

Спикер: Михаил Костицын
Эксперт: Максим Паршин

Использование инструментов рефакторинга IDE в ИИ-агентах

О нас



Михаил Костицын

Спикер

Ведущий разработчик **Veai**

Начал путь в разработке с анализа программ и формальной верификации: JetBrains Research, Huawei

С 2023 года занимаюсь разработкой систем с искусственным интеллектом



Максим Паршин

Эксперт

Ведущий разработчик **Veai**

Ранее разрабатывал движок символьного исполнения и генерацию тестов в Huawei

Разрабатываю ИИ-агента для JetBrains IDE в Veai

Исследую, как подружить формальные методы анализа программ с ИИ-моделями

План доклада

01

Введение

02

Подходы к
рефакторингу в агентах

03

Итоги

Определение

Рефакторинг – это изменение структуры кода без изменения поведения

Задача 1: найти все места

Нужна полнота: 97 % не годится

78 вызовов × 97 % = 2 пропуска

Задача 2: применить

Нужна корректность: правка не меняет поведение программы

Зачем это QA

- Миграция фреймворка: Selenium → Playwright, JUnit 4 → 5
- Рефакторинг тестов: переход на Page Object, унификация, фикстуры
- Смена тестового API: переименование, смена базового класса тестов
- ...

Зелёные тесты не всегда доказывают, что поведение не изменилось

Кейс: detekt

detekt – статический анализатор Kotlin: набор правил, каждое ищет в коде свой вид проблем
Правила тестируют на сниппетах – короткие фрагменты Kotlin-кода прямо в тесте

```
fun Rule.compileAndLint(code)
```

сначала компилирует сниппет компилятором Kotlin, потом прогоняет правило

```
fun Rule.lint(code)
```

сразу прогоняет правило: сниппет может быть неполным и не компилироваться

ЗАДАЧА: PR #7873

Оставить одну функцию: lint(code, compile = true)

compileAndLint(code) → lint(code) – 1428 вызовов

lint(code) → lint(code, compile = false) – 78 вызовов

Суммарно нужно изменить: 1506 вызовов

Не менять: перегрузку lint(ktfFile) – 36 вызовов, FormattingRule.lint – 66 вызовов

Кейс: detekt

Simplify test api #7873

**Merged**

BraisGabin merged 2 commits into `main` from `simplify-test-api` on Feb 20, 2025



Conversation **2**



Commits **2**



Checks **0**



Files changed **134**



Commits on Feb 19, 2025

Simplify api



BraisGabin committed on Feb 19, 2025

Rename compileAndLint to lint



BraisGabin committed on Feb 19, 2025

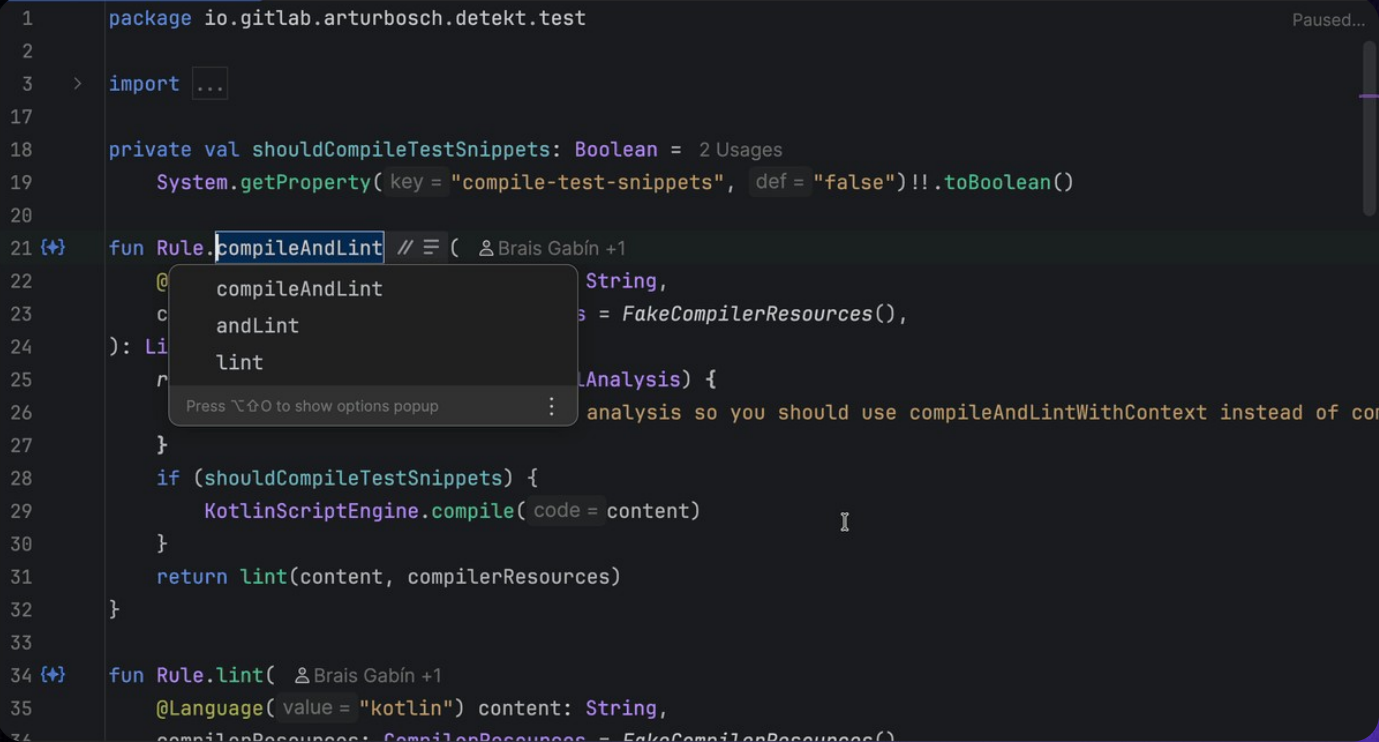
Инструменты IDE: Find Usages

```
20
21 {+} fun Rule.compileAndLint( @Brais Gabín +1
22     @Language( value = "kotlin") content: String,
23     compilerResources: CompilerResources = FakeCompilerResources(),
24 ): List<Finding> {
25     require( value = this !is RequiresFullAnalysis) {
26         "${this.ruleName} requires full analysis so you should use compileAndLintWithContext instead of
27     }
28     if (shouldCompileTestSnippets) {
29         KotlinScriptEngine.compile( code = content)
30     }
31     return lint(content, compilerResources)
32 }
33
34 {+} fun Rule.lint( @Brais Gabín +1
35     @Language( value = "kotlin") content: String,
36     compilerResources: CompilerResources = FakeCompilerResources(),
37 ): List<Finding> {
38     require( value = this !is RequiresFullAnalysis) {
39         "${this.ruleName} requires full analysis so you should use lintWithContext instead of lint"
40     }
41     val ktFile = compileContentForTest(content)
42     return visitFile( root = ktFile, compilerResources = compilerResources).filterSuppressed( rule = this)

```

compileAndLint()

Инструменты IDE: Rename



The screenshot shows a Kotlin file in an IDE. The code is as follows:

```
1 package io.gitlab.arturbosch.detekt.test
2
3 > import ...
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18 private val shouldCompileTestSnippets: Boolean = 2 Usages
19     System.getProperty( key = "compile-test-snippets", def = "false")!!.toBoolean()
20
21 fun Rule.compileAndLint // Brás Gabín +1
22     @Language( value = "kotlin" ) content: String,
23     compilerResources: FakeCompilerResources(),
24 ): List<Lint> {
25     r
26     analysis so you should use compileAndLintWithContext instead of cor
27 }
28 if (shouldCompileTestSnippets) {
29     KotlinScriptEngine.compile( code = content)
30 }
31 return lint( content, compilerResources )
32 }
33
34 fun Rule.lint( // Brás Gabín +1
35     @Language( value = "kotlin" ) content: String,
36     compilerResources: FakeCompilerResources()
37 ) {
38     ...
39 }
```

A popup menu is visible over the `compileAndLint` function name on line 21. The menu contains the following options:

- compileAndLint
- andLint
- lint

At the bottom of the popup, it says: "Press ⌘⇧O to show options popup".

Инструменты IDE: Change Signature

```
21 fun Rule.compileAndLint( @Brais Gabín +1 ✓
26     "${this.ruleName} requires full analysis so you should use compileAndLintWithContext instead of
27 }
28 if (shouldCompileTestSnippets) {
29     KotlinScriptEngine.compile( code = content)
30 }
31 return lint(content, compilerResources)
32 }
33 💡
34 {+} fun Rule.lint( @Brais Gabín +1
35     @Language( value = "kotlin") content: String,
36     compilerResources: CompilerResources = FakeCompilerResources(),
37 ): List<Finding> {
38     require( value = this !is RequiresFullAnalysis) {
39         "${this.ruleName} requires full analysis so you should use lintWithContext instead of lint"
40     }
41     val ktFile = compileContentForTest(content)
42     return visitFile( root = ktFile, compilerResources = compilerResources).filterSuppressed( rule = this)
43 }
44
45 {+} fun Rule.lintWithContext( @Brais Gabín +3
46     environment: KotlinCoreEnvironment,
47     @Language( value = "kotlin") content: String
```

Инструменты IDE: Safe Delete

```
2
3 > import ...
17
18 private val shouldCompileTestSnippets: Boolean = 2 Usages
19     System.getProperty( key = "compile-test-snippets", def = "false")!!.toBoolean()
20
21 {+} fun Rule.compileAndLint( @ Brais Gabín +1
22     @Language( value = "kotlin") content: String,
23     compilerResources: CompilerResources = FakeCompilerResources(),
24 ): List<Finding> {
25     require( value = this !is RequiresFullAnalysis) {
26         "${this.ruleName} requires full analysis so you should use compileAndLintWithContext instead of
27     }
28     if (shouldCompileTestSnippets) {
29         KotlinScriptEngine.compile( code = content)
30     }
31     return lint(content, compilerResources)
32 }
33
34 {+} fun Rule.lint( @ Brais Gabín +1
35     @Language( value = "kotlin") content: String,
36     compilerResources: CompilerResources = FakeCompilerResources(),
```

Человек с IDE: результаты

Полнота

✓ все 1506 вызовов

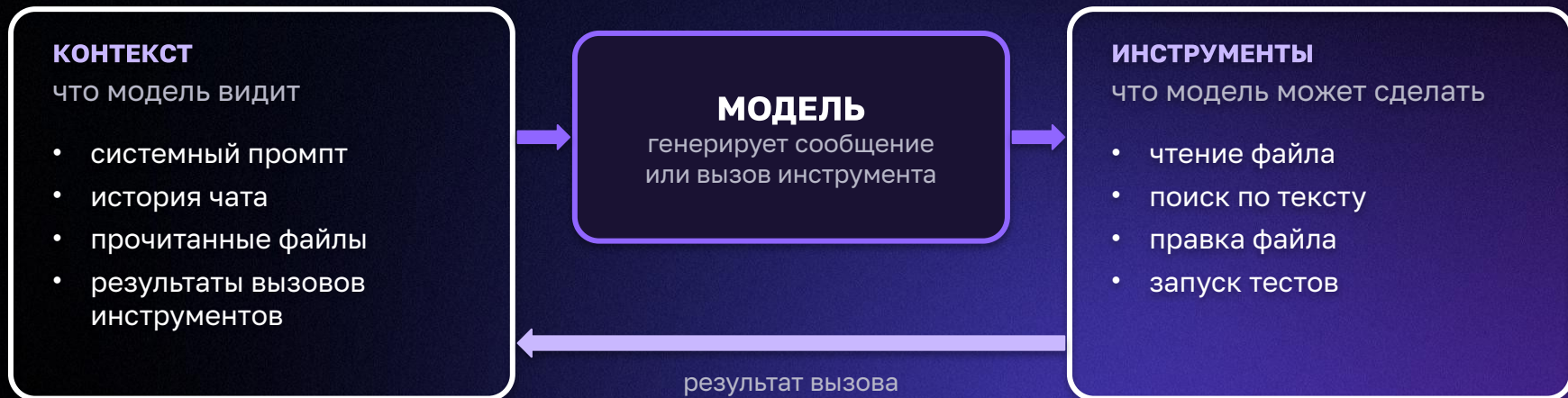
Корректность

✓ поведение не изменилось

Время
≈ 40 мин

Можно ли быстрее?

ИИ-агент



ИИ-агент = модель + контекст + инструменты

План доклада

01

Введение

02

Подходы к
рефакторингу в агентах

03

Итоги

Подходы к рефакторингу в агентах

1

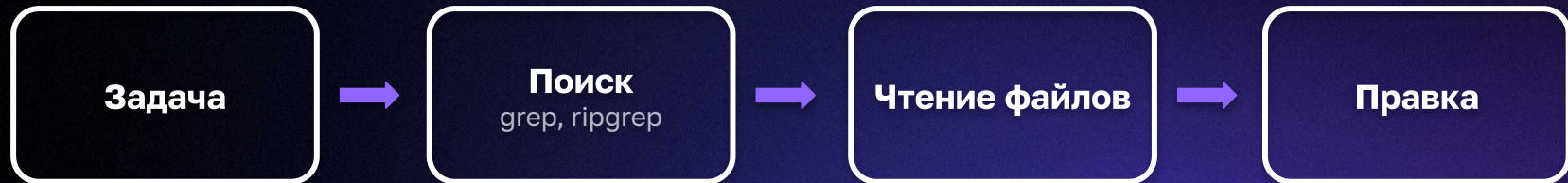
Текстовый подход

синтаксис

семантика

Подход 1: текстовый

Агент работает с кодом как с текстом: строки и подстроки, а не функции и вызовы



МЕХАНИЗМЫ ПРАВКИ

- Скрипт / Regex по файлам – заменяет подстроку везде, где нашёл
- Инструмент «Edit_File» – удалить / добавить / заменить; каждое место – отдельный вызов
- Перезапись файла целиком – модель генерирует новый файл

Подход 1: текстовый

Claude Code · Opus 5.5

```
code you can set .lint(code, compile = false) to specify that you don't want to verify the snippet.  
Existing tests must keep their current  
behavior. Leave lintWithContext / compileAndLintWithContext as they are.
```

```
Searched for 5 patterns, read 1 file, ran 1 shell command
```

- Still mapping call sites. Checking the two formatting specs that import the generic `lint`, then I'll edit.

```
Searched for 1 pattern, ran 1 shell command
```

- Now apply the edits with a script: add `compile = false` to string-based `lint(...)` calls (outside formatting), then rename `compileAndLint()` → `lint()`.

- Running `mkdir -p /tmp/lintscript && cat > /tmp/lintscript...`

```
L $ mkdir -p /tmp/lintscript && cat > /tmp/lintscript/fix.py <<'EOF'  
import re, sys, subprocess  
files = subprocess.check_output(["git","grep","-lE",r"\blint\(", "--", "*.kt"],text=True).split()  
skip_args = re.compile(r"^(ktFile\w*|file|ktsFile)$")  
changed = 0  
for f in files:  
    if f.startswith("detekt-for...")
```

* Ruminating... (54s · ↓ 4.3k tokens)

Скрипт дописывает `compile = false` к старым вызовам.
Упал – починил через `sed`

Подход 1: текстовый

Полнота

✓ все 1506 вызовов

Корректность

✗ сломал соседний модуль и комментарий

Время
6:20

Ошибок / Исправлено
6 / 3

Токены
1,8M

Стартовый контекст
49k

Плюсы и минусы

Плюсы

- Никаких зависимостей
- Скорость
- Не требователен к ресурсам
- Работает где угодно: CI, удалённая машина
- Любой язык и формат файла

Минусы

- Не учитывает семантику языка: перегрузки, переопределения, вызов с точкой и без точки, и т. д.
- Применение не проверяет результат: строка заменена – не значит, что поведение не изменилось
- Каждое место – отдельная операция, в которой можно ошибиться

Полнота и корректность

Одна строка, два разных контекста

```
detekt-rules-style/.../UnusedParameterSpec.kt
    subject.lint(code)    // наша detekt-test: параметр compile есть
detekt-sample-extensions/.../TooManyFunctionsSpec.kt
    subject.lint(code)    // detekt-test 1.23.7 из Maven: параметра compile нет
```

Полнота

Поиск по подстроке: пропускает нужные места и захватывает лишние

Корректность

После замены вызов может вести в другую функцию – замена этого не проверяет

Сравнение подходов

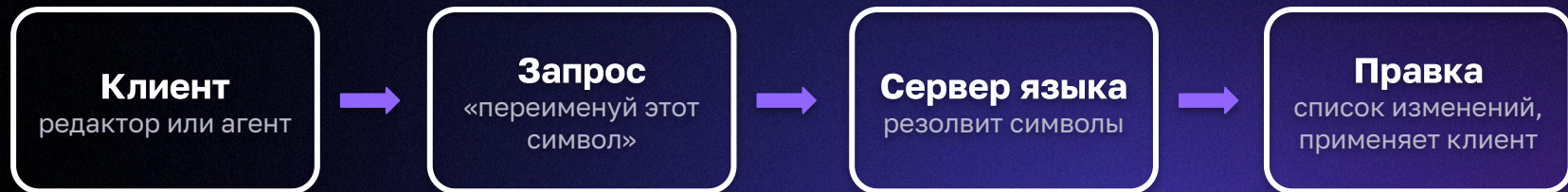
	Человек с IDE	Агент + текстовый подход
Полнота	✓	✓
Корректность	✓	✗
Время	≈ 40 мин	6:20
Ошибок / Исправлено	–	6 / 3
Токены	–	1,8М
Стартовый контекст	–	49k

Подходы к рефакторингу в агентах



Подход 2: LSP

LSP (Language Server Protocol) – протокол между редактором / агентом и сервером языка
Сервер – это отдельный процесс, который разбирает код и знает, к какому символу что относится
Один сервер на один язык



Что это даёт для рефакторинга

- Переход к определению, поиск использований, переименование по символу
- Серверы есть для всех основных языков: rust-analyzer, gopls, pyright; Kotlin LSP – пока экспериментальный

Подход 2: LSP

Claude Code + Serena (LSP) · Opus 5.5

What to do is to rename `compileAndLint` to `lint` so the function is always the same AND if you in one case doesn't want to compile the snippet of code you can set `.lint(code, compile = false)` to specify that you don't want to verify the snippet. Existing tests must keep their current behavior. Leave `lintWithContext` / `compileAndLintWithContext` as they are.

- I'll start by loading Serena's instructions, then locate the `lint/compileAndLint` definitions.

Searched for 2 patterns, read 1 file, called serena, ran 1 shell command

- I'll merge the methods into a single `Rule.lint(content, compilerResources, compile = true)`, updating existing `lint(String)` calls to pass `compile = false` and `compileAndLint(...)` calls to use the new `lint(...)` signature. First, let me survey the call sites.

Searched for 2 patterns, ran 1 shell command

- Many `lint(...)` calls take a `KtFile` (that overload stays). To separate String-based calls reliably, I'll use Serena's reference-aware rename on the String overload to a temporary name.

- Calling serena...

* Cultivating... (37s · ↓ 2.3k tokens · thought for 3s)

Агент сам объясняет, зачем LSP:
grep не отличит `lint(KtFile)` от `lint(String)`

Подход 2: LSP

Полнота

✓ все 1506 вызовов

Корректность

✓ поведение не изменилось

Время
4:38

Ошибок / Исправлено
3 / 3

Токены
2,4M

Стартовый контекст
59k

Плюсы и минусы

Плюсы

- Нужен только сервер языка
- Скорость
- Не требователен к ресурсам
- Работает где угодно: CI, удалённая машина

Минусы

- Из рефакторингов – только переименование
- Строгие рамки протокола LSP
- На каждый язык нужен отдельный сервер
- Не учитывает межязыковые связи: например, Kotlin - Java

Сравнение подходов

	Человек с IDE	Агент + текстовый подход	Агент + LSP
Полнота	✓	✓	✓
Корректность	✓	✗	✓
Время	≈ 40 мин	6:20	4:38
Ошибок / Исправлено	–	6 / 3	3 / 3
Токены	–	1,8М	2,4М
Стартовый контекст	–	49k	59k

Подходы к рефакторингу в агентах



Подход 3: движок рефакторинга IDE (IntelliJ)

Инструменты IDE позволяют разделить: модель решает «что», движок – «как»

Код представлен не текстом, а графом символов: символ – узел, ссылка – ребро к объявлению

КАК ВИДИТ GREP

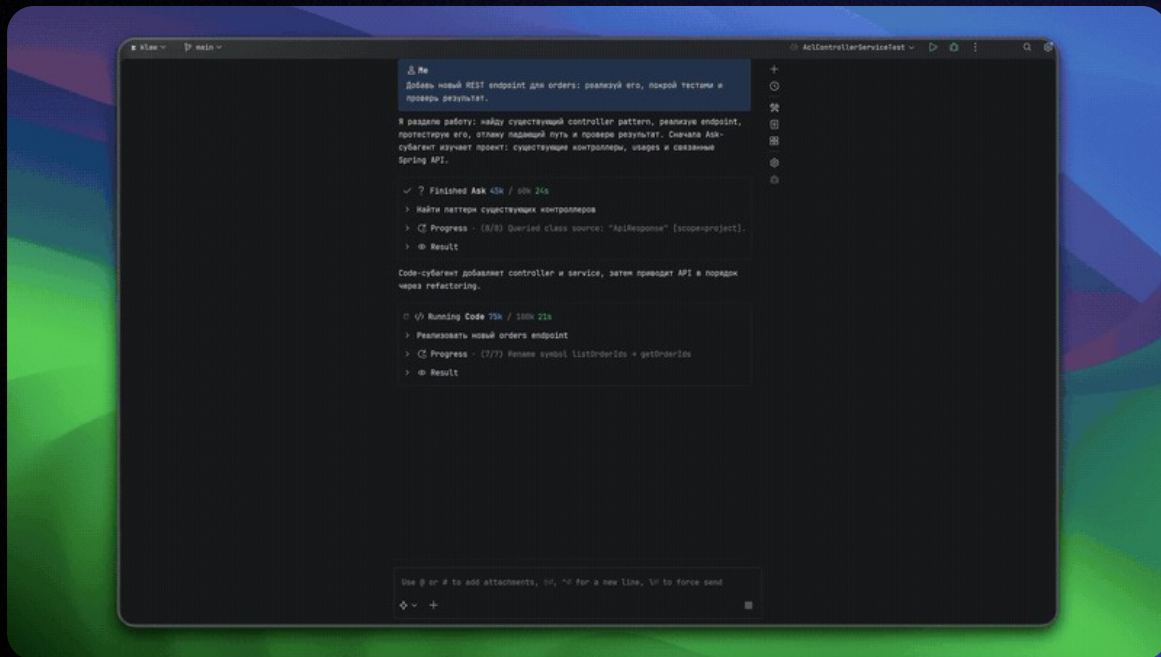
```
grep '\.lint(' → 180 строк:  
  rule.lint(code)      Rule.lint  
  subject.lint(code)   FormattingRule.lint  
  rule.lint(ktfFile)   перегрузка  
  lint("fun f() ...")  не найдено
```

КАК ВИДИТ IDE

```
Rule.lint(content)  99 ссылок  
Rule.lint(ktfFile)  другой символ  
FormattingRule.lint другой символ
```

PSI (Program Structure Interface) – модель кода: дерево + ссылки между узлами
IDE держит её в памяти и синхронизирует с редактором

Veai агент в JetBrains IDE



ИНСТРУМЕНТЫ IDE

- Find usages
- Rename
- Move
- Safe delete
- Отладчик
- Инспекции IDE
- ...

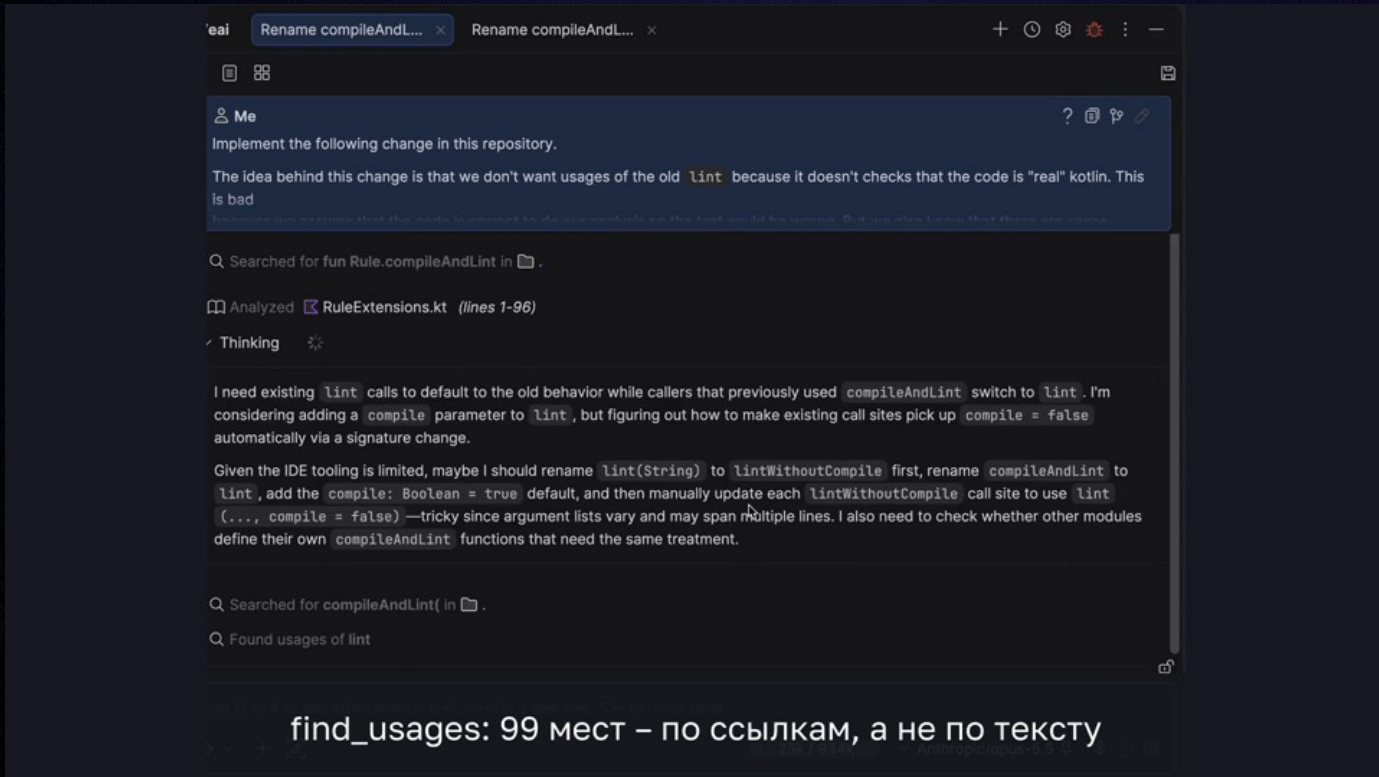
ИНСТРУМЕНТЫ АНАЛИЗА

- Поиск уязвимостей (SAST)
- Анализ на моргающие тесты
- Создание тестов из исполнения
- ...

Veai = ИИ + инструменты IDE + инструменты анализа программ

Подход 3: движок рефакторинга IDE

Veai · Opus 5.5



Подход 3: движок рефакторинга IDE

Полнота

✓ все 1506 вызовов

Корректность

✓ поведение не изменилось

Время
4:54

Ошибок / Исправлено
2 / 2

Токены
3,3М

Стартовый контекст
20k

Плюсы и минусы

Плюсы

- Полнота: находит все ссылки, включая другие языки и конфигурации
- Корректность: проверяет конфликты до применения
- Атомарность: одна операция рефакторинга, один Ctrl + Z

Минусы

- Нужна запущенная IDE
- Индексация в IDE – не самая быстрая операция
- Применение правки – блокировка на запись
- Проблематично с CI и удалёнными машинами
- Есть ограничения: например, Runtime-рефлексия

PSI: как IDE видит код

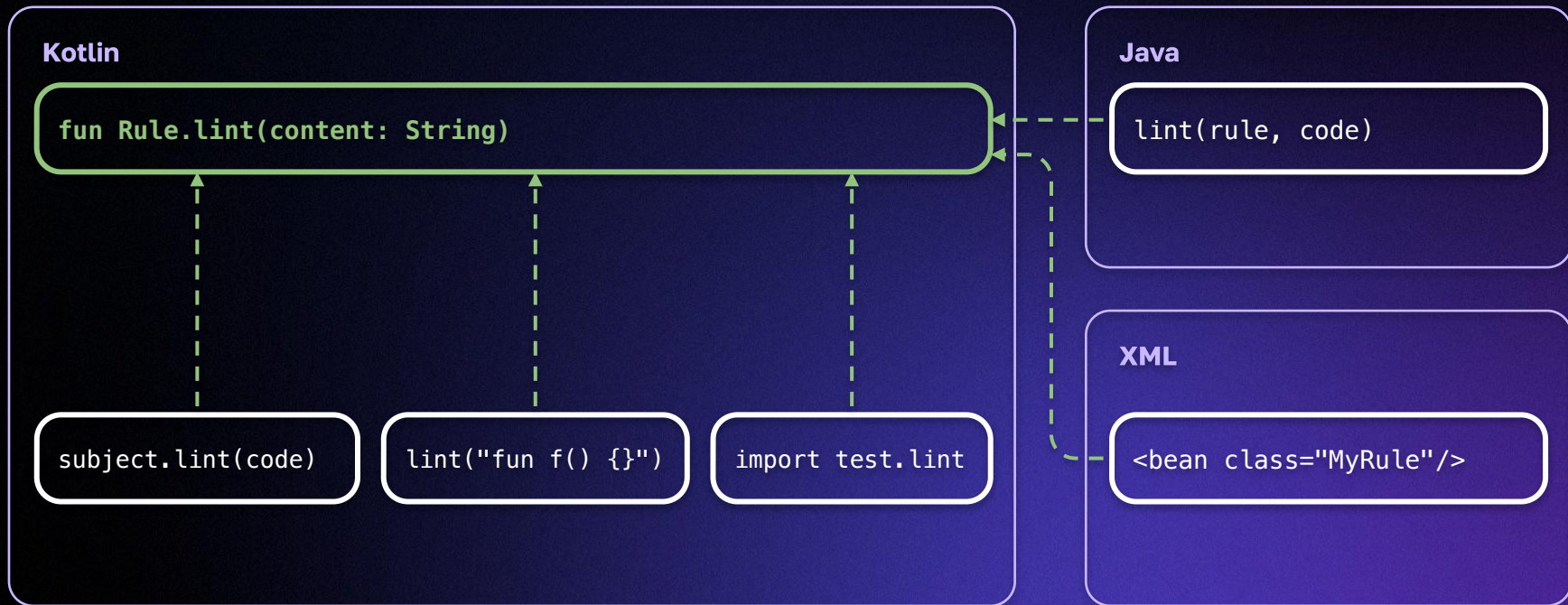
```
2210 private fun BracesOnIfStatements.test(code: String, vararg locations: Pair<Int, Int>) {  
2211     // This creates a 10 character prefix (signature/9, space/1) for every code example.  
2212     // Note: not compileAndLint for performance reasons, compilation is in a separate test.  
2213     val findings = Lint(content = "fun f() { $code }")  
2214     // Offset text locations by the above prefix, it results in 0-indexed locations.  
2215     val offset = 10  
2216     assertThat(findings)
```

PSI Tree

- ▼ BLOCK
 - PsiElement(LBRACE)
 - PsiComment(EOL_COMMENT)
 - PsiComment(EOL_COMMENT)
- ▼ PROPERTY
 - PsiElement(val)
 - PsiElement(IDENTIFIER)
 - PsiElement(EQ)
- ▼ CALL_EXPRESSION
 - ▼ REFERENCE_EXPRESSION
 - PsiElement(IDENTIFIER)
 - ▼ VALUE_ARGUMENT_LIST
 - PsiElement(LPAR)
 - ▼ VALUE_ARGUMENT
 - ▼ STRING_TEMPLATE

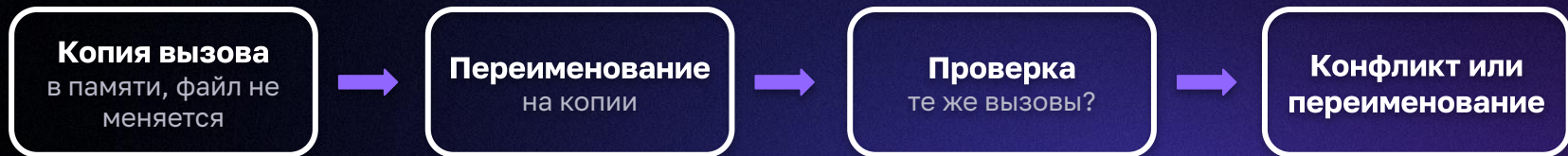
RuleExtensions.kt
fun Rule.lint(content, ...)

Полнота



Корректность

Правка примеряется на копии в памяти: конфликты видны до того, как изменен файл



```
rule.lint(code)           // переименовано  
rule.lint(code)           // неразличимы: 78 вызовов меняют смысл
```

Движок видит это на копии и останавливается: конфликт

Сравнение подходов

	Человек с IDE	Агент + текстовый подход	Агент + LSP	Агент + инструменты IDE
Полнота	✓	✓	✓	✓
Корректность	✓	✗	✓	✓
Время	≈ 40 мин	6:20	4:38	4:54
Ошибок / Исправлено	–	6 / 3	3 / 3	2 / 2
Токены	–	1,8М	2,4М	3,3М
Стартовый контекст	–	49k	59k	20k

Сравнение подходов

Текст

Claude Code

15

задач

15

решено

5

без ошибок

9

попыток начать заново

LSP

Claude Code + Serena

15

задач

10

решено

7

без ошибок

10

попыток начать заново

Инструменты IDE

Veai

15

задач

15

решено

15

без ошибок

0

попыток начать заново



Дали агенту
классные
инструменты, а он
ими не пользуется...
Почему?

Bias агентов

Codex CLI – системный промпт

- When searching for text or files, prefer using `rg` or `rg --files` respectively because `rg` is much faster than alternatives like `grep`. (If the `rg` command is not found, then use alternatives.)
- Try to use `apply_patch` for single file edits, but it is fine to explore other options to make the edit if it does not work well. Do not use `apply_patch` for changes that are auto-generated (i.e. generating `package.json` or running a lint or format command like `gofmt`) or when scripting is more efficient (such as search and replacing a string across a codebase).

«Для поиска используй `rg`,
для редактирования –
скрипты»

Claude Code – системный промпт

While bypass permissions mode is active:

Do your work through the Bash tool wherever it can accomplish the job: read files with `cat`, `head`, or `sed -n`, search with `grep` and `find`, and make file changes with `sed`, heredocs, or short scripts, rather than using the dedicated Read, Edit, or Write tools. Fall back to a dedicated tool only when Bash genuinely cannot do the job.

«Работай через `bash`: для
чтения – `cat`, для поиска –
`grep`, для редактирования –
`sed`»

Bias агентов

Форум Cursor, июль 2026 – ответы сотрудников Cursor



Colin Community Support Engineer

Jul 22

Hey all! Let me try to answer your questions.



aithal:



So, the absence of codebase Indexing in the recent update is expected?

Yes. Semantic/embeddings indexing is being turned down in favor of grep-based retrieval.

«Семантический индекс (RAG) отключён в пользу grep»



Maxime maximep

Jul 21

@Alphayellowcat thanks for the question, you are correct, we recently made some changes to how agents search the codebase. Over the last months, models have gotten really good at using grep / indexed search, so the older dedicated semantic search path was rapidly no longer helping in a meaningful way.

«Модели научились искать с помощью grep, индекс (RAG) не нужен»

Bias агентов

Документация Serena – MCP-сервера с инструментами LSP для агентов, октябрь 2026

! Attention

Recent updates to Claude Code (CC) and to the Opus line of models resulted in drastically reduced adherence to instructions pertaining to Serena's tools.

After extensive analysis, we identified part of the reason to be very long and detailed tool descriptions for built-in tools and parts of the default system prompt. The descriptions of CC's system tools take almost 16k tokens, cannot be adjusted by the user, and introduce a very strong bias towards internal tools, making it almost impossible to convince Opus 4.7 to use Serena.

As a workaround, we crafted a system prompt that counteracts this bias. When using Serena, we highly recommend that you start CC as

```
claude --system-prompt="$(serena prompts print-cc-system-prompt-override)"
```

You can also consider adding the content of `serena prompts print-cc-system-prompt-override` to your `CLAUDE.md` files, but the effect may be insufficient for counteracting Claude Code's bias towards internal tools.

**«Почти невозможно
убедить Opus 4.7
пользоваться Serena»**

Авторы Serena дают отдельный системный промпт, чтобы это исправить

Bias моделей

Наш эксперимент, октябрь 2026

Доля задач, где агент пользовался инструментами IDE

	Sol 6.1	Opus 5.5	GLM 5.3
Поиск в коде	53 %	0 %	13 %
Рефакторинг	67 %	47 %	40 %

Эксперимент AgentConnect, август 2026

Доля задач, где агент выбрал LSP

Если LSP навязать – успешных задач 89 % вместо 100 %

	Opus 4.8	Sonnet 4.6	Haiku 4.5
Поиск в коде	0 %	4 %	6 %
Рефакторинг	3 %	–	–

agentconnect.md/blog/grep-beat-lsp-harness

План доклада

01

Введение

02

Подходы к
рефакторингу в агентах

03

Итоги

Что выбрать

Текст

Нужна легковесность; нет IDE: CI, удаленная машина; хорошее тестовое покрытие, guardrails

LSP

Большой проект, в котором агент путается; нужна легковесность; нет IDE: CI, удаленная машина; нет межъязыковых связей;

Движок IDE

Большой проект, в котором агент путается; нужны гарантии; есть IDE и соответствующий компьютер

Veai CLI

```
🚀 Veai AI Agent v0.3.2  
esc interrupt · ctrl+c/ctrl+d clear/exit · / commands · ! bash
```

```
✓ New session started
```

```
Найди и исправь падающие юнит тесты
```

```
✦ mcp 0/0 • veai/auto • medium • 0%/1.0M
```

«Русский Claude Code»

- Все топовые модели в одной подписке
- Работает без VPN
- Оплата русской картой
- Джентльменский набор: субагенты, правила, скиллы, MCP, ...
- Управление из Telegram

veai.ru/cli

Главные мысли

1

**Рефакторинг –
преобразование с
гарантиями**

Полнота при поиске,
корректность при
применении

2

**Автоматическая
проверка –
обязательна**

Хорошие тесты, линтер,
скрипты проверок, ...
Агент, как и мы, допускает
ошибки

3

**Инструменты – это
руки и ноги агента**

Выбирайте инструменты в
зависимости от того, что
вам важно
От них сильно зависит
поведение агента и
качество результата

Спасибо за внимание

Вопросы?

Попробовать агента: veai.ru

Попробовать CLI-агента: veai.ru/cli

Промокод «Heisenbug 2026» – бесплатные токены: app.veai.ru

Telegram-канал: [@veai_devs](https://t.me/veai_devs)

MCP-серверы для рефакторинга

MCP (Model Context Protocol) – протокол общения LLM с инструментами

MCP-сервер – сервер, отвечающий на запросы LLM по MCP

MCP-СЕРВЕР INTELIJ

Встроен в IDE с версии 2025.2, включается в настройках

- Rename – переименование символа
- Move – нет
- Safe Delete – нет

MCP-СЕРВЕР VEA1

Встроен в агента с версии Veai 5.17, бесплатно

- Rename – переименование символа
- Move – перенос класса, функции или пакета с обновлением ссылок
- Safe Delete – удаление с проверкой использований