

Анализ качества автодополнения кода в интегрированных средах разработки

Калина Алексей Игоревич

научный руководитель: канд. физ.-мат. наук, ст. преп.

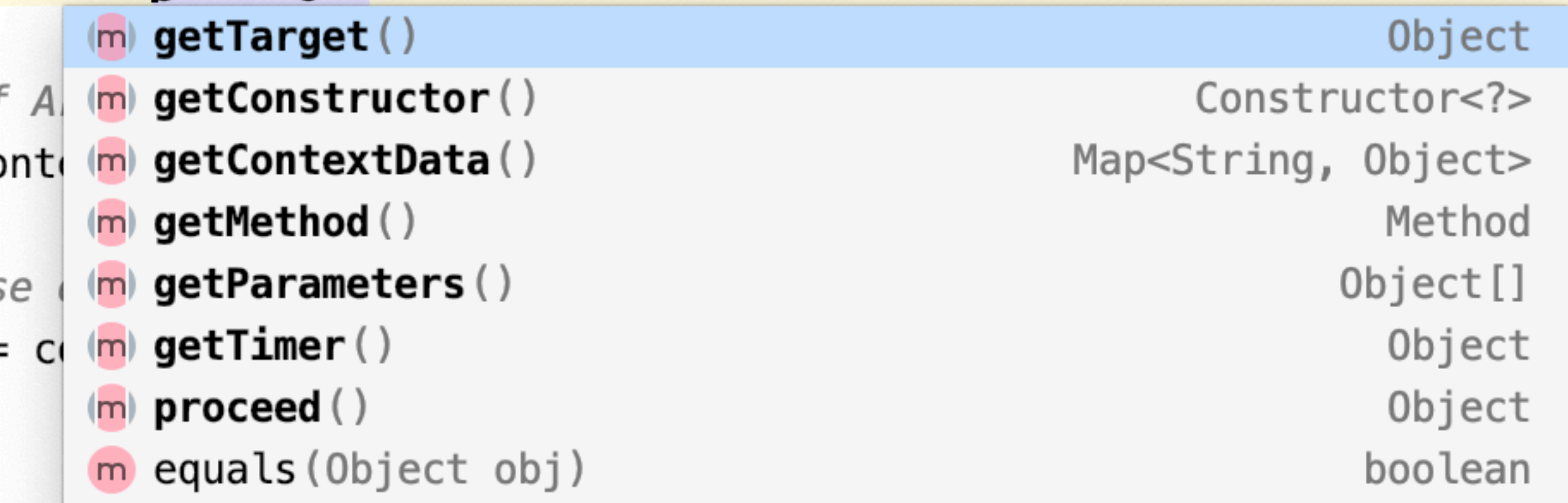
Луцив Д.В.

рецензент: программист ООО “Интеллиджей Лабс”

Бибаев В.И.

Автодополнение кода в IntelliJ IDEA

```
public Object onConstruct(InvocationContext context) throws Exception {  
    // null before the InvocationContext.proceed() returns  
    Object target = context.getTarget();  
    isNull(target);  
    // null in case of A  
    Method method = context.getMethod();  
    isNull(method);  
    // NOT null in case of B  
    Constructor ctor = context.getConstructor();  
    isNotNull(ctor);  
}
```



The image shows a screenshot of the IntelliJ IDEA code editor. A yellow highlight is under the line `Object target = context.getTarget();`. To the left of this line is a lightbulb icon. A dropdown menu is open, showing a list of methods available for `context`. The first method, `getTarget()`, is highlighted in blue. The other methods listed are `getConstructor()`, `getContextData()`, `getMethod()`, `getParameters()`, `getTimer()`, `proceed()`, and `equals(Object obj)`. Each method is preceded by a small red circle containing the letter 'm'. To the right of each method name is its return type: `Object`, `Constructor<?>`, `Map<String, Object>`, `Method`, `Object[]`, `Object`, `Object`, and `boolean` respectively.

Method	Return Type
<code>getTarget()</code>	<code>Object</code>
<code>getConstructor()</code>	<code>Constructor<?></code>
<code>getContextData()</code>	<code>Map<String, Object></code>
<code>getMethod()</code>	<code>Method</code>
<code>getParameters()</code>	<code>Object[]</code>
<code>getTimer()</code>	<code>Object</code>
<code>proceed()</code>	<code>Object</code>
<code>equals(Object obj)</code>	<code>boolean</code>

Оценка качества автодополнения

Зачем?

- Как влияет изменение в реализации автодополнения на его качество?
- В каких ситуациях автодополнение работает плохо?
- Как сравнить два алгоритма автодополнения кода?

Как оценивать качество автодополнения

online	offline
требуется участия пользователей: собираем логи с сессиями автодополнения	без участия пользователей: используем готовую кодовую базу
✓ точно отражает действительность	✗ искусственные запросы автодополнения не всегда соответствуют реальным
✗ длительный процесс	✓ быстрые результаты

Цель и задачи

Цель:

Создание инструмента для offline оценки качества автодополнения кода

Задачи:

- Анализ существующих подходов к оценке качества автодополнения
- Реализация инструмента оценки качества автодополнения кода
- Сравнение искусственных запросов автодополнения с пользовательскими

Обзор

- Многие исследователи занимаются разработкой автодополнения кода
- Для оценки качества своих алгоритмов они используют собственные фреймворки
- Авторы* показали, что в 57% случаев в подобных системах автодополнение вызывается для пунктуационных символов



* Hellendoorn et al. When Code Completion Fails: A Case Study on Real-World Completions, ICSE'19

Принцип работы

```
void parse(InputStream in, OutputStream out) {  
    BufferedInputStream bis = new BufferedInputStream(in);  
    bis.mark(0);  
    while (char c = bis.read()  $\neq$  -1) {  
        out.write(c);  
    }  
    bis.reset();  
}
```

Принцип работы





```
void parse(InputStream in, OutputStream out) {  
    BufferedInputStream bis = new BufferedInputStream(in);  
    bis.mark(0);  
    while (char c = bis.read()  $\neq$  -1) {  
        out.write(c);  
    }  
    bis.<cursor>();  
}
```

read	()	int
close	()	void
reset	()	void

Ожидаемый результат

```
void parse(InputStream in, OutputStream out) {  
    BufferedInputStream bis = new BufferedInputStream(in);  
    bis.mark(0);  
    while (char c = bis.read()  $\neq$  -1) {  
        out.write(c);  
    }  
    bis.reset();  
}
```

Верный элемент на позиции:

-  — 1
-  — 2..5
-  — >5
-  — отсутствует

Реальность

- разные языки программирования
- разные среды разработки
- длительное выполнение на больших объемах

Схема решения



Унификация языка



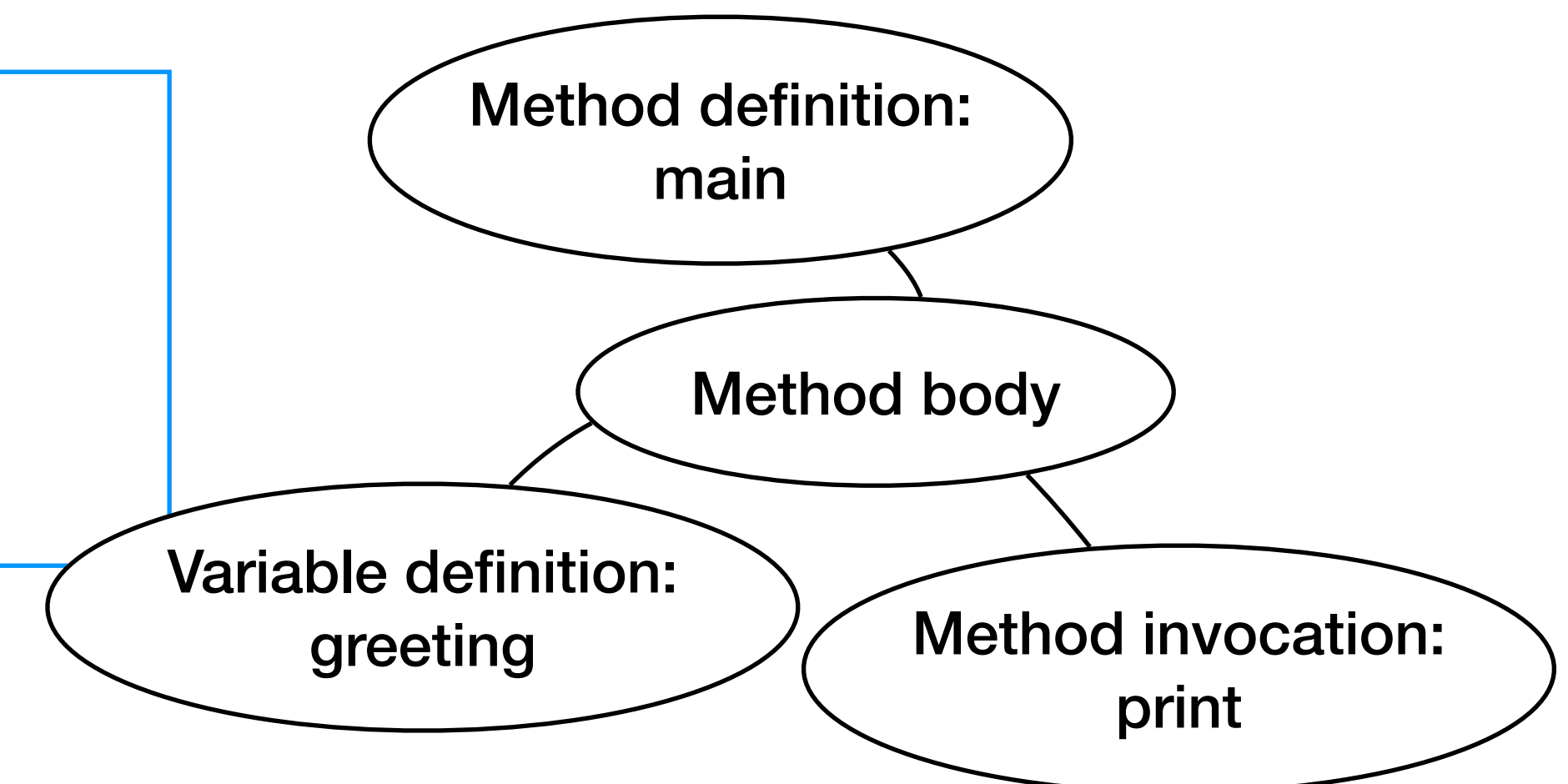
Java:

```
void main() {  
    String greeting = "hello";  
    System.out.print(greeting);  
}
```

Python:

```
def main():  
    greeting = "hello"  
    print(greeting)
```

Unified AST:



Генерация IDE действий



Сдвиг каретки
Печать текста
Удаление интервала текста
Вызов автодополнения
Открытие файла
Закрытие файла

Контексты

```
void parse(InputStream in, OutputStream out) {  
    BufferedInputStream bis = new BufferedInputStream(in);  
    while (char c = bis.read()  $\neq$  -1) {  
        out.write(c);  
    }  
}
```

Полный:

```
void parse(InputStream in, OutputStream out) {  
    BufferedInputStream bis = new <cursor>(in);  
    while (char c = bis.read()  $\neq$  -1) {  
        out.write(c);  
    }  
}
```

Предыдущий:

```
void parse(InputStream in, OutputStream out) {  
    BufferedInputStream bis = new <cursor>  
}
```

Префиксы

```
BufferedInputStream bis = new BufferedInputStream(in);
```

- Пустой:

```
BufferedInputStream bis = new <cursor>(in);
```

- Простой:

```
BufferedInputStream bis = new Buf<cursor>(in);
```

- Капитализированный:

```
BufferedInputStream bis = new BIS<cursor>(in);
```

Фильтры

ВЫЗОВЫ МЕТОДОВ:

```
bis.mark(0);  
while (char c = bis.read()  $\neq$  -1) {  
    out.write(c);  
}  
bis.reset();
```

Обращения к переменным:

```
bis.mark(0);  
while (char c = bis.read()  $\neq$  -1) {  
    out.write(c);  
}  
bis.reset();
```

Все:

```
bis.mark(0);  
while (char c = bis.read()  $\neq$  -1) {  
    out.write(c);  
}  
bis.reset();
```


Исполнение действий



Сессия автодополнения

Исходный токен: reset

Выдача:

read	()	int
close	()	void
reset	()	void

Получение результатов



Отчеты: Общий отчет

3577 file(s) successfully processed

After

Metrics visibility

Redraw table

Show empty rows

File Report ▲	Found@1 BASIC ▼	Found@5 BASIC ▲	Mean Rank BASIC ▲	Recall BASIC ▲	Sessions BASIC ▲
Summary	0.681	0.878	2.417	1.000	49213
AfterSuiteEvent.java	1.000	1.000	0.000	1.000	1
GitPushAfterCommitD...	1.000	1.000	0.000	1.000	4
AfterTestEvent.java	0.800	1.000	0.467	1.000	15
ShowDiffAfterWithLo...	0.500	0.500	7.500	1.000	2
UpdateBreakpointsAft...	0.467	0.667	15.667	1.000	15

Page Size 25 ▾ First Prev 1 Next Last

- **Found@k** — доля сессий, в которых верный элемент был среди первых k позиций выдачи
- **Recall** — доля сессий, в которых верный элемент присутствовал в выдаче
- **Mean Rank**— средняя позиция верного элемента в выдаче по всем сессиям
- **Mean Latency** — средняя длительность сессии автодополнения по всем сессиям

Отчеты: Отчет для файла

```
9 export function parseCookieValue(cookieStr: string, name: string): string|null {
10   name = encodeURIComponent(name);
11   for (const cookie of cookieStr.split(';')) {
12     const eqIndex = cookie.indexOf('=');
13     const [cookieName, cookieValue]: string[] =
14       eqIndex = -1 ? [cookie, ''] : [cookie.slice(0, eqIndex), cookie.slice(eqIndex + 1)];
15     if (cookieName.trim() === name) {
16       return decodeURIComponent(cookieValue);
17     }
18   }
19   return null;
20 }
21
```

prefix: " st "; latency: 288
string
ast

Верный элемент на позиции

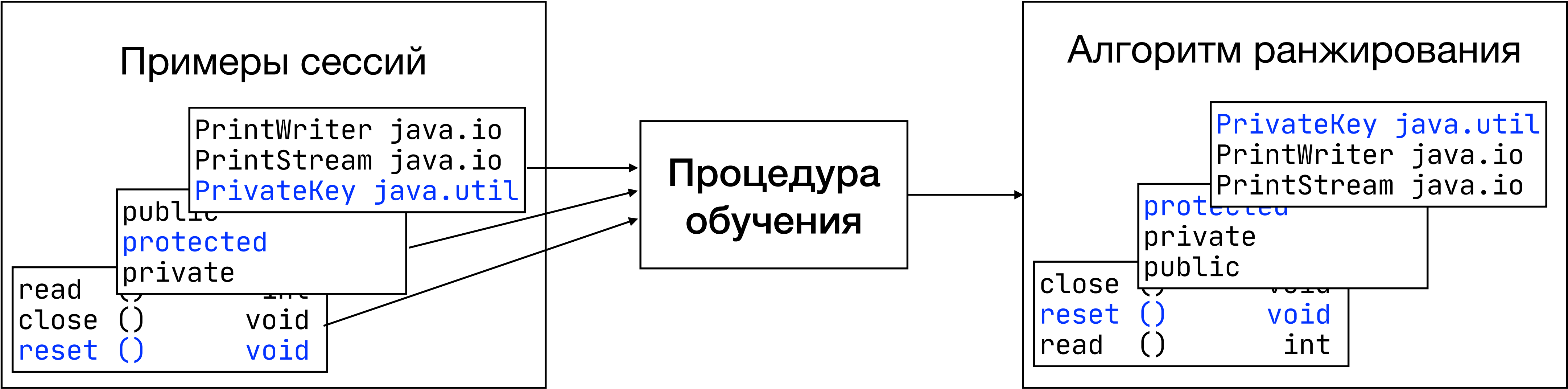
- 1

- 2..5

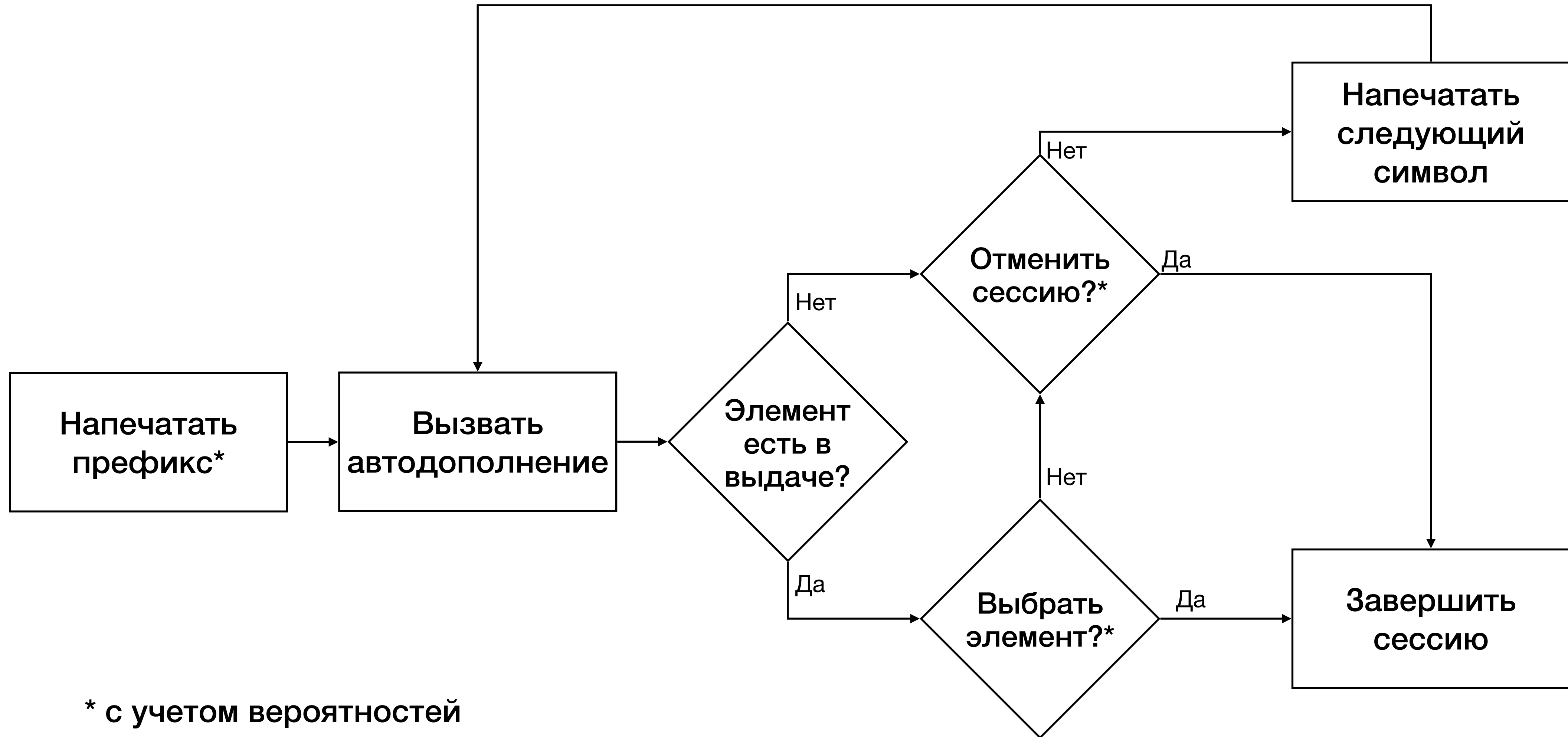
- >5

- отсутствует

Ранжирование выдачи автодополнения



Моделирование пользователя



Эксперименты

Был использован исходный код из Github репозиториев

репозиторий	описание	число сессий, тыс.	число звезд, тыс.
JetBrains/intellij-community	Среда разработки	21	10
square/retrofit	HTTP-клиент	27	36

Общее число сессий: 48 тысяч

Сравнение моделей с исходным ранжированием

	<i>user</i>	<i>methods</i>	<i>all</i>	<i>all + modeling</i>
Прирост Found@1, % (больше — лучше)	<i>1.91</i>	-0.23	1.15	1.82
Прирост Found@5, % (больше — лучше)	<i>3.19</i>	0.17	2.45	2.67
Прирост Mean Rank (меньше — лучше)	<i>0.834</i>	0.834	0.794	0.835

- *user* — реальные пользовательские сессии
- *methods* — вызовы методов с единичным префиксом
- *all* — все токены с единичным префиксом
- *all + modeling* — все токены и моделирование пользователя

Результаты

- Исследованы подходы к оценке качества автодополнения
- Разработан плагин оценки качества автодополнения. На данный момент он поддерживает 10 языков программирования
- Эксперименты показали, что моделирование пользователя позволяет формировать наиболее близкие к реальным запросы автодополнения