Поиск кодовых аномалий на языке Kotlin

Автор: Кирилл Смиренко

Санкт-Петербургский государственный университет
Кафедра системного программирования

14 мая 2018 г.

Кодовые аномалии языка Kotlin

- Кодовая аномалия — фрагмент кода, выделяющийся нестандартной структурой
- “Аномальный” код может быть синтаксически и семантически корректен
- Применение кодовых аномалий:
  - выявление неучтённых вариантов использования языковых конструкций
  - тестирование производительности компилятора и языковых инструментов
  - выявление недостатков самого языка программирования

```
1  fun <T1, T2, T3, T4, /* ... */, T22, U> Converters.object22
     (mapper: (T1, T2, T3, T4, /* ... */, T22) -> U, t1:
     Pair<Pair<String, (U) -> T1>, Converter<T1>>, t2:
     Pair<Pair<String, (U) -> T2>, Converter<T2>>, t3:
     Pair<Pair<String, (U) -> T3>, Converter<T3>>, t4:
     Pair<Pair<String, (U) -> T4>, /* ... */, t22:
     Pair<Pair<String, (U) -> T22>, Converter<T22>>):
     Converter<U> {
2      val encoder: Encoder<U> = {
3          Encoders.object_(t1.first.first to t1.second.first(t1
     .first.second(it)), t2.first.first to t2.second.first(t2
     .first.second(it)), t3.first.first to t3.second.first(t3
     .first.second(it)), t4.first.first to t4.second.first(t4
     .first.second(it)), /* ... */, t22.first.first to t22.second
     .first(t22.first.second(it)))
4      }
5      val decoder = map(Decoders.field(t1.first.first,
     t1.second.second), Decoders.field(t2.first.first, t2.second
     .second), Decoders.field(t3.first.first, t3.second.second),
     Decoders.field(t4.first.first, t4.second.second), /* ... */
     Decoders.field(t22.first.first, t22.second.second), mapper)
6      return encoder to decoder
7  }
```

Постановка задачи

Цель: создание системы поиска и анализа кодовых аномалий на языке Kotlin

Задачи:

- Провести обзор предметной области
- Выполнить проектирование системы поиска кодовых аномалий
- Реализовать требуемую систему
- Провести апробацию системы и предоставить отчёт разработчикам языка Kotlin

Поиск кодовых аномалий

- Задача сводится к известной задаче поиска аномалий в данных
- Выделяются подзадачи:
  - векторизация исходного кода на Kotlin
  - поиск выбросов в векторизованных данных

Обзор: векторизация программного кода

- Наиболее популярные подходы:
  - вычисление кодовых метрик
  - получение неявных признаков из дерева разбора
- Работы, связанные с метриками кода:
  - Milepost GCC: Machine Learning Enabled Self-tuning Compiler (Fursin et al., 2011)
  - De-anonymizing Programmers via Code Stylometry (Caliskan-Islam et al., 2015)
  - Suggesting Accurate Method and Class Names (Allamanis et al., 2015)
  - Unsupervised Learning Based Approach for Plagiarism Detection in Programming Assignments (Yasaswi et al., 2017)
- MetricsReloaded — библиотека синтаксических метрик Java-кода

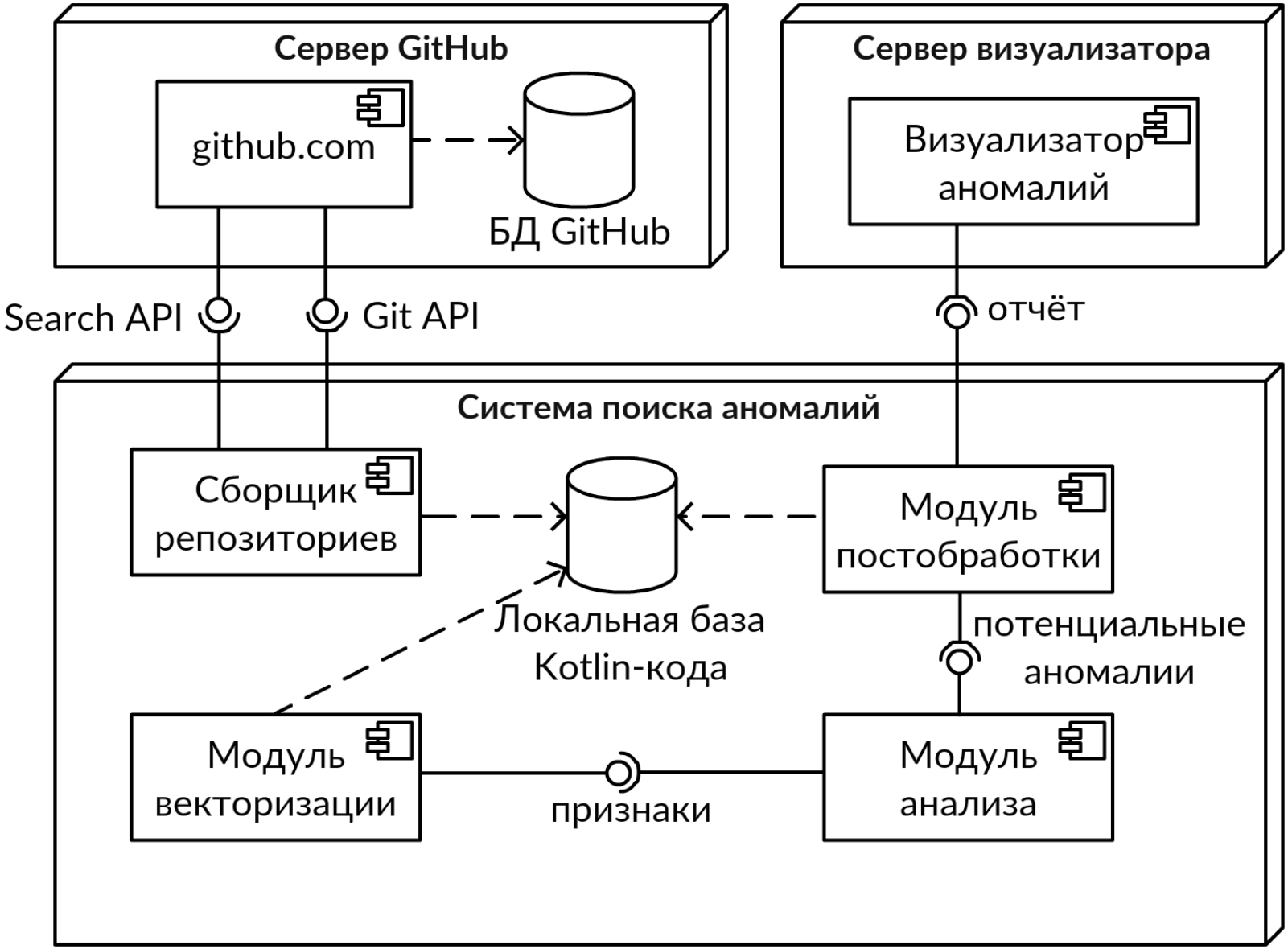
Обзор: поиск аномалий

- Анализ экстремальных значений
- Эллипсоидальная аппроксимация (Elliptic Envelope)
- Фактор локальных выбросов (Local Outlier Factor)
- Метод опорных векторов (Support Vector Machine)
- Метод “изолирующий лес” (Isolation Forest)

- Anomaly Detection: A Survey (Chandola et al., 2009)
- A survey of outlier detection methodologies (Hodge et al., 2004)

- Библиотека Scikit-learn предоставляет реализации алгоритмов

Архитектура системы



Сбор репозиториев

- Поиск репозиториев на GitHub по следующим критериям:
  - язык Kotlin указан как основной
  - создан в указанный промежуток времени
  - не ответвлённый репозиторий
- Поиск ведётся с учётом ограничений API поиска GitHub
- Скачивание репозиториев средствами Git

Векторизация кода

- Единица векторизации — функция
- Из конкретного синтаксического дерева (Program Structure Interface) извлекается 51 признак
- Каждой функции присваивается порядковый номер и “сигнатура”

Метрики Kotlin-функций

- общие
  - число строк кода, количество узлов, высота CST
- внешние
  - количество формальных аргументов, типовых параметров, аннотаций; наличие модификатора suspend и др.
- структурные
  - цикломатическая сложность, глубина вложенности циклов, среднее и максимальное число веток в when-выражении
- число определённых элементов языка
  - выражений, операторов, ключевых слов, вызовов функций, строковых шаблонов и др.

Поиск аномалий

- EllipticEnvelope(contamination=0.0001)
- LocalOutlierFactor(contamination=0.001, n_neighbors=20, metric='minkowski', p=2)
- IsolationForest(contamination=0.0001, n_estimators=200)

- Параметры требуется подбирать под конкретный набор данных
  - параметр зашумлённости (contamination) задаёт долю потенциальных аномалий

Апробация: локальные эксперименты

- Скачан 47 751 репозиторий с датой создания по 18.02.2018 г.
- Проанализировано 4 044 790 Kotlin-функций
- Установлен параметр зашумлённости 0,01%
- Отобраны наиболее интересные примеры и разделены на 23 класса

Метод	**Потенц. ан.**	**Отобрано**	**Доля**
EllipticEnvelope	405	17	4,2 %
IsolationForest	405	179	44,2 %
LocalOutlierFactor	405	128	31,6 %
Итого (без дубликатов)	1213	322	26,5 %

Апробация: экспертная оценка

- Q — экспертная оценка полезности класса
- 7 из 23 классов получили высокие оценки (4 и 5)

Краткое описание	Q	Краткое описание	Q
Много типовых параметров	**5**	Много throw-операторов	2
Много веток в when-выражении	**5**	Много ссылок на класс	2
Много делегированных свойств	**5**	Много схожих фрагментов кода	2
Необычные кодовые конструкции	**4**	Много лямбда-выражений	2
Сложные аннотации функции	**4**	Много пустых строковых литералов	1
Много if-выражений	**4**	Много преобразований типов	1
Много схожих выражений	**4**	Много assert-операторов	1
Сложная структура кода	3	Код из руководств по Kotlin	1
Большое тело функции	3	Много строковых литералов	1
Много конструкций try-catch	3	Много локальных переменных	1
Много циклов	2	Много вложенных функций	1
Большие литеральные коллекции	2		

Результаты

- Выполнен обзор предметной области
- Создана архитектура системы поиска и анализа кодовых аномалий на языке Kotlin
- Реализована требуемая система
- Проведена апробация системы на 47 751 проектах на Kotlin с GitHub
- Отчёт со 111 наиболее интересными аномалиями передан разработчикам языка Kotlin

Метрики функций на Kotlin

avgNumBlockChildren	numCollectionLiterals	numOneConstants
avgNumWhenEntries	numConstExpr	numOperationReferences
cstHeight	numDeclarations	numPlusOperations
cyclomaticComplexity	numDistinctKeywords	numReferences
designComplexity	numEmptyBlocks	numReturns
isSuspend	numEmptyStringLiterals	numSafeExpressions
isVoid	numExpressions	numStatementExpressions
maxLoopNestingDepth	numFinally	numStringLiteralTemplates
maxNumBlockChildren	numForceUnwraps	numStringTemplates
maxNumWhenEntries	numIfExprs	numThrows
nodeCount	numKeywords	numTry
numAnnotations	numLambdas	numTypeParameters
numAssigns	numLoopStatements	numTypecastExpr
numBlockStringTemplates	numMethodCalls	numValueParameters
numBlocks	numNestedClasses	numZeroConstants
numCatch	numNestedFuns	relativeLoc
numClassLiterals	numOneChildBlocks	sloc

```
1   "DOPRI" {}
2   "DOCOL" {}
3   d.DOCOL_ptr = d.toCfa(d.latest)
4   "DOVAR" { data.push(fip); fip = ret.pop() }
5   d.DOVAR_ptr = d.toCfa(d.latest)
6   "DOCON" { data.push(mem.read32(fip)); fip = ret.pop() }
7   d.DOCON_ptr = d.toCfa(d.latest)
8   "DODOES" { ip = mem.read32(fip) } // TODO: is this correct?
9
10  "EXIT" { fip = ret.pop() }
11  d.EXIT_ptr = d.toCfa(d.latest)
12  "LIT" { data.push(mem.read32(fip)); fip += cellsize } IS
      COMPILE_ONLY
13  d.LIT_ptr = d.toCfa(d.latest)
14
15  "BRANCH" { fip += mem.read32(fip) } IS COMPILE_ONLY
16  d.BRANCH_ptr = d.toCfa(d.latest)
17  "0BRANCH" { fip += if (!data.pop().v) mem.read32(fip) else 4
      } IS COMPILE_ONLY
18  d.CBRANCH_ptr = d.toCfa(d.latest)
```

Обозначение	W_1	W_2	Q	Обозначение	W_1	W_2	Q
kotlin_tutorials	+		1	many_function_arguments		+	3
many_nested_structures		+	1	many_consecutive_arithm_expr		+	3
many_assertions	+		1	many_assignment_statements		+	3
many_literal_strings	+	+	1	many_try_catch	+		3
many_empty_string_literals	+		1	big_multiline_strings		+	3
many_type_casts	+		1	big_methods	+	+	3
many_safe_calls		+	1	big_code_hierarchy	+	+	3
big_init_method_in_bytecode		+	1	many_root_function_definitions		+	4
big_companion_object		+	1	many_similar_call_expressions	+	+	4
many_nested_functions	+		1	many_sq_bracket_annotations		+	4
many_local_variables	+		1	big_and_complex_enums		+	4
big_static_arrays_or_map	+	+	2	strange_code_constructs	+		4
many_loops	+	+	2	long_enumerations		+	4
complex_or_long_logical_exprs		+	2	long_calls_chain		+	4
many_type_reified_params		+	2	complex_annotation	+		4
many_throws	+		2	big_constants_set		+	4
many_lambdas	+		2	many_if_statements	+	+	4
many_class_references	+		2	nested_calls		+	4
many_concatenations		+	2	many_generic_parameters	+	+	5
many_inline_functions		+	2	many_delegate_properties	+	+	5
many_similar_code_fragments	+		2	many_case_in_when	+	+	5
many_not_null_assertions		+	2				