

Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 23.Б10-мм

Реализация быстрых исправлений для IDE на базе Eclipse

СОТНИКОВ Илья Александрович

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
ст. преподаватель кафедры ИАС, Смирнов К. К.

Консультант:
старший программист-разработчик, ООО «СофтКом», Бабанов П. А.

Санкт-Петербург
2025

Оглавление

Введение	3
1. Постановка задачи	4
2. Обзор	5
2.1. Eclipse	5
2.2. CLDT и clang-tidy в его составе	6
2.3. Выбор типов маркеров для исправления	7
2.4. Обзор используемых технологий	7
2.5. Выбор инструмента для рефакторинга	7
2.6. Clang-refactor в составе IDE	9
2.7. Выводы	10
3. Реализация	11
3.1. Архитектура механизма быстрых исправлений	11
3.2. Проведение необходимых преобразований для реализации быстрых исправлений	12
3.3. Формирование класса с идентификаторами маркеров, поддерживающих исправлений	13
3.4. Реализация автоматических исправлений	14
3.5. Реализация исправления с переименованием	16
4. Тестирование и апробация	19
4.1. Тестирование	19
4.2. Апробация	20
Заключение	22
Список литературы	23

Введение

В современной разработке программного обеспечения ключевым аспектом является повышение качества кода и ускорение процесса его написания и отладки. Интегрированные среды разработки (IDE) играют важную роль в этом процессе, предоставляя разработчикам инструменты для эффективного взаимодействия с программой и более глубокого понимания кода. Одним из таких инструментов является рефакторинг, позволяющий улучшить читабельность кода, облегчить его поддержку и дальнейшее расширение, а иногда даже ускорить выполнение программы. На настоящее время рефакторинг широко используется в популярных IDE, например, для поддержки быстрых исправлений кода.

ООО «Софтком» занимается разработкой IDE на базе платформы Eclipse и ставит перед собой задачу улучшить процесс работы с кодом, сделав его более интуитивным и эффективным. В частности, в среде разработки уже интегрирован clang-tidy, генерирующий маркеры предупреждений и ошибок в нужных местах исходного кода программ на C/C++. Однако никаких подсказок или предложений по исправлению маркеров clang-tidy не предлагается.

Целью данной работы является разработка и интеграция механизма быстрых исправлений ошибок на основе clang-tidy в IDE на базе Eclipse и реализация необходимого графического пользовательского интерфейса.

Для решения поставленной задачи был принят поэтапный подход, включающий анализ существующих инструментов для рефакторинга, разработку логики для исправления определенного перечня ошибок, обнаруживаемых clang-tidy, а также тестирование работоспособности быстрых исправлений в составе IDE. Таким образом, данная работа направлена на создание инструмента, который упростит процесс исправления ошибок.

1. Постановка задачи

Целью работы является создание механизма быстрых исправлений для ошибок и замечаний, сделанных clang-tidy, в IDE для C/C++ на базе Eclipse. Для её выполнения были поставлены следующие задачи:

1. Спроектировать архитектуру механизма для реализации быстрых исправлений.
2. Реализовать быстрые исправления для замечаний clang-tidy, которые поддерживают автоматическое исправление.
3. Реализовать быстрые исправления для замечаний clang-tidy, которые требуют переименования переменных.
4. Протестировать механизм быстрых исправлений и провести апробацию.

2. Обзор

2.1. Eclipse

Eclipse [7] — это гибкая платформа с открытым исходным кодом, которая способна выполнять задачи в роли многоязыковой среды разработки и одновременно с этим предоставляет инструментарий для создания специализированных решений на основе своего фреймворка. В контексте данной работы Eclipse полезен своей функциональностью для работы с маркерами ошибок. Так, внутри Eclipse уже присутствуют необходимые инструменты для создания, просмотра и разрешения маркеров, а также гибкая настройка их атрибутов. Маркеры в Eclipse [6] чаще всего обладают следующими атрибутами:

- `IMarker.MESSAGE` — текстовое сообщение, описывающее маркер;
- `IMarker.SEVERITY` — уровень серьезности;
- `IMarker.LINE_NUMBER` — номер строки, к которой привязан маркер;
- `IMarker.CHAR_START` и `IMarker.CHAR_END` — начальная и конечная позиции символов в файле.

Вместе с атрибутами, предлагаемыми Eclipse, разработчик может создавать свои собственные атрибуты маркеров.

В платформе Eclipse также есть инструменты для обеспечения решений проблем, помеченных маркерами. Механизм быстрых исправлений реализуется с помощью двух ключевых интерфейсов:

- `org.eclipse.ui.IMarkerResolution` [11] — описывает конкретное исправление для маркера;
- `org.eclipse.ui.IMarkerResolutionGenerator` [12] — генерирует список исправлений для маркера.

2.2. CLDT и clang-tidy в его составе

Clang Development Tool (CLDT) — IDE для языков C/C++, основанный на clang [10] и LLVM [9], которую разрабатывает «Софтком» на базе Eclipse. IDE поддерживает статический анализ clang-tidy для исходного кода и генерирует маркеры предупреждений, ошибок и замечаний на нужной строке исходного кода. На картинке ниже показана архитектура решения для интеграции clang-tidy (Рис. 1).

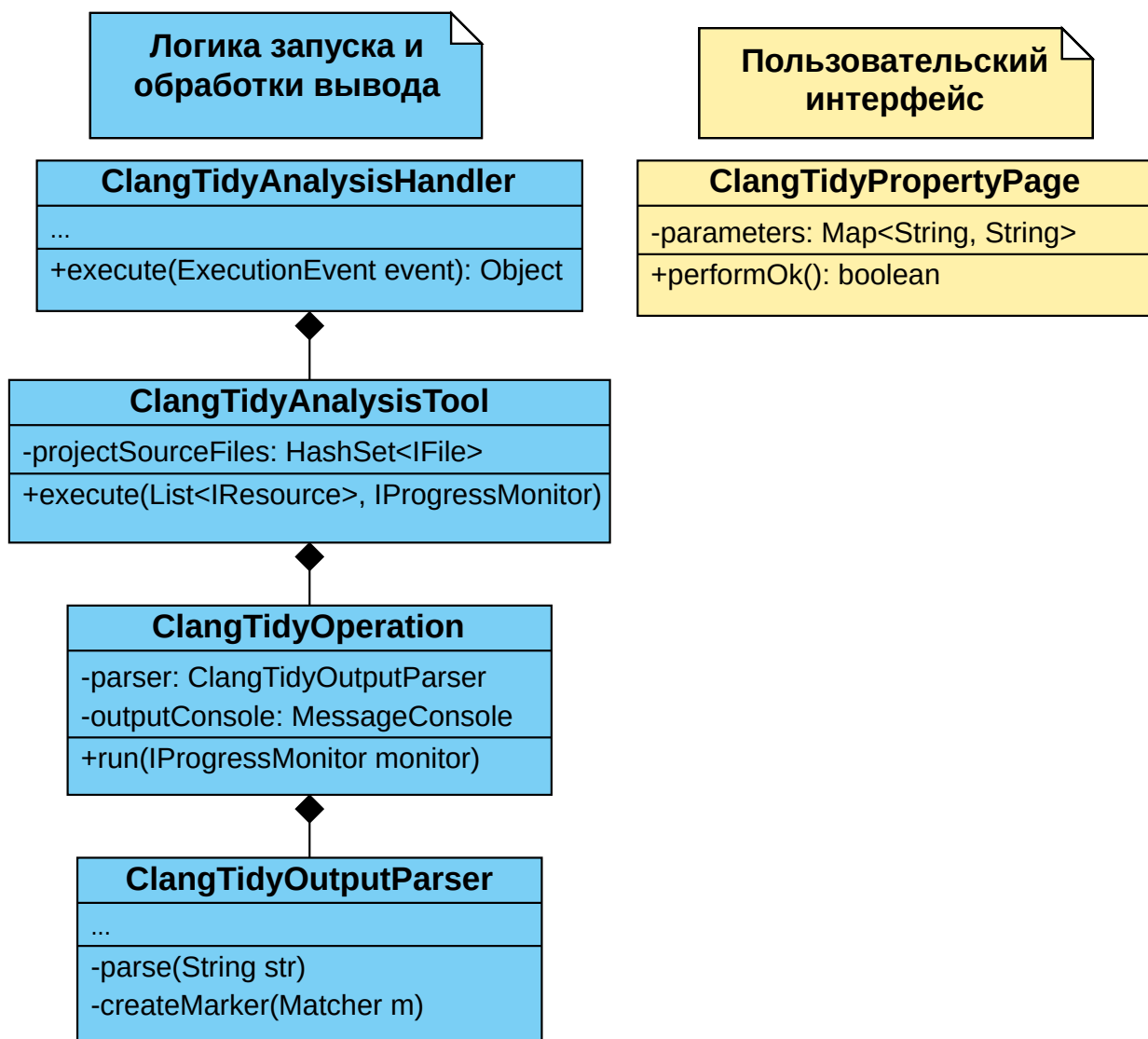


Рис. 1: Архитектура решения для статического анализа clang-tidy.

2.3. Выбор типов маркеров для исправления

Задачей, которую поставил заказчик, является внедрение возможности применения исправлений для ряда маркеров проверок. Первая группа проверок — та, для которой clang-tidy официально предоставляет свои готовые исправления, подробнее с их списком можно ознакомиться на сайте с документацией [5]. На момент выполнения работы список проверок был взят из документации Extra Clang Tools версии 21.0.0git. Их общее количество составляет 224 проверки. Кроме того, заказчик хочет внедрения и других полезных исправлений маркеров проверок, которые требуют взаимодействия с пользователем. К таким проверкам относятся readability-identifier-length и misc-confusable-identifiers. Обе проверки связаны с именованием переменных.

2.4. Обзор используемых технологий

Eclipse IDE for RCP and RAP Developers — специальная среда разработки, предназначенная для создания плагинов и расширений для Eclipse. Она включает в себя Eclipse Plug-in Development Environment [8], который предоставляет инструменты для создания, разработки, тестирования, отладки, сборки и развертывания проектов на основе Eclipse.

JDK (Java Development Kit) — набор инструментов для разработки программного обеспечения на языке программирования Java, включающий вспомогательные библиотеки, Java Runtime Environment (JRE) и Java Virtual Machine (JVM).

2.5. Выбор инструмента для рефакторинга

Для выбора инструмента, с помощью которого будут реализованы быстрые исправления, были рассмотрены следующие критерии:

- разнообразие предлагаемой функциональности;

- простота реализации быстрых и эффективных исправлений с использованием данного инструмента;
- интеграция с CLDT.

Clang содержит несколько инструментов для рефакторинга кода:

- Clang-rename [3]

Узконаправленный инструмент, позволяющий производить переименование классов, функций, переменных, аргументов, пространств имён и т. д. как в небольших, так и в масштабных проектах.

- Clang-tidy [1]

Статический анализатор, уже внедренный в разрабатываемую IDE. Имеет готовые быстрые исправления для множества ошибок и позволяет писать свои собственные проверки.

- Clang-refactor [2]

Мощный инструмент, уже интегрированный в CLDT, предлагающий несколько высокоуровневых операций для рефакторинга кода, может быть полезен для реализации исправления предупреждений, которые находит clang-tidy, но не имеет собственных предложений по исправлению.

- Clang-transformer [4]

В отличие от clang-refactor является низкоуровневым. Рефакторинг осуществляется при взаимодействии с AST исходного кода с помощью абстракций ASTMatcher и ASTEdit. Больше подходит для реализации сложных рефакторингов, требующих анализа AST.

Критерий	Clang-rename	Clang-tidy	Clang-refactor	Clang-transformer
Разнообразие функциональности	Переименование	Собственные исправления	Основные рефакторинги	Гибкие AST-трансформации
Простота реализации исправлений	Высокая	Высокая	Высокая	Низкая
Интегрирован в CLDT	Нет	Да	Да	Нет

Таблица 1: Сравнение инструментов рефакторинга

Таким образом, использование встроенных исправлений clang-tidy является оптимальным и покрывает большинство случаев, а для реализации исправлений, требующих взаимодействия с пользователем, таких как изменение имени переменной, подходит clang-refactor, так как он обладает нужной функциональностью и уже поддерживается IDE.

2.6. Clang-refactor в составе IDE

Как было упомянуто ранее, clang-refactor внедрён в состав среды разработки. Ниже представлена архитектура механизма для его интеграции (Рис. 2).

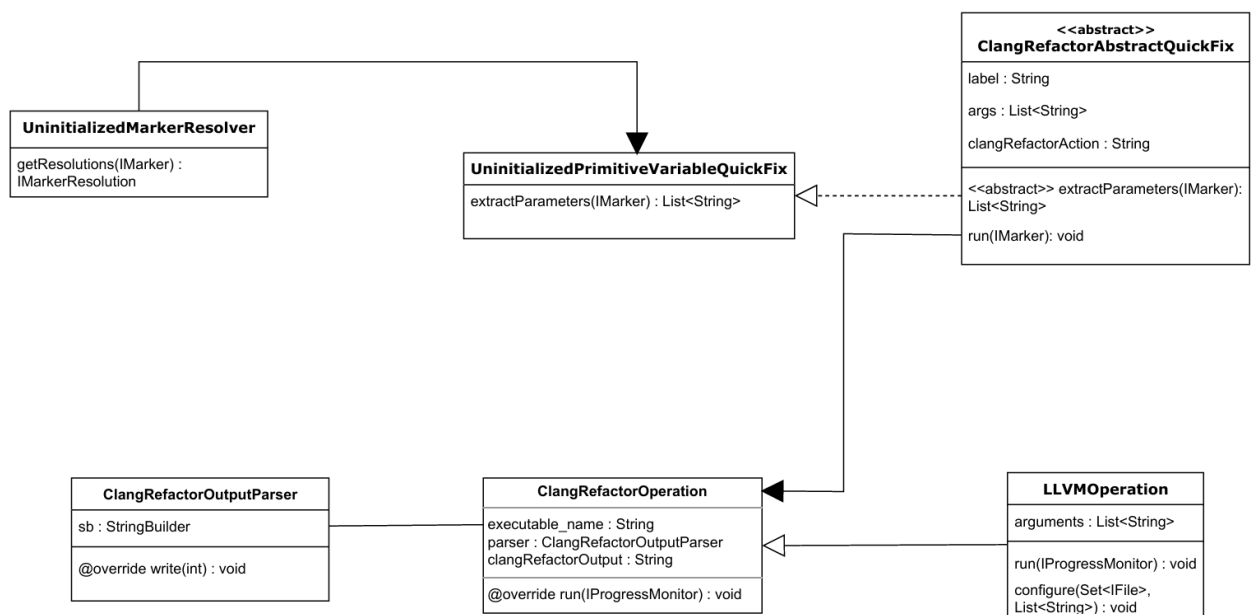


Рис. 2: Архитектура механизма интеграции clang-refactor.

2.7. Выводы

Подводя итог, для создания механизма разрешения маркеров, сгенерированных после статического анализа clang-tidy, будет использовано два инструмента: сам clang-tidy, предлагающий широкий спектр исправлений, и clang-refactor для исправлений, требующих переименования переменных.

3. Реализация

В данном разделе будет рассмотрена архитектура решения и классы с методами, которые были реализованы в процессе разработки механизма быстрых исправлений маркеров, полученных после анализа clang-tidy.

3.1. Архитектура механизма быстрых исправлений

Реализация быстрых исправлений маркеров clang-tidy разбивается на 5 компонентов, как показано на рисунке 3.

- Генерация маркеров анализа

В IDE уже интегрирован статический анализ clang-tidy. Он представлен классами ClangTidyAnalysisHandler, ClangTidyAnalysisTool, ClangTidyOperation и ClangTidyOutputParser. В контексте данной работы наибольший интерес вызывают последние два класса, они изображены на иллюстрации архитектуры 3. ClangTidyOperation представляет операцию запуска clang-tidy, а ClangTidyOutputParser создает маркеры анализа на нужных строках исходного кода.

- Поиск и применение быстрых исправлений

Для хранения всех возможных checkId (идентификаторов проверок clang-tidy), которые поддерживают быстрые исправления, используется класс ClangTidyQuickFixes с соответствующими методами, проверяющими поддержку быстрого исправления и конкретизирующие тип исправления (автоматическое или переименование).

ClangTidyMarkerResolver реализует интерфейс IMarkerResolutionGenerator и необходим для создания объектов классов быстрых исправлений для конкретных маркеров в исходном коде.

Компонент ClangTidyAutoQuickFix реализует IMarkerResolver, и

его задачей является запуск операции ClangTidyOperation с соответствующими флагами для проведения быстрого исправления. Так как ClangTidyOperation требует наличия базы данных команд компиляции, перед запуском нужно удостовериться, что она существует.

За осуществление исправлений кода, требующих переименования переменных, отвечает ClangTidyRenameQuickFix, использующий для переименования другой инструмент, имеющийся в IDE — clang-refactor и связанный с ним ClangRefactorOperation.

Класс VariableRenameDialog представляет собой диалоговое окно, в которое пользователь должен ввести новое имя переменной.

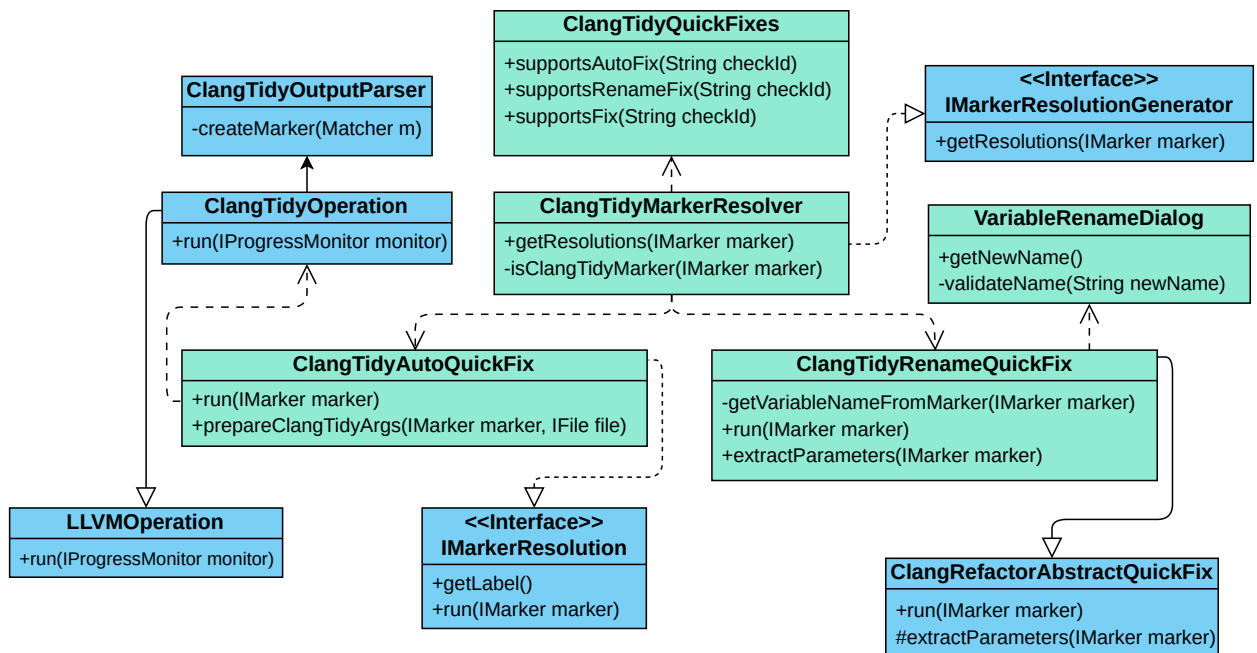


Рис. 3: Архитектура механизма быстрых исправлений. Классы, отмеченные зелёным цветом реализованы в процессе работы.

3.2. Проведение необходимых преобразований для реализации быстрых исправлений

Перед созданием механизма быстрых исправлений ClangTidyOutputParser нуждался в доработке, так как изначально все

маркеры генерировались с идентификатором для анализа по умолчанию. Кроме того, маркеры не содержали в себе атрибут, обозначающий их принадлежность к конкретному типу проверок clang-tidy. С целью это исправить, был создан и добавлен в нужную точку расширения идентификатор маркера «su.softcom.cldt.core.marker.clangTidyMarker», а внутри метода создания маркера был добавлен атрибут checkId.

В дальнейшем оказалось, что этих преобразований было недостаточно. Текущая архитектура интеграции анализа clang-tidy хотя и позволяла запускать ClangTidyOperation асинхронно, но после запуска возникали проблемы с ожиданием её завершения. С целью улучшения взаимодействия с ClangTidyOperation его реализация была изменена. ClangTidyOperation теперь наследуется от LLVMOperation, а ClangTidyOutputParser расширяет OutputStream. Кроме того, на момент выполнения работы реализация ClangRefactorOperation, которая должна переписывать файл и сразу применять рефакторинг, работала также некорректно, стирая всё содержимое файла. Источником этого поведения являлось неправильное использование ProcessOutputMonitor при считывании вывода clang-refactor. Было принято решение заменить парсинг вывода clang-refactor и переписывание файла из кода на добавление параметра «-i» к запуску clang-refactor при применении исправления, который записывает применённый рефакторинг в файл автоматически. Некоторые дополнительные изменения, связанные с механизмом интеграции clang-tidy будут описаны дальше в тексте в связи с тем, что они вносились на этапе тестирования.

3.3. Формирование класса с идентификаторами маркеров, поддерживающих исправлений

ClangTidyQuickFixes должен предоставлять доступ к идентификаторам маркеров, которые поддерживают исправления. В первую очередь, это должны быть все маркеры, напротив которых стоит метка «Yes» на сайте со списком возможных проверок clang-tidy и наличием их автоматического исправления [5]. Их перечень был полу-

чен путём парсинга страницы python-скриптом. Следующая категория — маркеры, которые можно исправить переименованием переменной. Среди них: readability-identifier-length, misc-confusable-identifiers. Для хранения проверок, для которых поддерживаются быстрые исправления, были созданы два множества AUTO_FIXABLE_CHECKS и RENAME_FIXABLE_CHECKS. Проблемой данного подхода является невозможность автоматического масштабирования. Все проверки clang-tidy, для которых в новых версиях Clang Extra Tools будут реализованы автоматические быстрые исправления, придётся добавлять в множество AUTO_FIXABLE_CHECKS вручную. Например, в сравнении с версией Clang Extra Tools 18.1.8 новая версия 21.0.0git насчитывает на 12 исправлений больше.

3.4. Реализация автоматических исправлений

Для реализации исправлений маркеров проверок, для которых существует официальное исправление (224 проверки), был реализован механизм запуска clang-tidy с параметром «-fix».

Чтобы к маркеру сгенерировались быстрые исправления, необходимо, чтобы существовал связанный с этим типом маркера класс, реализующий IMarkerResolutionGenerator, который внутри себя вернёт массив объектов классов, реализующих IMarkerResolver. Для генерации исправлений был реализован класс ClangTidyMarkerResolver. Метод isClangTidyMarker(IMarker marker) проверяет идентификатор маркера, getResolutions(IMarker marker) возвращает массив с исправлениями. Чтобы генератор исправлений распознавался внутри Eclipse, он был добавлен в нужную точку расширения в файле plugin.xml.

При реализации класса автоматического быстрого исправления ClangTidyAutoQuickFix в его конструктор был добавлен аргумент IMarker marker, с последующим определением его checkId и добавлением названия проверки в поле label класса. Это сделано с целью различать быстрые исправления, которые появляются при генерации нескольких маркеров на одной строке. Это не помогает процессу исправ-

ления, но делает более наглядным для пользователя список доступных исправлений. Пример показан на рисунке 4.

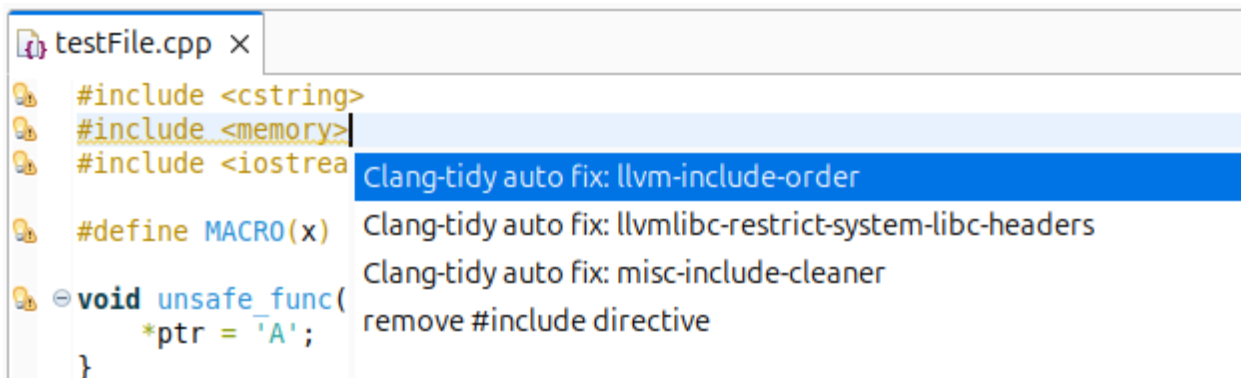


Рис. 4: Именованное исправление clang-tidy.

В реализации метода `run(IMarker marker)` происходит подготовка аргументов `clang-tidy` и генерация базы данных компиляции, если такой нет, в методах `prepareClangTidyArgs(IMarker marker, IFile file)` и `prepareCompileCommands(IProject project, String buildFolder)`. В аргументы `clang-tidy` добавляются параметры «`-fix`» для исправления и «`-line-filter`», который локализует область проверки кода и позволяет исправить не все маркеры файла, а единственный на нужной строке. Далее происходит асинхронный запуск `ClangTidyOperation`. Важен факт, что изначально проверкой наличия команд компиляции занимался класс операции `ClangTidyOperation`, но впоследствии метод пришлось вынести в вышестоящие классы в связи со следующей проблемой. `ClangTidyOperation` в случае быстрого исправления блокирует файл исходного кода с помощью метода `setRule(ISchedulingRule rule)` фреймворка `Job`. Метод позволяет заблокировать доступ к ресурсу до конца работы потока. Конфликт возникает в случае отсутствия файла `compile_commands.json`, когда `ClangTidyOperation` пытается запустить конфигурацию проекта для его генерации, и этот запуск происходит с менее строгим правилом блокировки, что противоречит задумке фреймворка. То же самое происходит и в ситуации, когда требуется обновить файл внутри кода с помощью метода `refreshLocal`. При этом, если запускать обновление файла после вызова асинхронного запуска `ClangTidyOperation`, оно успешно произойдет, но содержимое файла мо-

Листинг 1: Запуск операции автоматического быстрого исправления.

```
Job job = Job.create("Быстрое исправление clang-tidy", monitor -> {
    try {
        ClangTidyOperation operation = new ClangTidyOperation(project, Set.of(file), args,
            ↪ buildFolder);
        operation.run(monitor);

        marker.delete();

        file.touch(monitor);
        return Status.OK_STATUS;
    } catch (CoreException e) {
        return new Status(IStatus.ERROR, "Clang-tidy", "Ошибка при исправлении", e);
    }
});
```

жет не успеть измениться. Для решения этой проблемы был использован подход, описанный на листинге 1.

Создается job, который содержит в себе последовательный вызов ClangTidyOperation, marker.delete и file.touch. Так гарантируется, что маркер не удалится раньше запуска операции, а операция завершится перед вызовом метода touch, который не обновляет файл, а создает на нём метку о наличии изменений.

Таким способом было достигнуто корректное поведение механизма автоматических исправлений.

3.5. Реализация исправления с переименованием

Две проверки, а именно readability-identifier-length и misc-confusable-identifiers, связаны с именами переменных, поэтому для того, чтобы разрешить связанный с ними маркер, было написано быстрое исправление с переименованием проблемной переменной.

Сценарий использования этого быстрого исправления следующий:

1. Пользователь запускает анализ clang-tidy.

2. На нужных строках исходного кода генерируются маркеры анализа.
3. Для маркеров с `checkId readability-identifier-length` или `misc-confusable-identifiers` становится доступным меню быстрых исправлений.
4. В меню быстрых исправлений пользователю предлагается изменить название переменной (рисунок 5).
5. При выборе этого быстрого исправления открывается диалоговое окно ввода нового имени переменной.
6. После ввода нового имени применяется переименование.

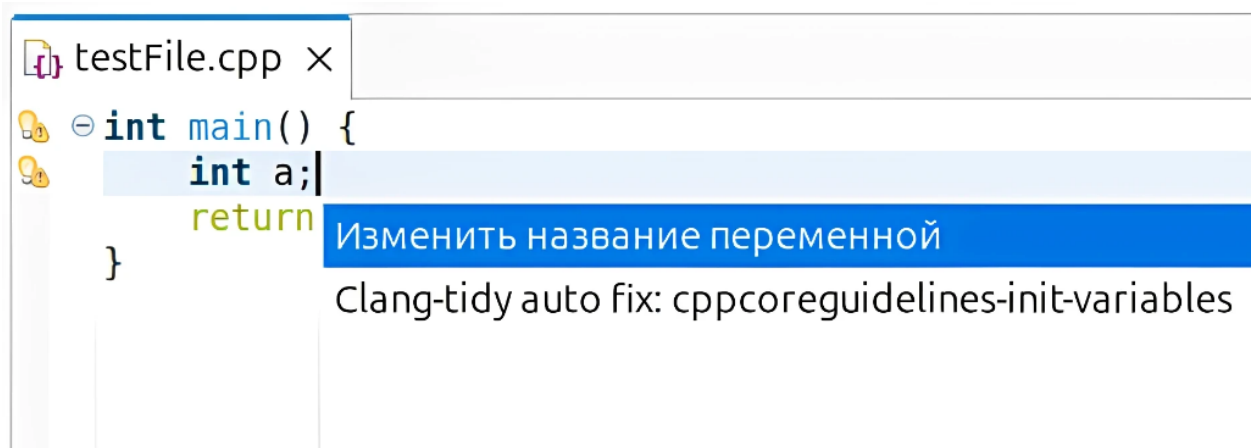


Рис. 5: Меню выбора быстрых исправлений. В данном случае длина имени переменной `b` слишком короткая.

Для реализации быстрых исправлений, связанных с переименованием, был создан класс `ClangTidyRenameQuickFix`, наследующийся от `ClangRefactorAbstractQuickFix`. В `ClangTidyMarkerResolver` добавилась обработка маркеров, поддерживающих это исправление. Уже имеющийся класс `ClangRefactorAbstractQuickFix` выполняет рефакторинг с заданным действием. Для выполнения переименования в работе используется `clang-refactor local-rename`, так как других средств для переименования на момент выполнения работы не было. `Local-rename` переименовывает переменные только в одной единице трансляции. Для

полного переименования нужно использовать иные действия clang-refactor, разработка которых не относится к этой работе и будет реализована отдельно. Для подготовки аргументов для clang-refactor local-rename был реализован метод `extractParameters(IMarker marker)`, который по маркеру получает параметр «-selection». Для использования этого быстрого исправления был написан класс диалогового окна `VariableRenameDialog`, который предлагает ввести новое имя переменной. В методе `validateName(String newName)` происходит валидация нового имени: проверка, что имя не начинается с цифры, не зарезервировано C/C++ и т. д., если валидация прошла успешно, к параметрам добавляется «-new-name». Для удобства пользователя в диалоговом окне написано старое имя переменной, которое находит метод `getVariableNameFromMarker(IMarker marker)`, используя простое регулярное выражение.

4. Тестирование и апробация

Данный раздел посвящён тестированию механизма быстрых исправлений и апробации.

4.1. Тестирование

Для механизма быстрых исправлений были написаны интеграционные и модульные тесты с помощью фреймворка JUnit. Покрытие кода тестами составляет более 84%. Сценариями для интеграционных и модульных тестов являются:

1. ClangTidyMarkerResolver генерирует автоматическое быстрое исправление для маркеров clang-tidy.
2. ClangTidyMarkerResolver генерирует исправление с переименованием для маркеров clang-tidy.
3. ClangTidyMarkerResolver не генерирует исправление для неподходящего маркера.
4. ClangTidyAutoQuickFix корректно исправляет маркер clang-tidy.
5. ClangTidyAutoQuickFix корректно исправляет нужный маркер clang-tidy при наличии нескольких маркеров на строке.
6. ClangTidyRenameQuickFix корректно исправляет маркер clang-tidy.

Также были проведены ручные тесты:

1. При выборе исправления применяется выбранное исправление.
2. При наличии нескольких исправлений все они отображаются для выбранной строки исходного кода.
3. При введении некорректного имени переменной диалоговое окно выводит правильную ошибку.

4. При введении корректного имени переменной быстрое исправление применяется с заданным новым именем переменной.
5. Имя переменной, которую предстоит переименовать, отображается корректно в диалоговом окне.

4.2. Апробация

Апробация проводилась с участием сотрудников компании «Софт-ком». Целью апробации являлись оценка работоспособности кода и поиск предложений для улучшения решения.

По результатам апробации были выявлены следующие проблемы:

- вывод в консоль находится в классе `ClangTidyOperation` в `ClangTidyOutputParser`;
- неточный шаблон для имени переменной;
- в тестах используются необязательные файлы.

Сотрудники предложили добавить дополнительную функциональность:

- сохранение маркеров после перезапуска CLDT;
- графическое окно настроек проверок clang-tidy;
- создать интуитивно понятные для пользователя названия проверок clang-tidy.

Замечания были исправлены, и почти все предложения были реализованы, а именно:

- вывод в консоль перенесен из `ClangTidyOperation` в `ClangTidyOutputParser`;
- скорректирован шаблон имени переменной;
- исправлена реализация тестов;

- реализовано сохранение маркеров после перезапуска CLDT;
- добавлено графическое окно настроек проверок clang-tidy.

Заключение

В ходе выполнения данной работы были достигнуты все поставленные цели:

- Спроектирована архитектура плагина для реализации быстрых исправлений маркеров, сгенерированных clang-tidy.
- Реализовано автоматическое быстрое исправление.
- Реализовано быстрое исправление, требующее переименования переменной.
- Протестирован механизм быстрого исправления и проведена апробация.

Механизм быстрых исправлений был влит в основную ветку проекта. Код доступен в репозитории [github](https://github.com)¹.

¹Ссылка на репозиторий проекта: <https://github.com/softcom-su/CLDT>

Список литературы

- [1] Clang-tidy документация. — URL: <https://clang.llvm.org/extra/clang-tidy/>.
- [2] Документации clang's refactoring engine и описание clang-refactor. — URL: <https://clang.llvm.org/docs/RefactoringEngine.html>.
- [3] Документация clang-rename. — URL: <https://releases.llvm.org/3.9.0/tools/clang/tools/extra/docs/clang-rename.html>.
- [4] Документация clang-transformer. — URL: <https://clang.llvm.org/docs/ClangTransformerTutorial.html>.
- [5] Исправления, поддерживаемые clang-tidy. — URL: <https://clang.llvm.org/extra/clang-tidy/checks/list.html>.
- [6] Маркеры в Eclipse. — URL: <https://archive.eclipse.org/eclipse/downloads/documentation/2.0/html/plugins/org.eclipse.platform.doc.isv/reference/api/org/eclipse/core/resources/IMarker.html>.
- [7] Официальный сайт Eclipse IDE. — URL: <https://eclipseide.org/>.
- [8] Официальный сайт Eclipse PDE. — URL: <https://eclipse.dev/pde/>.
- [9] Сайт LLVM. — URL: <https://llvm.org/>.
- [10] Сайт clang. — URL: <https://clang.llvm.org/>.
- [11] Спецификация IMarkerResolution в Eclipse. — URL: <https://archive.eclipse.org/eclipse/downloads/documentation/2.0/html/plugins/org.eclipse.platform.doc.isv/reference/api/org/eclipse/ui/IMarkerResolution.html>.

- [12] Спецификация `IMarkerResolutionGenerator` в Eclipse. —
URL: <https://archive.eclipse.org/eclipse/downloads/documentation/2.0/html/plugins/org.eclipse.platform.doc.isv/reference/api/org/eclipse/ui/IMarkerResolutionGenerator.html>.