Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 24.М71-ММ

Подсистема CRC Python IDE для анализа и ассистирования модификаций кода

Кузнецов Илья Александрович

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
профессор кафедры системного программирования, д.т.н. Д. В. Кознов

Консультант:
ведущий инженер ключевых проектов ООО «Техкомпания Хуавэй» Н. В. Тропин

Санкт-Петербург
2026

Saint Petersburg State University

System Programming's chair

Group 24.M71-MM

Ilya Alexandrovich Kuznetsov

Code Analysis, Modification and Assistance in CRC Python IDE

Internship report
in a «Production» form

Scientific supervisor:
professor System Programming's chair, D.E.Sc. D. V. Koznov

Consultant:
principal engineer at «Huawei Technologies CO.LTD» N. V. Tropin

Saint Petersburg
2026

Оглавление

Введение

Средства разработки — редакторы кода, статические анализаторы, среды разработки (IDE), ИИ-помощники (ADE) и системы автоматизации разработки (CI/CD) — давно превратились из вспомогательных утилит в ключевой инструмент инженерной деятельности. Они определяют рабочие процессы разработчиков, ускоряют поиск ошибок и напрямую влияют на стоимость сопровождения программного обеспечения [1]. Исследования показывают, что все больше индивидуальных специалистов, исследователей и продуктовых команд выбирают продвинутые средства разработки, существенно полагаясь на анализ кода, исправления и рефакторинги, автодополнение и подсказки, интеграцию с системами тестирования и CI/CD, фреймворками и большими языковыми моделями (LLM) [2].

Коммерческий рынок средств разработки демонстрирует устойчивый рост, и многие технологические компании рассматривают их как стратегический актив, обеспечивающий независимость, качество и конкурентоспособность продуктов [3]. Появление LLM-ассистентов и их интеграция с инструментами разработки изменили рынок: такие решения позволяют автоматизировать рутинные задачи, повысить продуктивность и ускорить выпуск продуктов. Однако в исследованиях все чаще отмечают риски использования LLM для генерации кода без последующей верификации, что делает необходимым внедрение встроенных механизмов анализа и контроля качества кода [4][5].

Язык программирования Python остается одним из наиболее популярных на протяжении многих лет и широко используется в сферах от веб-разработки до анализа данных и машинного обучения [6]. Однако, его динамическая семантика затрудняет статический анализ кода и усложняет сопровождение крупных проектов. Для разработки на Python сформировался широкий набор решений: статические анализаторы (Муру [7], Pyright [8] и др.), среды разработки (PyCharm [9], VSCode [10] и др.), а также ИИ-помощники на основе LLM (Cursor [11], Copilot [12] и др.). Данные системы предоставляют инструменты для

анализа, исправления и рефакторинга кода, навигации по проекту, а также интеллектуальных ассистентов для подсказок, автодополнения и генерации кода, что делает их важным элементом современной экосистемы разработки на Python [2].

Несмотря на зрелость существующих решений, важной задачей остается создание инструментов, способных обеспечить высокое качество статического анализа и ассистирования модификаций кода. Одной из перспективных сред разработки для Python является CRC Python IDE, которая основывается на собственной модели кода, языковом сервере и клиенте, совместимых с протоколом LSP [10]. В рамках данной работы, в CRC Python IDE будет спроектирована и внедрена подсистема, предоставляющая инструменты для статического анализа, исправления и рефакторинга кода, а также информационных и LLM-ассистентов. Для обеспечения конкурентоспособного уровня качества будут определены основные преимущества разработанной подсистемы, а также будет проведено разностороннее тестирование и апробация решения на основе отзывов пользователей.

1. Постановка цели и задач

Целью выпускной квалификационной работы является проектирование и реализация подсистемы в существующей среде разработки для языка Python (CRC Python IDE), предоставляющей инструменты (Features) для анализа и ассистирования модификаций кода.

Для достижения цели выпускной квалификационной работы были поставлены следующие задачи.

1. Провести обзор инструментов для анализа и ассистирования модификаций кода в существующих средствах разработки для языка Python.

2. Выполнить проектирование подсистемы, определив группы инструментов (Features), предоставляемые пользователям, а также изменения существующей модели кода CRC Python IDE, необходимые для реализации этих инструментов.

3. Выполнить реализацию спроектированной подсистемы и интегрировать ее в CRC Python IDE.

4. Провести разностороннее тестирование подсистемы, модернизировав существующую тестовую инфраструктуру в CRC Python IDE.

5. Провести апробацию и доработку подсистемы на основе обратной связи от пользователей CRC Python IDE.

2. Обзор предметной области

В данной главе будут рассмотрены инструменты разработки для Python, их возможности и модели кода, с помощью которых реализуется данная функциональность.

2.1. Инструменты разработки для Python

Для языка Python существует множество различных инструментов разработки, наиболее популярными являются статические анализаторы и среды разработки.

- Статические анализаторы – MyPy и PyRight – предлагают долгий, но наиболее точный анализ кода на Python. Их можно интегрировать в IDE, и они часто используются для оценки качества кода.
- Среды разработки – SRC IDE, PyCharm и VSCode (Pylance) – кроме анализа кода, который должен быть быстрым и как можно более точным, предлагают исправления проблем и действия по реструктуризации кода, а также ассистируют при написании кода на Python. И это лишь базовая часть их возможностей, поэтому они часто используются в продуктовой разработке.

2.2. Возможности инструментов разработки для Python

Теперь рассмотрим некоторые возможности, которые предоставляют инструменты разработки.

- Анализы (в PyCharm – инспекции, в VSCode – диагностики) – обнаруживают и интерпретируют проблемы в коде, определяя исправления.
- Исправления (фиксы) – это действия по модификации кода для устранения проблем, найденных инспекциями.

- Реструктуризации (рефакторинги) – это действия по модификации кода для изменения его структуры, абстракции или упрощения.

- Ассистенты (помощники) – представляют информацию о коде в удобном виде, упрощая его понимание и снижая вероятность ошибок.

Производительность и точность возможностей инструментов разработки напрямую зависят от модели кода. А их качество и полнота определяется соответствием спецификациям Python и библиотек, покрытием различных сценариев как в правильном, так и в неправильном коде, а также проработкой пользовательского опыта и дизайном.

2.3. SRC IDE

SRC IDE — это мультиязыковая среда разработки, первоначально создававшаяся для внутренних нужд известной технологической компании, но подлежащая выпуску в виде продукта в будущем. На момент начала работы она поддерживала работу только с одним языком — Java. Но теперь на основе существующего решения предстоит создание семейства продуктов, чтобы обеспечить поддержку Python в обновлённой среде разработки. Её разработку планируется выполнять в общем технологическом процессе с решением для языка Java и путём создания повторно используемой инфраструктуры в текущей архитектуре, с сохранением ключевых технологий проекта. Перейдём к их рассмотрению.

2.3.1. Архитектура

Архитектурно SRC IDE состоит из двух основных компонентов: языкового сервера (Language Server) и клиентской части на основе Visual Studio Code [**?**] с собственным пользовательским интерфейсом, соединённых по протоколу языкового сервера (Language Server Protocol) [**?**], основанном на технологии JSON-RPC. Возможности IDE могут быть

гибко изменены при помощи расширений (Extension) для клиентской части IDE. В составе таких расширений могут подключаться дополнительные языковые сервера.

Языковой сервер основывается на кодовой модели, которой делегирует обработку пользовательского кода в виде проектов (Project). Многие компоненты языкового сервера основываются на интерфейсах кодовой модели для обеспечения как базовых, так и продвинутых возможностей IDE. К таким можно отнести: структуру проекта (Project Structure), структуру файла (File Structure), навигацию по коду (Navigation), текстовый поиск (Text Search), поиск использований кода (Find Usages), всплывающую информация о коде (Hover), информацию о сигнатурах (Signature Help), рефакторинг кода (Refactoring), быстрые исправления (Quick Fixes), проверки кода (Inspection), автодополнение (Completion) и другие. Пример работы языкового сервера по протоколу LSP изображён на рисунке 1.

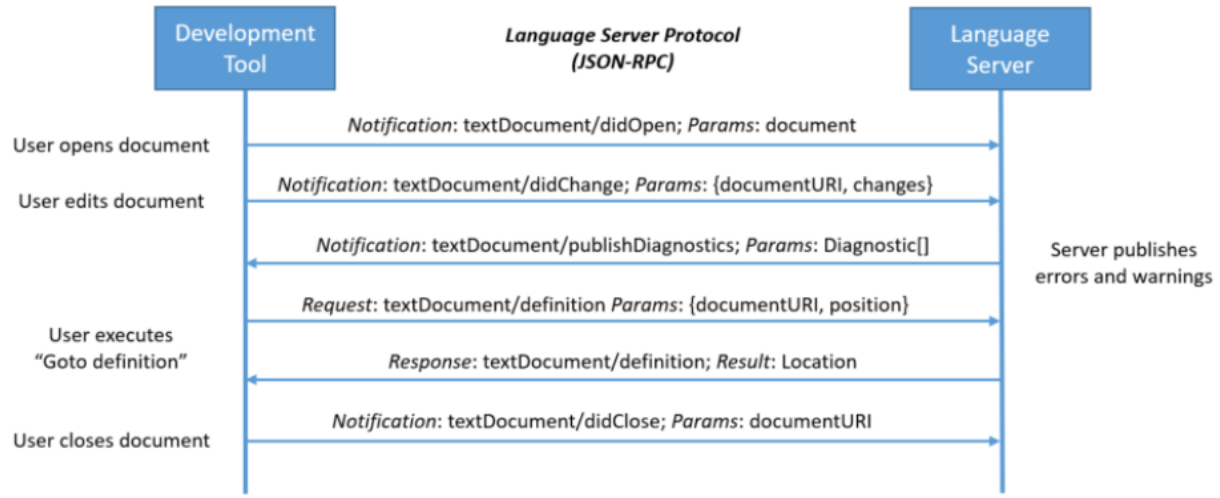

Рис. 1: Диаграмма последовательности сценария работы LS по протоколу LSP [?]

Кодовая модель отражает детальное представление пользовательского проекта в IDE с учетом структуризации кода, задаваемой языком программирования, пакетным менеджером или системой сборки (пакеты, модули). Она осуществляет разбор исходного кода и генерацию синтаксических и семантических деревьев разной степени детализации,

а также предоставляет возможности их модификации. На рисунке 2 на уровне компонентов разбирается устройство SRC IDE для языка Java.

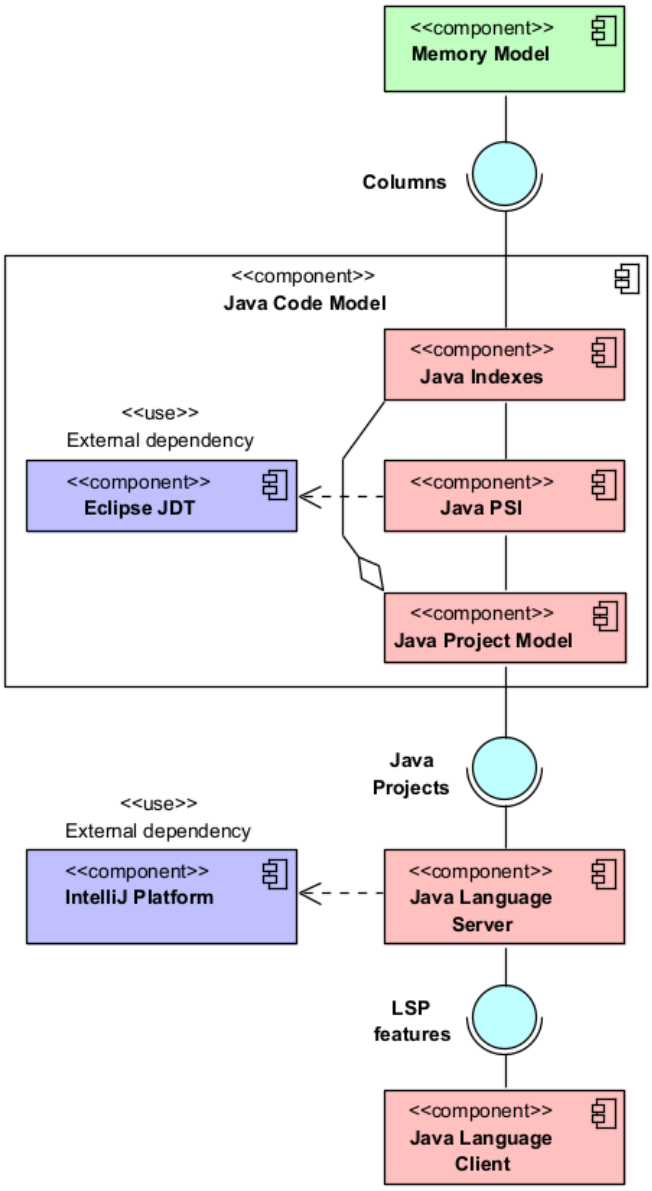

Рис. 2: Диаграмма компонентов UML для существующего решения для Java в SRC IDE

За построение индексов (Indexes) в SRC IDE отвечает собственная модель памяти (Memory Model), представленная в виде системы сжатия объектов в памяти (Object Compression System). Она предоставляет интерфейсы отображений (Mapping), которые используются для компактного хранения и быстрого доступа к классам, необходимым для работы большинства возможностей среды разработки.

Рассмотрим подробнее основные компоненты кодовой модели SRC IDE.

2.3.2. Проектная модель

Модель кода включает в себя проектную модель (Project Model). Это высокоуровневый компонент системы, отвечающий за структуру проекта и соответствующих ему зависимостей с точки зрения файловой системы, обеспечивающий конфигурирование и интеграцию с различными пакетными менеджерами и системами сборки.

В нём строится представление всего проекта в виде древовидной иерархии. Так, в отдельных логических контейнерах хранятся файлы с пользовательским кодом и зависимостями, а также их содержимое, которые распределены в структуре по соответствующим модулям и пакетам. На уровне проекта происходит объединение с возможностями других подсистем кодовой модели. Таким образом, интерфейсы проектной модели являются основными для потребления остальными частями SRC IDE. При этом хранение значительной части данных происходит в индексах.

2.3.3. Синтаксическая структура

Данный компонент позволяет по грамматике языка строить абстрактные синтаксические деревья (AST) разной степени детализации, то есть преобразовывать конкретный синтаксис в удобную для дальнейшей обработки форму. Таким образом, здесь происходит разбор файлов с кодом, содержащихся в проекте, и предоставляется доступ к соответствующим синтаксическим деревьям в виде индексов.

2.3.4. Семантическая структура

Данная подсистема осуществляет семантический анализ построенных синтаксических деревьев обнаруженных файлов с кодом. Здесь центральной сущностью является символ (Symbol) — высокоуровневый

элемент (Member) языка, являющийся абстракцией над узлами дерева. С его помощью можно устанавливать семантические связи между различными файлами с кодом, обрабатывать их зависимости, а также осуществлять разрешение простых имён (Simple Resolve) и квалифицированных имён (Qualified Resolve). Для этого также вводится такое понятие как пространство имён (Namespace), отвечающее за доступность символов в конкретном контексте, на основе которого задаётся область видимости. Доступ к информации также осуществляется при помощи индексов.

2.3.5. Модифицируемая структура

Данная подсистема отвечает за модификацию деревьев в ответ на изменения в коде и сильно связана с предыдущими двумя компонентами. Более подробно она описана в работе [**?**].

2.3.6. Модель памяти

В SRC IDE для управления памятью и её оптимизации, хранения данных в индексах и на диске используется система сжатия объектов. Её особенностью является компактное (сжатое) представление объектов, которое не только снижает необходимое для функционирования среды разработки количество оперативной памяти, но и обеспечивает прирост производительности благодаря увеличению количества объектов, помещающихся в кэше процессора. Таким образом, максимизируется число кэш-попаданий, за счёт чего обращаться к оперативной памяти приходится реже.

Модель памяти позволяет упаковать объекты и их поля в колонки (Column). С точки зрения времени компиляции, разницы между сохраняемыми и возвращаемыми объектами из системы сжатия не существует. Однако во время исполнения модель памяти на основе сохранённых объектов генерирует прокси-классы с точно такими же программными интерфейсами. Таким образом, при доступе к системе возвращается компактный объект прокси-класса, соответствующий исходному клас-

су. Также для работы с некоторыми сжимаемыми объектами и управления их жизненным циклом применяется технология JavaBeans.

Помимо указанных преимуществ, система сжатия имеет ограничения: граф сохраняемых объектов должен заканчиваться исключительно примитивными типами, не допускаются циклы (из-за чего иногда приходится создавать обёртки над классами), также устанавливаются определённые правила именования элементов сохраняемых классов (для внутренней работы с рефлексией). Для конфигурирования системы сжатия и упаковываемых объектов предоставляется API с аннотациями.

Для построения индексов модель памяти предоставляет гибкий интерфейс с отображениями (Mapping) разных видов: один-к-одному (OneToOne) и один-ко-многим (OneToMany). Они определяют количество значений, которые могут соответствовать одному ключу. В дополнение, предоставляются оптимизированные для работы IDE реализации отображений.

2.3.7. Индексы

Индексы являются основным компонентом системы для сохранения данных в модели памяти SRC IDE и последующего доступа к ним. Они основываются на колонках модели памяти. Остальные подсистемы как кодовой модели, так и других частей среды разработки используют существующие или собственные индексы.

Индексы образуют ациклический граф, задающий зависимость по данным одних индексов от других. Это важно для их корректного построения и обновления в ответ на поступающие изменения в виде разницы в данных (Delta). Также благодаря им осуществляется кэширование часто используемых данных, сохранение на диск и загрузка с него.

2.4. IntelliJ Platform

IntelliJ IDEA от компании JetBrains является одной из ведущих IDE, получившей немалую популярность среди разработчиков. Важным пре-

имуществом является наличие поддержки в одном продукте не только JVM-ориентированных языков Java, Koltin и Scala, но и множества других, поставляемых в виде плагинов (Plugin). К таким можно отнести Python, Ruby, TypeScript и прочие. Альтернативным предложением компании являются производные продукты из их семейства IDE, специализированные для разработки с определённым набором технологий: PyCharm, RubyMine, WebStorm и другие [**?**]. Такое разнообразие решений возможно поддерживать благодаря грамотно созданной повторно используемой инфраструктуре, образующей мультиязыковую IntelliJ Platform [**?**].

2.4.1. Архитектура

IntelliJ Platform обладает модульной архитектурой, скрывающей от разработчиков детали реализации кодовой модели и предоставляющей им набор инструментов (SDK) для расширения возможностей платформы путём добавления плагинов. Возможные точки расширения (Extension Points) определяются спецификацией платформы. Различные продукты компании собираются из платформы и включенных в поставку расширений.

Для поддержания модульной структуры, платформа разделяется на иерархию компонентов разного уровня: всего приложения (Application), проекта (Project) и модуля (Module), жизненным циклом которых управляют контейнеры, поддерживающие инъекцию зависимостей (Dependency Injection). Помимо этого, для оптимизации производительности существуют сервисы (Service), поддерживающие ленивую инициализацию.

2.4.2. Виртуальная файловая система

Работа с нативной файловой системой организована через абстракцию — виртуальную файловую систему (Virtual File System). Это позволяет платформе работать с различными файловыми системами по одному интерфейсу, обрабатывать изменения от них, а свойство перси-

стентности, за счёт сохранения снимка (Snapshot) всех файлов проекта, решает проблемы с синхронизацией доступа.

2.4.3. Инфраструктура обмена сообщениями

Для взаимодействия различных компонентов платформы создана инфраструктура обмена сообщениями, основанная на паттерне «издатель-подписчик». Это позволяет определять иерархию распространения сообщений в системе и ограничивать круг их получателей. Таким образом, существуют специализированные каналы сообщений, называемые «топики» (Topic), к которым может подключиться подписчик и ожидать получение сигналов. Издатель может посылать сообщения в соответствующую шину (Bus), которые будут впоследствии распространены всем ожидающим подписчикам.

2.4.4. Проектная модель

Проектная модель IntelliJ Platform мультиязычна и расширяема. Она определяет общий интерфейс проекта, инкапсулирующего весь исходный код и поддерживающего разделение на модули, а также зависимые библиотеки и SDK. Для определения пользовательского кода применяются логические контент-руты (Content Root), позволяющие распознать, где в проекте лежит исходный код, код зависимостей и тестов, а какой следует исключить из рассмотрения. Они же влияют и на способ обработки кода в среде разработки. Также в проектной модели обеспечивается интеграция с различными системами сборки.

В связи с монолитной архитектурой IntelliJ Platform многие компоненты проектной и кодовой моделей содержат методы для реагирования на события как других подсистем, так и пользовательского интерфейса. Подобное взаимодействие происходит через инфраструктуру обмена сообщениями. Например, IntelliJ Platform предоставляет возможности для работы с пользовательским интерфейсом и используется в качестве клиентской части в среде разработки для .NET — Rider. При этом вся функциональность кодовой модели вынесена в отдельный

процесс языкового сервера на .NET, использующий собственный протокол RD (Reactive Distributed) для взаимодействия с платформой.

2.4.5. Кодовая модель

Роль модели кода в IntelliJ Platform играет мультиязыковой интерфейс программной структуры (PSI). Он отвечает за синтаксический анализ файлов и создание синтаксической и семантической модели кода в виде деревьев, обеспечивающей работу многих функций платформы. После обработки пользовательского кода, его элементы доступны через PSI-файлы как PSI-элементы, с которыми удобно работать для реализации продвинутых возможностей платформы. Необходимо отметить, что интерфейс программной структуры — модифицируемый. Изменения приходят от виртуальной файловой системы и обновляют синтаксические деревья, за чем следует обновление изменившейся информации в PSI. При этом для хранения данных и поддержания модели кода в производительном состоянии используются индексы.

2.4.6. Индексы

Быстрый доступ к хранимым данным платформы реализуется на основе индексов, которые строятся и поддерживаются в актуальном состоянии на протяжении всей работы платформы. Они основываются на произвольных отображениях ключей в значения и хранятся в компактном формате. Индексы активно применяются во всех компонентах системы, в том числе для обеспечения общего доступа к данным между ними.

2.5. Сравнение моделей кода

Исходные характеристики модели кода могут быть оценены различными метриками. В статьях часто встречается метрика — количество разрешённых квалифицированных ссылок, поскольку для этого задействуются многие подсистемы модели кода, например Resolve и Type Inference.

В таблице представлено сравнение с другими решениями на основе этой метрики на конец 2023 года, новые данные ещё собираются. SRC IDE обходила все существующие решения на популярных проектах, а Pylance даже не останавливался на некоторых из них. Таким образом, при даже при идентичной реализации, например, какого-нибудь анализа, ожидается, что в SRC IDE он будет точнее.

	Pylance (PyRight)	PyCharm	SRC IDE
Flask (5072 QR)	32.96 %	29.20 %	39.33 %
Jedi (10884 QR)	32.77 %	45.61 %	60.63 %
PyTorch (599914 QR)		54.29 %	61.36 %
Tensorflow (491977 QR)		65.96 %	84.09 %
Django (202578 QR)	28.75 %	59.16 %	76.08 %
Numpy (85004 QR)		61.51 %	87.64 %
ToolBench (2913 QR)	57.56 %	62.92 %	79.71 %
SciPy (123591 QR)		32.37 %	85.63 %
Scikit-learn (90175 QR)		33.78 %	69.52 %

Рис. 3: Сравнение разрешенных квалифицированных ссылок в моделях кода

Заключение

Для достижения цели выпускной квалификационной работы были получены следующие результаты.

- Рассмотрены соответствующие инструменты (Features) в статических анализаторах Mypy и Pyright, средах разработки PyCharm, VSCode и CRC Python IDE, а также в ИИ-помощниках Cursor и Copilot. Определены сильные и слабые стороны этих решений, а также базовые потребности пользователей при разработке программ на Python.

- Спроектирована подсистема среды разработки, для реализации выбрано 5 групп инструментов (Features): для статического анализа, исправления и рефакторинга кода, а также информационные и LLM-ассистенты. Спроектированы необходимые изменения модели кода: расширение и стабилизация синтаксического/семантического представления программы, уточнение/конфигурирование системы разрешения ссылок, а также предоставление общего интерфейса для вывода типов.

- Выполнена реализация спроектированной подсистемы: 40 инструментов для статического анализа (Syntax Correctness, Reference Resolve, Expected Type и др.), 18 инструментов для исправления кода (Install Package, Import Reference, Create Reference и др.) и 8 инструментов для рефакторинга кода (Extract, Inline, Rename и др.), а также 3 информационных ассистента (Hover, Signature Help и Inlay Hints) и 3 LLM-ассистента (AI Explain Code, AI Improve Code и AI Fix Code).

- Выполнена модернизация тестовой инфраструктуры (поддержка конфигурируемых тестовых проектов с фиксацией ожидаемых результатов для каждой группы инструментов), создано 3000 модульных тестов и 5 бенчмарков для регрессионного тестирования подсистемы, в качестве комплексного тестирования были настро-

ены автотесты с помощью фреймворка Selenium и применялся догфудинг (Dogfooding).

- Проведена апробация подсистемы на основе отзывов пользователей CRC Python IDE, что способствовало созданию 3 новых инструментов (Version Compatibility, Incorrect Formatting и Reformat) и исправлению 10 проблем (ложно-положительные/отрицательные срабатывания и др.).

Список литературы