RefaFlex - Safer Refactorings for Reflective Java Programs

TECHNISCHE UNIVERSITÄT DARMSTADT

EC SPRIDE
EUROPEAN CENTER FOR SECURITY AND PRIVACY BY DESIGN

Andreas Thies (Fernuni Hagen) and Eric Bodden (TU Darmstadt)

Motivation

- **Reflection** is a mechanism that allows programs to load and invoke program components through runtime-computed strings
- **A refactoring** is a program transformation that is meant to preserve the program's semantics
- **Integrated development environments** (IDEs) support many refactorings such as renaming or moving classes or members
- **Problem:** if programs use reflection, all refactorings that current IDEs support are unsound: they are unaware of accesses through reflection

rename field

```
package a;

class Super {
    public int j = 23;
}

public class C extends Super {
    public int i = password();
}                 j

public class Reflection {
    public static void main(String[] args) throws Exception {
        Class<?> c = Class.forName("a.C");
        Field f = c.getField("j");
        Object myC = c.newInstance();
        System.out.println(f.get(myC));
    }
}
```

program that accesses field `C.j` through the reflection API initially, this access returns `Super.j`; renaming `C.i` to `C.j` causes `C.j` (and thus the `password`) to be returned instead

RefaFlex from a User Perspective

- **Through instrumentation / method wrappers**, RefaFlex [TB12] monitors which classes and members the application actually accesses through reflection on a set of given test runs executed in the Eclipse IDE
- **The results of the dynamic analysis** are stored in a set of log files
- **The user** then invokes a refactoring in the Eclipse IDE
- **RefaFlex detects** if the application of the refactoring would rename or move classes or members that the application previously accessed through reflection on the recorded test runs
- **RefaFlex warns** the user that the refactoring may alter the program's behavior if such a situation is encountered

Constraint-Based Refactoring

- **RefaFlex combines** two existing tools: **TamiFlex** [BSS+11] (for monitoring reflective calls) and **RefaCola** [SKP11], the Refactoring Constraint Language
- **RefaCola** is a definition language for contraint-based refactorings
- **Declarative rules** written in Refacola express the programming language semantics (here Java)
- **Concrete refactoring tool** then uses these rules as patterns to generate constraints necessary to maintain the program's semantics
- **In RefaFlex** such constraints are also generated from runtime data about calls to the reflection API
- **Each solution** to the constraint system is a valid refactoring

$$\frac{\text{(Class\#getField , } \texttt{class}\text{, } \texttt{field}\text{)}}{\text{accessibility(} \texttt{field}\text{) = } \texttt{public}}$$

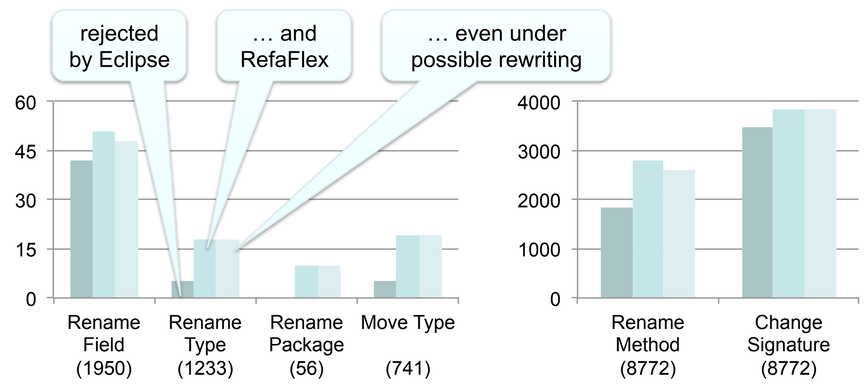
precondition from runtime analysis

constraint generation pattern

constraint generation rule stating that if the program accesses a `field` of some `class`, then the refactoring must assure that this `field` remains `public`

Empirical Evaluation

- **applied RefaCola** to 21,524 refactorings on 3 open-source projects



- **Results:** problem is real; RefaFlex helps solve the problem, should best be used in combination with existing tools

Future Work

- **Non-functional properties** such as security, performance and maintainability are impacted by program changes
- **Wish to extend RefaCola** with constraints that capture such properties
- **Result: users get informed** about the impact of program changes with respect to multiple dimensions
- **Suggest users** alternative but behaviorally equivalent program changes that are optimal with respect to non-functional properties

[TB12] RefaFlex: Safer Refactorings for Reflective Java Programs (Andreas Thies, Eric Bodden), In International Symposium on Software Testing and Analysis (ISSTA 2012), pages 1–14, 2012

[BSS+11] Taming Reflection: Aiding Static Analysis in the Presence of Reflection and Custom Class Loaders (Eric Bodden, Andreas Sewe, Jan Sinschek, Hela Oueslati, Mira Mezini), In International Conference on Software Engineering (ICSE 2011), pages 241–250, ACM, 2011.

[SKP11] A refactoring constraint language and its application to Eiffel (Friedrich Steimann, Christian Kollee, and Jens von Pilgrim), In European Conference on Object-oriented Programming (ECOOP 2011), pages 255-280, Springer, 2011.

[RBJ97] A refactoring tool for Smalltalk (Don Roberts, John Brant, and Ralph Johnson), In Theory and Practice of Object Systems, 3:253–263, October 1997.

SIGSOFT SPECIAL INTEREST GROUP ON SOFTWARE ENGINEERING

Awarded with SIGSOFT Distinguished Paper Award at ISSTA 2012

http://sse.ec-spride.de/

SECURE SOFTWARE ENGINEERING GROUP