Code anomalies in Kotlin programs

Timofey Bryksin

Saint Petersburg
State University

Feb 3, 2019

NATIONAL RESEARCH
UNIVERSITY

What is Kotlin

- General purpose
- Statically typed
- Supports object-oriented and functional programming
- Interoperability with Java
- Runs on the JVM, Android, can be compiled to JavaScript or native code
- Also named after an island
- Open source
  - https://kotlinlang.org/
  - https://github.com/jetbrains/kotlin

Code in Java

```java
public void sendMessageToClient(
    @Nullable Client client,
    @Nullable String message,
    @NotNull Mailer mailer
) {
    if (client == null || message == null)
        return;

    PersonalInfo personalInfo = client.getPersonalInfo();
    if (personalInfo == null)
        return;

    String email = personalInfo.getEmail();
    if (email == null)
        return;

    mailer.sendMessage(email, message);
}
```

Code in Kotlin

```kotlin
fun sendMessageToClient(
        client: Client?,
        message: String?,
        mailer: Mailer
) {
    val email = client?.personalInfo?.email
    if (email != null && message != null) {
        mailer.sendMessage(email, message)
    }
}
```

https://try.kotlinlang.org/#/Kotlin Koans/Introduction/Nullable types/

“Abnormal” Kotlin code

```
"INTERPRET" compose {
    label("loop") // empty stack
    "BL WORD"()
    "DUP COUNT NIP"()
    cbranch("end") // name
    "FIND"()
    "DUP"()
    cbranch("word_not_found") // wptr flags
    "STATE @ 0="()
    cbranch("compile") // wptr flags
    "DUP -1 <> SWAP 2 AND AND"()
    cbranch("compile_only") // wptr
    "$msgCompileOnly COUNT TYPE HERE COUNT TYPE"() // this "HERE" is cheating, this word needs to be
refactored anyways
    "ABORT"()

    label("compile_only") // wptr
    ">CFA EXECUTE (VALIDATE-STATE)"()
    branch("loop") // empty stack
```

https://github.com/therealfarfetchd/kforth

Code anomalies detection

- What?
  - Code fragments not typical for the programming language community
  - Are syntactically and semantically correct
  - No examples available (almost)
- Why?
  - Unknown compiler defects
  - Compiler performance tests
  - Insights on the language improvement
- How?

Related work

- Nguyen et. al. Graph-based Mining of Multiple Object Usage Patterns. // ESEC/FSE’09
  - GrouMiner, detects anomalous object interactions (DAG analysis)
- Wasylkowski et. al. Detecting object usage anomalies // ESEC-FSE'07
  - detects unusual patterns sequences of method calls
- S. Hangal and M. Lam. Tracking down software bugs using automatic anomaly detection // ICSE'02
  - DIDUCE, dynamic analysis of invariants in code
- Feng et. al. Anomaly Detection Using Call Stack Information // IEEE Symposium on Security and Privacy, 2003
  - dynamic analysis of system calls

What should we analyze?

- Source code
  - how people write programs
- Byte code
  - how the compiler works
- Both!

What structural units should we analyze?

- Project
- File
- Class
- Function
- Code block
- Token

How?

- Traditional anomaly detection approach
  - vector representation of the input code
  - anomaly detection on vectorized data
- Classification of detected anomalies
- ???
- PROFIT!

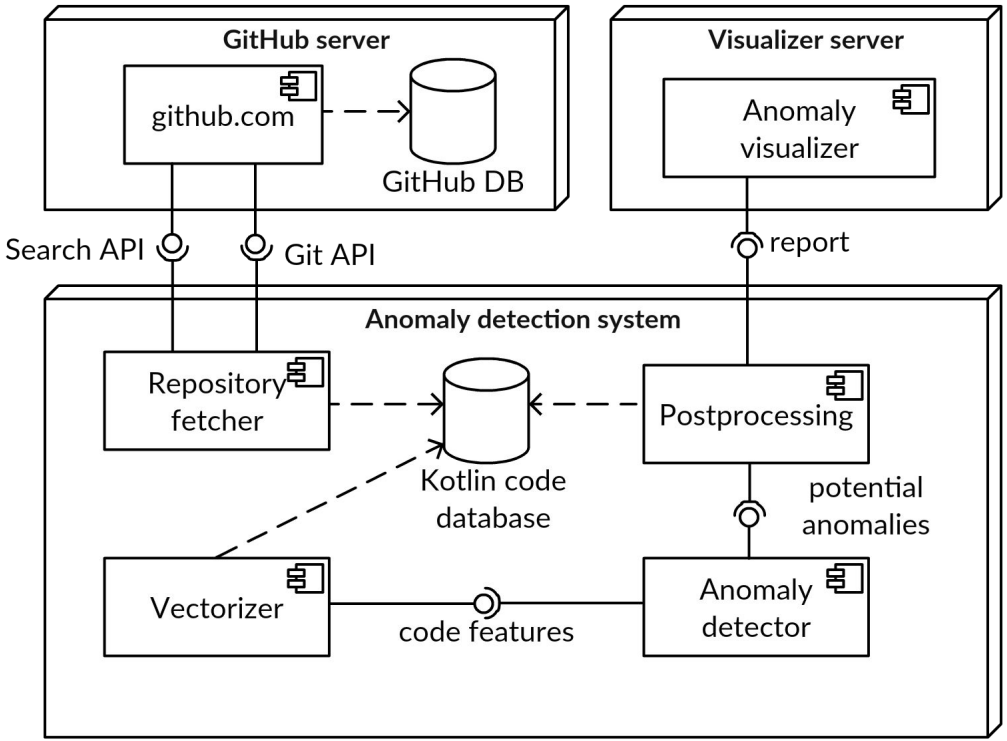
Code representation

- Explicit features
  - software metrics (AST or cohesion/coupling metrics),
  - simple natural language processing features (unigrams, bag-of-words)
  - path-based representations
  - ...
- Implicit features
  - N-grams
  - abstract syntax tree encoding
  - feature hashing
  - autoencoder neural networks
  - ...

Anomaly detection techniques

- Local Outlier Factor
- Isolation Forest
- Clustering algorithms
  - DBSCAN
  - ROCK
  - SNN
- Autoencoders
- One-class SVM

The pipeline



The dataset

- GitHub repositories
  - Kotlin as the main language
  - created before March 2018
  - are not forks of some other repositories
- 47 751 repositories
- 932 548 source files
- 4 044 790 functions

Experiment 1: explicit features

- Features: 51 software metrics values
  - general (LOC, number of nodes, AST height, etc.)
  - structural (cyclomatic complexity, loops depth, number of ‘when’ branches, etc.)
  - signature-specific (number of arguments, type parameters, annotations, etc.)
  - language element-specific (number of statements, operators, keywords, function calls, etc.)
- PCA reduction down to 20 (0.8 variance explained)
- Local Outlier Factor
  - contamination=0.0001
- Isolation Forest
  - contamination=0.0001, n_estimators=200

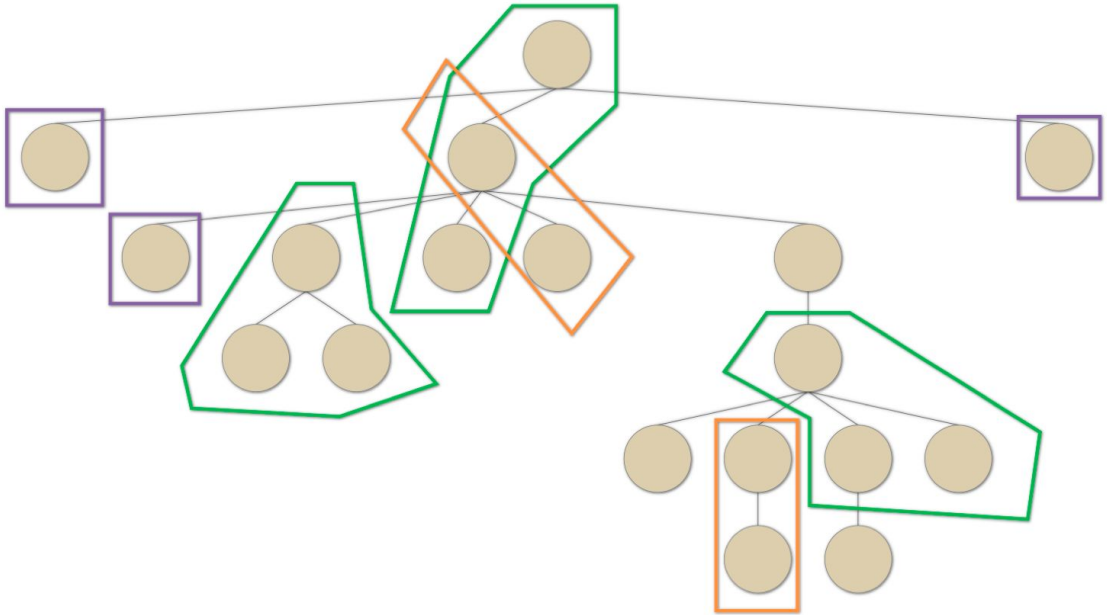
Experiment 1: results

Detected 322 anomalies

Method	Potential anomalies	Filtered anomalies
Local Outlier Factor	405	128 (36.1%)
Isolation Forest	405	179 (44.2%)

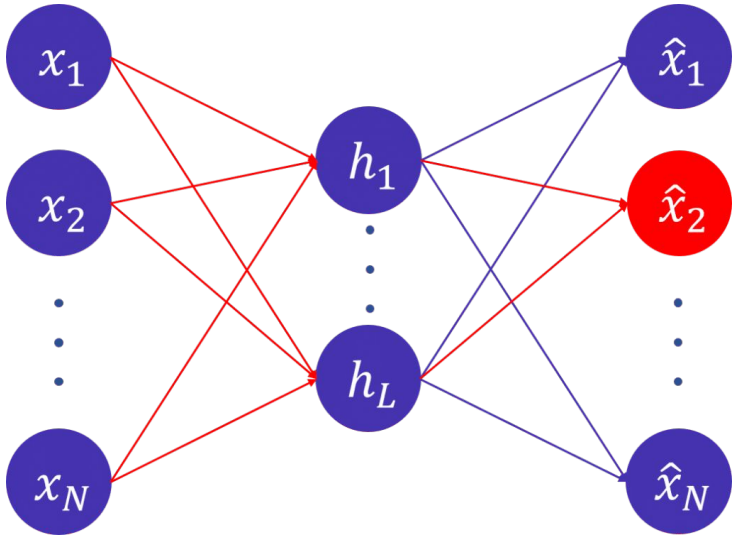
Experiment 2: implicit features

- Also analyzing functions
- Uni-, bi- and trigrams
  - 11k of them



Experiment 2: results

- Autoencoder neural network
  - minimizing the reconstruction error
  - epochs: 5, batch size: 1024, compression rates between 0.25 and 0.75
  - distance between input and output values, 3σ threshold
- Detected 362 anomalies



© https://www.incubegroup.com/blog/autoencoder-anomaly-detection/

Evaluation

Anomaly type	E_1	E_2	R	Anomaly type	E_1	E_2	R
Lots of when cases	+	+	5	Long enumeration lists		+	4
Lots of delegated props	+	+	5	Strange code constructs	+		4
Lots of generic type params	+	+	5	Lots of similar calls	+	+	4
Lost of nested calls		+	4	Lots of global functions		+	4
Lots of if statements	+	+	4	Complex functions	+	+	3
Large sets of constants		+	4	Long multiline strings		+	3
Complex annotations	+		4	Multiple catch blocks	+		3
Long call chains		+	4	Lots of function args	+	+	3

“Lots of when cases”

```
fun Activity.getThemeId(color: Int = baseConfig.primaryColor) = when (color) {
    -12846 -> R.style.AppTheme_Red_100
    -1074534 -> R.style.AppTheme_Red_200
    -1739917 -> R.style.AppTheme_Red_300
    -1092784 -> R.style.AppTheme_Red_400
    -769226 -> R.style.AppTheme_Red_500
    -1754827 -> R.style.AppTheme_Red_600
    -2937041 -> R.style.AppTheme_Red_700
[120 more]
    -13154481 -> R.style.AppTheme_Blue_Grey_800
    -14273992 -> R.style.AppTheme_Blue_Grey_900

    else -> R.style.AppTheme_Orange_700
}
```

“Lots of generic type parameters”

```
fun <L : Any, R : Any, R1 : Any, R2 : Any, ... , R22 : Any> validate(
        p1: Disjunction<L, R1>, p2: Disjunction<L, R2>, ... , p22:
Disjunction<L, R22>, ifValid: (R1, R2, ... , R22) -> R
): Disjunction<List<L>, R> {
    val validation = Validation(p1, p2, ... , p22)
    return if (validation.hasFailures) {
        Disjunction.Left(validation.failures)
    } else {
        Disjunction.Right(ifValid(p1.get(), p2.get(), ... , p22.get()))
    }
}
```

“Strange code constructs”

```
fun test() {
a(1
, {}
, { -> 1}
, {1}
, {x}
, {-> 1}
, {x -> 1}
, {x, y -> 1}
, {x -> 1}
, {(x)}
)
}
```

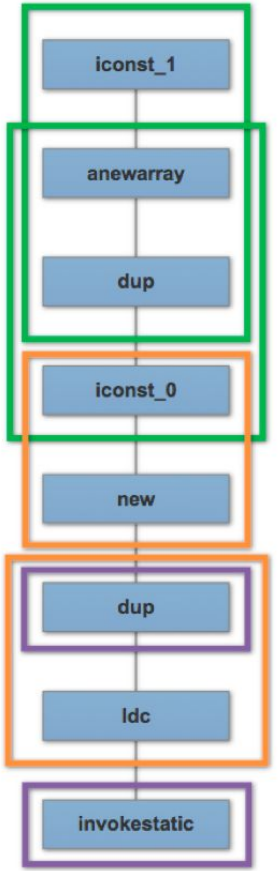
A parser bug found

```
fun main(args: Array<String>) {
    if (args.size > 0) {
        println("Hello Awesome Mobile Conferences)
        println("Hello Awesome Mobile Conferences)
        println("Hello Awesome Mobile Conferences)
        println("Hello Awesome Mobile Conferences)
        println("Hello Awesome Mobile Conferences)
        println("Hello Awesome Mobile Conferences)
        println("Hello Awesome Mobile Conferences)
        println("Hello Awesome Mobile Conferences)
        println("Hello Awesome Mobile Conferences)
        println("Hello Awesome Mobile Conferences)

[420 more]

        println("Hello Awesome Mobile Conferences)
   }
}
```

https://youtrack.jetbrains.com/issue/KT-27983

Experiment 3: conditional anomalies

- 40k source code files + bytecode for them
- N-grams + autoencoder network
- Detected 38 anomalies



```
class ConfigModel(val configs: Config) : ViewModel() {
    val ip = bind { configs.ipProperty }
    val dataBase = bind { configs.dataBaseProperty }
    val rootUser = bind { configs.rootUserProperty }
    val password = bind { configs.passwordProperty }
    val tableName = bind { configs.tableNameProperty }
    val entityName = bind { configs.entityNameProperty }
    val entityPackage = bind { configs.entityPackageProperty }
    val mapperPackage = bind { configs.mapperPackageProperty }
    val servicePackage = bind { configs.servicePackageProperty }
}
```

```
{
  "getIp" : [ "aload_0", "getfield", "areturn" ],
  "getDataBase" : [ "aload_0", "getfield", "areturn" ],
  "getRootUser" : [ "aload_0", "getfield", "areturn" ],
  "getPassword" : [ "aload_0", "getfield", "areturn" ],
  "getTableName" : [ "aload_0", "getfield", "areturn" ],
  "getEntityName" : [ "aload_0", "getfield", "areturn" ],
  "getEntityPackage" : [ "aload_0", "getfield", "areturn" ],
  "getMapperPackage" : [ "aload_0", "getfield", "areturn" ],
  "getServicePackage" : [ "aload_0", "getfield", "areturn" ],
  "getConfigs" : [ "aload_0", "getfield", "areturn" ],
  "<init>" : [ "aload_1", "ldc", "invokestatic", "aload_0", ... (4428 instructions)
}
```

https://github.com/wtksana/MG-gui/blob/master/src/main/java/com/wt/mggui/model/Config.kt

Current and future work

- Thorough comparative analysis
- Less naive metrics and anomaly detection techniques
- Semi-supervised learning
- Automated (automatic?) clustering and labeling of detected anomalies
- Working on different structural levels
- Feature-level anomalies
- Control-flow graph anomalies
- Going deeper into the compiler
- Kotlin/Android, Kotlin/Native, Kotlin/JS
- ...

Summary

- Detected 23 types of code anomalies
- 46 of 146 reported anomalies (12 types) were considered useful and used by the Kotlin compiler team

https://research.jetbrains.org/groups/ml_methods

https://github.com/ml-in-programming/kotlin-code-anomaly

https://github.com/PetukhovVictor/code-anomaly-detection