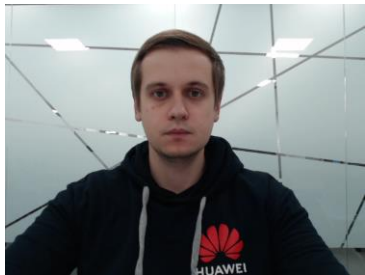
Diktat: Lightweight Static Analysis for Kotlin

Andrey Kuleshov[1], Petr Trifanov[1], Vladislav Frolov[1], Guangtai Liang[2]
[1] Russian Research Institute, Huawei Technologies Co., Ltd, Moscow, Russia
[2] Software Analysis Lab, Huawei Technologies Co., Ltd, Beijing, China
Email: {andrey.kuleshov, trifanov.petr, vladislav.frolov, liangguangtai}@huawei.com



CONTENT OF THE PAPER

I. INTRODUCTION
Prerequisites for creation of Diktat

II. CODE REPRESENTATIONS IN KOTLIN
AST, PSI, new IR

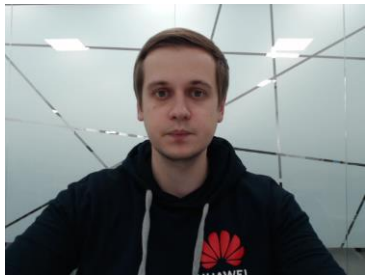
III. THE USAGE OF CONTEXT FOR CODE ANALYSIS
The usage of Kotlin code context for code analysis

IV. EXISTING SOLUTIONS
Existing open-source static analyzers for Kotlin

V. DIKTAT
New open-source light-weight static analyzer for Kotlin

VI. COMPARISSON
Comparison of Kotlin Analyzers: Metrics and Evaluation

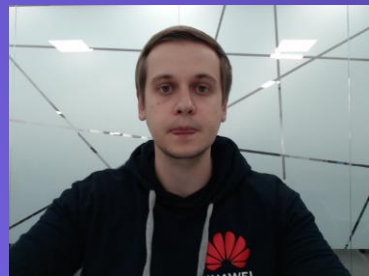
VII. CONCLUSION
Our results



Intro

- In the paper authors reviewed their popular open-source static analyzer and automatic code fixer for Kotlin called **Diktat***
- Authors tried to show that a lightweight AST analysis can be very powerful in specific scenarios
- Authors observed different types of Kotlin Internal Representation that could be used for static analysis: AST, PSI, new IR
- Paper contains the comparison of existing popular open-source static analyzers for Kotlin

*https://github.com/cqfn/diKTat



HUAWEI

Kotlin

Intermediate representations

- Simple AST
- Program Structure Interface (enriched AST)
- New Kotlin IR

```
1  if (a!=null) {}
```

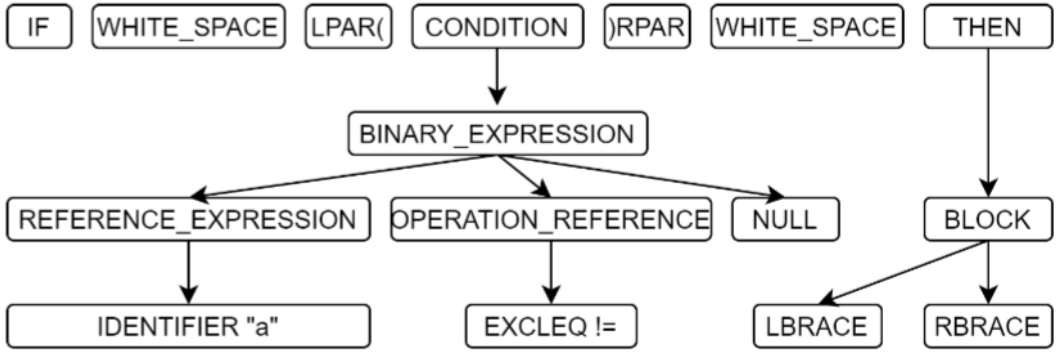
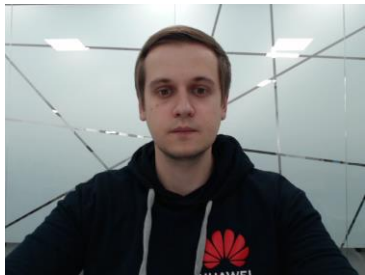


Fig. 2. Example of the PSI tree

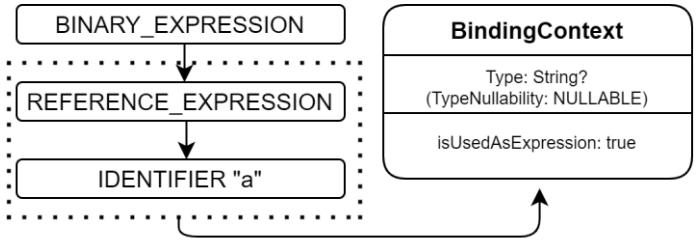


```
1  if (a!=null) {}
```

→

```
1  a?.let {}
```

Kotlin code analysis

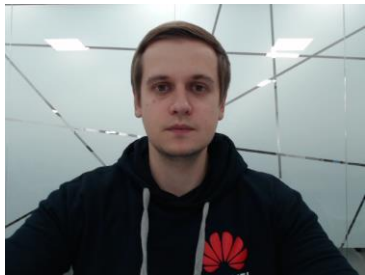


No-type and context Resolution

Analysis and autofixing are done only using PSI tree

Context Resolution

Analysis is done with **BindingContext** or Kotlin IR



Diktat

CONFIGURABLE
All inspections are configurable with a special file

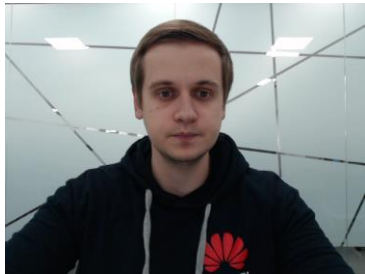
SUPRESS
All inspections could be suppressed in configuration or in code with annotation

PLUGINS
Gradle and Maven plugins

DETECTION
Detects 110+ types of design, code-style issues and bugs

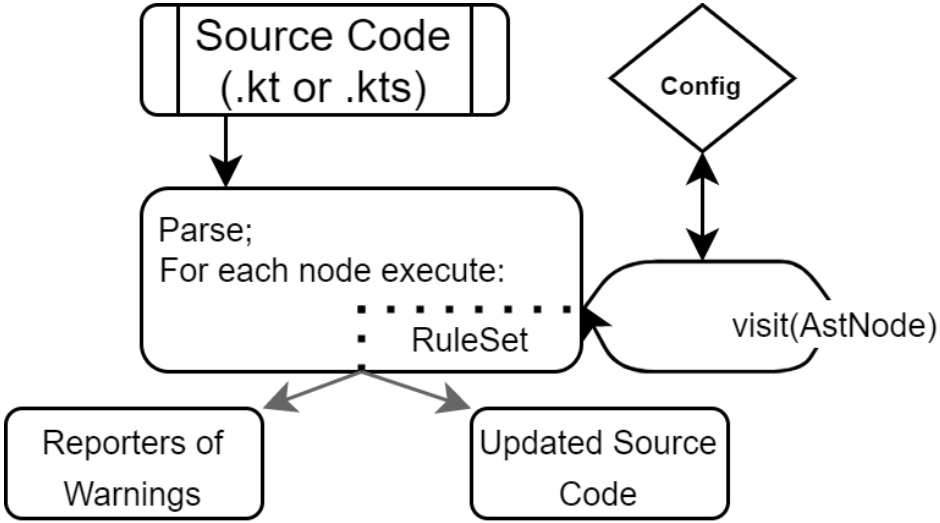
AUTO-FIX
Can automatically make a fix for 75+ inspections

LIGHTWEIGHT
Uses PSI and operates with syntax tree, can make analysis by a snippet of code



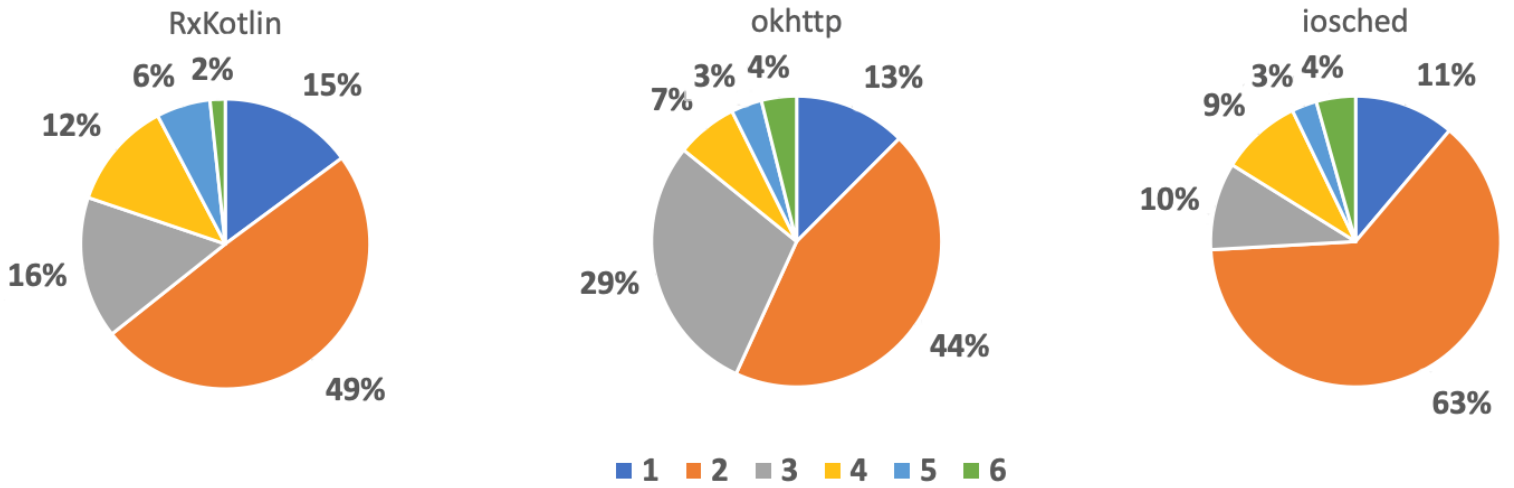
Processing on the high-level

```
1 @Suppress("LOCAL_VARIABLE_EARLY_DECLARATION")
2 fun foo() {
3     println("some logic")
4     val bar = calculation() + 42
5     function()
6     otherFunction(bar)
7 }
```

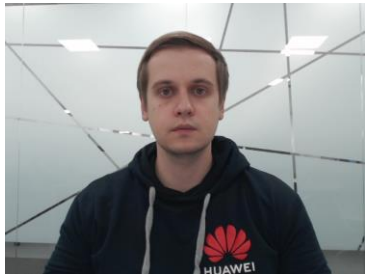


HUAWEI

Experiments



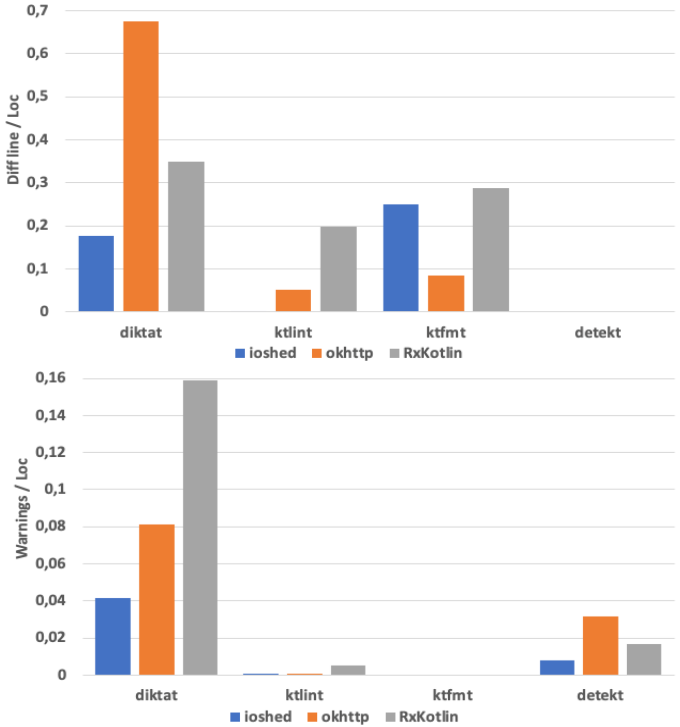
1 - Naming; **2** - Comments and Documentation; **3** - General formatting;
4 - Variables and types; **5** - Functions; **6** - Classes and interfaces



HUAWEI

Results

- In this paper authors of Diktat reviewed their way of solving industrial problems related to the code analysis in Kotlin language
- On real-life scenarios authors described problems when a lightweight AST analysis can be successfully used for detection and automated elimination of code issues
- This paper contains experiments with popular open-source static analyzers applied to mature and popular open-source libraries
- Results have proved that Diktat inspections detect and fix the number of issues higher than average



HUAWEI

THANK YOU

Follow the project updates:
https://github.com/cqfn/diKTat

To check the demo visit:
https://ktlint-demo.herokuapp.com