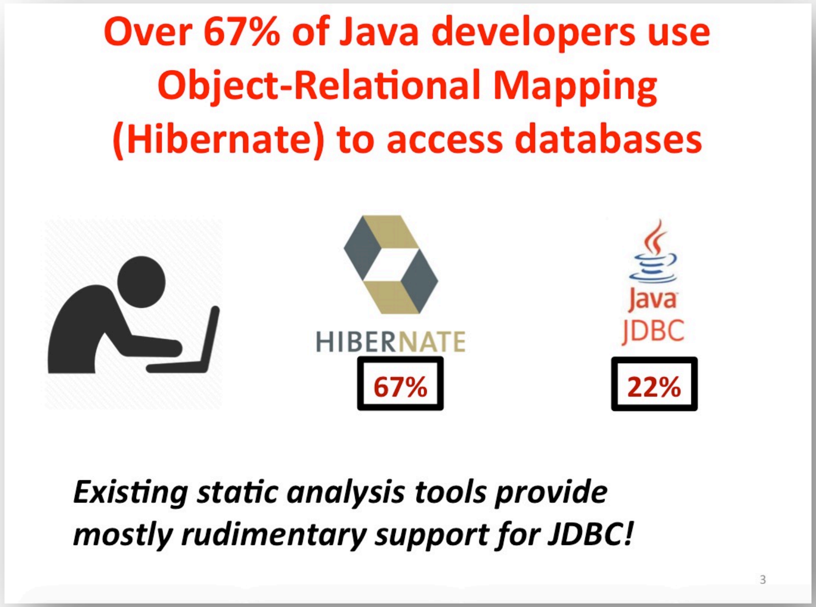


Existing static analysis tools focus on language-specific problems

Coverity
PMD
Google error-prone
FindBugs
Facebook Infer

However, many problems are related to how developers use different frameworks

2



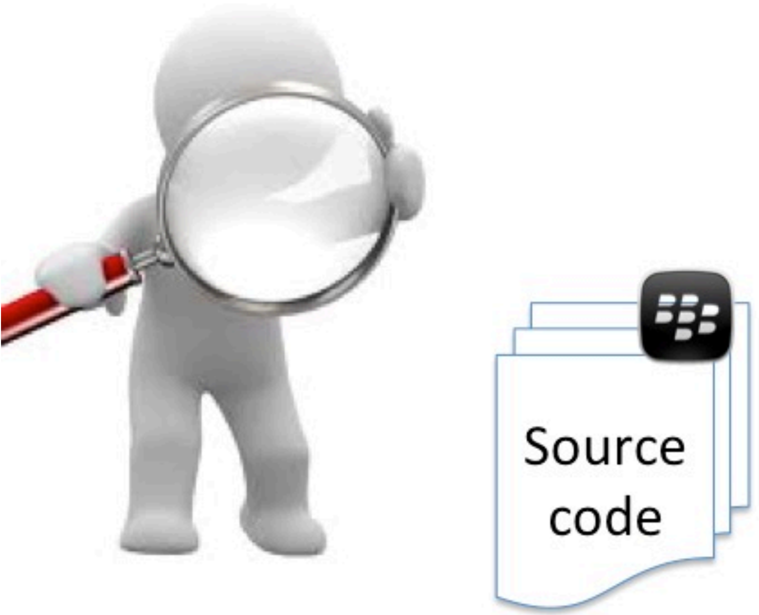
Over 67% of Java developers use Object-Relational Mapping (Hibernate) to access databases

HIBERNATE
67%
Java
JDBC
22%

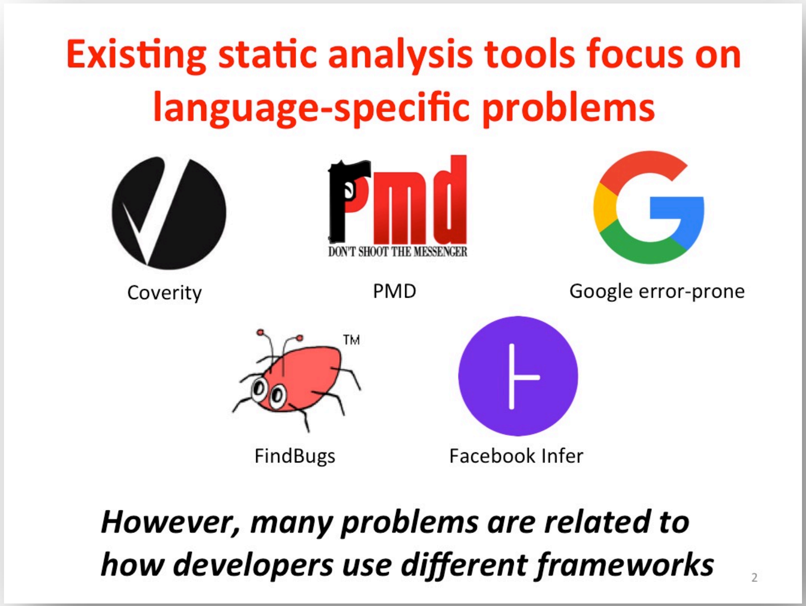
Existing static analysis tools provide mostly rudimentary support for JDBC!

3

Implementing DBChecker



- ***DBChecker*** looks for both *functional* and *performance* bug patterns
- ***DBChecker*** is integrated in industrial practice

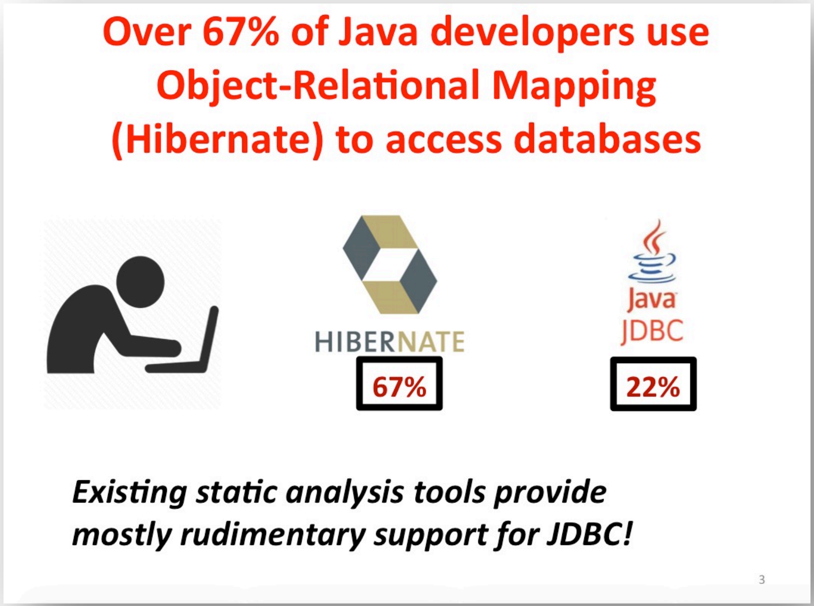


Existing static analysis tools focus on language-specific problems

Coverity
DON'T SHOOT THE MESSENGER
PMD
Google error-prone
FindBugs
Facebook Infer

However, many problems are related to how developers use different frameworks

2

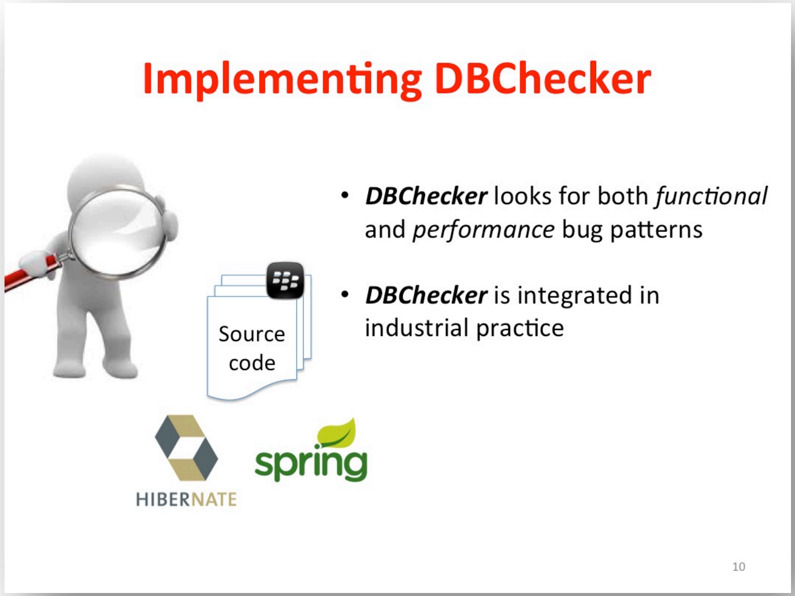


Over 67% of Java developers use Object-Relational Mapping (Hibernate) to access databases

HIBERNATE
67%
Java
JDBC
22%

Existing static analysis tools provide mostly rudimentary support for JDBC!

3



Implementing DBChecker

Source code
HIBERNATE
spring

- ***DBChecker*** looks for both *functional* and *performance* bug patterns
- ***DBChecker*** is integrated in industrial practice

10

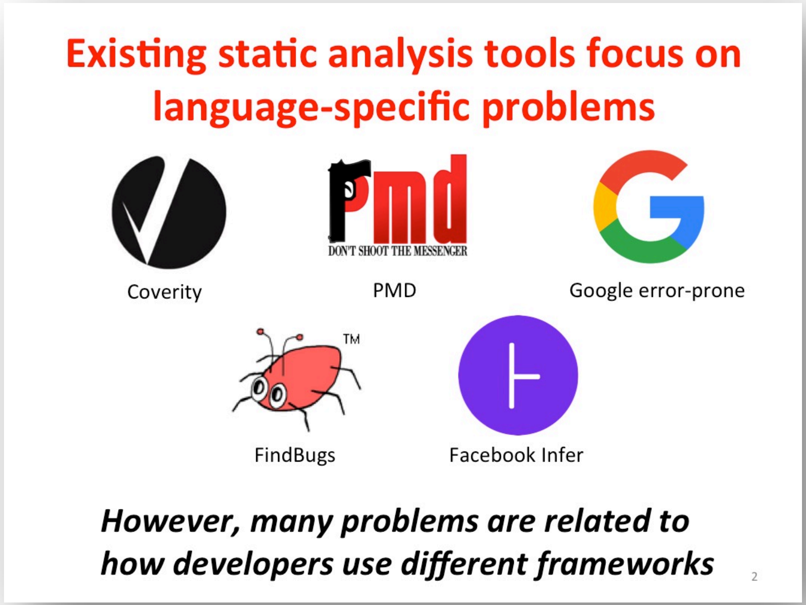
Overview of the presentation

Bug patterns

Lessons learned when adopting the tool in practice

Most discussed bug patterns are related to incorrect usage of frameworks

We prioritize problems based on DB tables, and educate developers about the problems



Existing static analysis tools focus on language-specific problems

Coverity

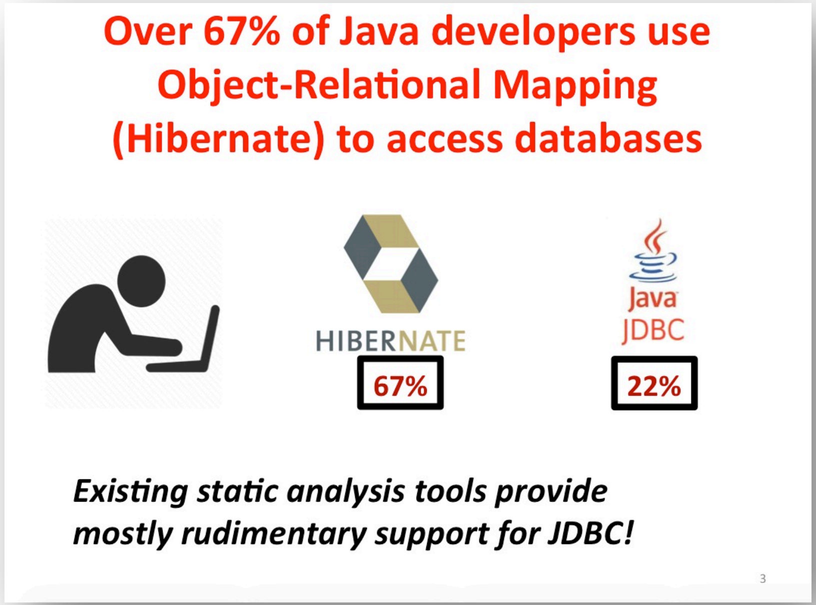
PMD

Google error-prone

FindBugs

Facebook Infer

However, many problems are related to how developers use different frameworks

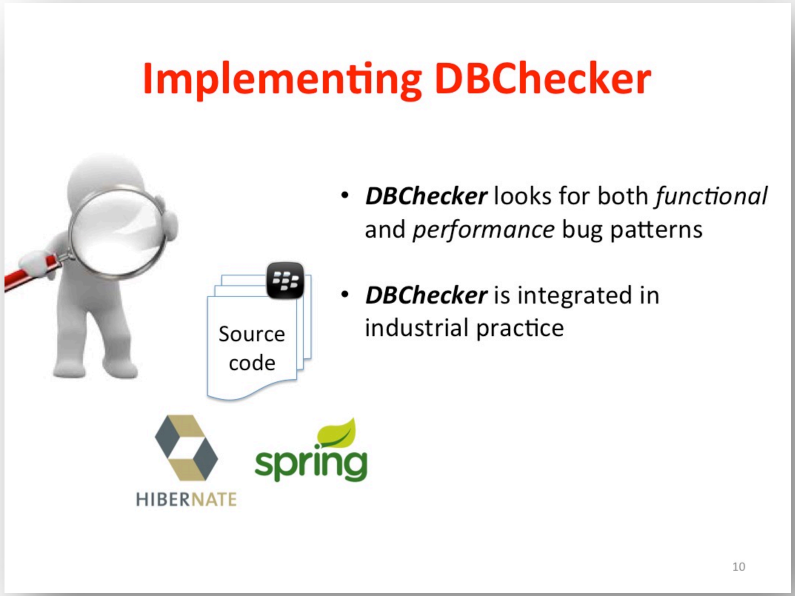


Over 67% of Java developers use Object-Relational Mapping (Hibernate) to access databases

HIBERNATE **67%**

Java JDBC **22%**

Existing static analysis tools provide mostly rudimentary support for JDBC!



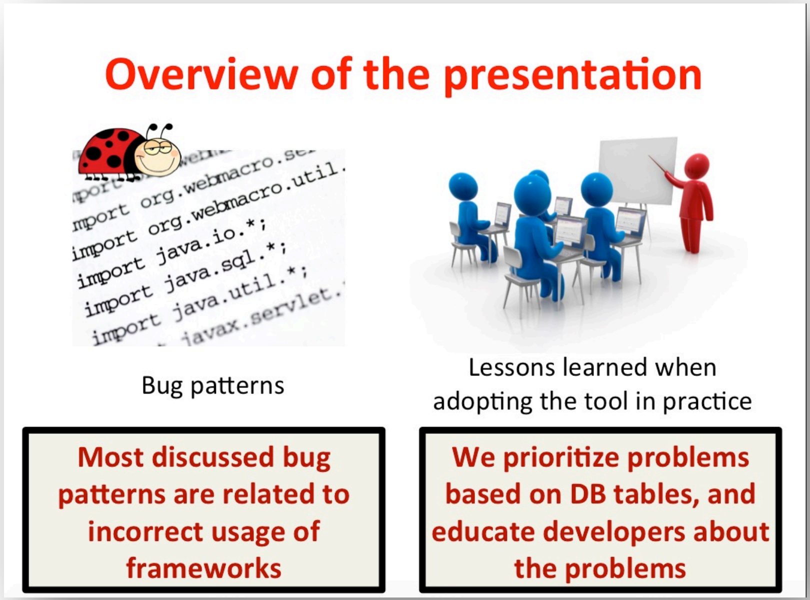
Implementing DBChecker

- ***DBChecker*** looks for both *functional* and *performance* bug patterns
- ***DBChecker*** is integrated in industrial practice

Source code

HIBERNATE

spring



Overview of the presentation

Bug patterns

Lessons learned when adopting the tool in practice

Most discussed bug patterns are related to incorrect usage of frameworks

We prioritize problems based on DB tables, and educate developers about the problems