

Санкт-Петербургский государственный университет

Целовальникова Екатерина Андреевна

Выпускная квалификационная работа

Интеграция инструментов оценки
тестового покрытия кода в IDE для языков
C/C++ на платформе Eclipse

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование
информационных систем»*

Основная образовательная программа *СВ.5162.2021 «Технологии программирования»*

Научный руководитель:

Доцент кафедры системного программирования, к. ф.-м. н., Луцив Д. В.

Рецензент:

Разработчик ПО, ООО «Софтком», Низамов Р. Ф.

Санкт-Петербург
2025

Saint Petersburg State University

Ekaterina Tselovalnikova

Bachelor's Thesis

Integration of code coverage assessment tools into the IDE for C/C++ languages on the Eclipse platform

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *CB.5162.2021 "Programming Technologies"*

Scientific supervisor:
C.Sc., docent D.V. Lutsiv

Reviewer:
Software Developer, "Softcom" R.F. Nizamov

Saint Petersburg
2025

Оглавление

Введение	4
1. Постановка задачи	6
2. Обзор	7
2.1. Обзор существующих плагинов	7
2.2. Анализ существующих плагинов	9
2.3. Обзор используемых технологий	16
3. Ход работы	18
3.1. Архитектура плагина	18
3.2. Создание конфигурации запуска	19
3.3. Отображение результатов покрытия	22
3.4. Подсветка строк в редакторе	26
3.5. Создание страницы глобальных настроек	28
3.6. Создание страницы проектных настроек	31
4. Эксперимент	35
4.1. Тестирование	35
4.2. Апробация	36
Заключение	37
Список литературы	39

Введение

Разработка программного обеспечения (ПО) является одной из ключевых областей современной индустрии, обеспечивающей автоматизацию процессов в различных сферах деятельности. Интегрированные среды разработки (IDE) представляют собой программные комплексы, объединяющие инструменты для проектирования, написания кода, отладки и тестирования, что позволяет обеспечить соответствие создаваемых продуктов высоким стандартам качества.

Платформа Eclipse [14] является одной из наиболее распространённых и востребованных сред разработки в мире. Благодаря модульной архитектуре и поддержке множества языков программирования платформа обеспечивает высокую степень гибкости в настройке и позволяет интегрировать сторонние плагины для расширения функциональности, что делает Eclipse универсальным инструментом для создания сложных программных решений.

В условиях возрастающих требований к качеству программного обеспечения внедрение современных методов анализа и тестирования на всех стадиях разработки становится необходимым. Одним из таких методов является анализ покрытия кода, который позволяет выявить участки программы, не охваченные тестированием, что способствует обнаружению потенциальных уязвимостей и повышению надёжности разрабатываемого продукта.

Общество с ограниченной ответственностью «Софтком» осуществляет разработку интегрированной среды разработки на основе платформы Eclipse. Основной целью деятельности компании является создание высокофункционального инструмента, соответствующего современным стандартам и требованиям в области разработки программного обеспечения, а также способного удовлетворять широкий спектр потребностей разработчиков на всех этапах жизненного цикла программных продуктов.

В рамках настоящей работы поставлена задача разработки плагина для анализа покрытия кода, предназначенного для интеграции в

интегрированную среду разработки на базе платформы Eclipse. Разработанный плагин предоставит разработчикам инструмент для детального анализа тестового покрытия, что позволит повысить эффективность процессов тестирования программных решений.

1. Постановка задачи

Целью данной работы является интеграция инструментов оценки тестового покрытия кода в IDE для языков C/C++ на платформе Eclipse.

Для достижения цели были поставлены следующие задачи:

1. Провести обзор существующих плагинов.
2. Разработать механизм конфигурации запуска анализа покрытия.
3. Разработать графический интерфейс:
 - отображение результатов анализа;
 - страница глобальных настроек;
 - страница проектных настроек.
4. Реализовать подсветку строк кода для наглядного отображения покрытия.
5. Провести тестирование и апробацию разработанного решения.

2. Обзор

Для интеграции инструмента анализа покрытия кода в IDE для языков C/C++ на платформе Eclipse необходимо разработать плагин, интегрируемый в существующую среду разработки. В настоящее время существует несколько готовых решений, предлагающих аналогичную функциональность.

2.1. Обзор существующих плагинов

В данном обзоре проведено сравнение плагинов по следующим критериям:

- независимость от CDT: плагин должен быть независим от CDT [13], так как разрабатываемая IDE не использует этот компонент;
- совместимость с Clang: плагин должен обеспечивать совместимость с компилятором Clang [1], так как IDE использует инструменты LLVM [22] для компиляции и анализа кода;
- возможность настройки анализа покрытия: плагин должен предоставлять интерфейс для настройки параметров анализа покрытия пользователем;
- отчетность и визуализация: плагин должен обеспечивать генерацию отчетов о покрытии кода и визуализацию результатов анализа.

2.1.1. LinuxTools GCov

LinuxTools GCov [15] — плагин для интеграции инструмента GCov [20] с Eclipse, предназначенный для анализа покрытия кода, скомпилированного с использованием компилятора GCC [19].

Достоинства:

- возможность настройки анализа покрытия: плагин предоставляет интерфейс для настройки параметров анализа покрытия;
- отчетность и визуализация: плагин обеспечивает генерацию отчетов о покрытии кода, отображаемых в Eclipse Views (Gcov View).

Недостатки:

- независимость от CDT: плагин зависит от CDT, что делает его несовместимым с IDE, не использующими этот компонент;
- совместимость с Clang: плагин ориентирован на GCC, официальной поддержки Clang нет, хотя адаптация возможна.

2.1.2. Testwell CTC++

Testwell CTC++ [21] — коммерческий инструмент промышленного уровня для анализа покрытия кода, широко используемый в разработке встраиваемых систем.

Достоинства:

- совместимость с Clang: инструмент поддерживает Clang, обеспечивая интеграцию с экосистемой LLVM;
- возможность настройки анализа покрытия: плагин предоставляет интерфейс для настройки параметров анализа покрытия;
- отчетность и визуализация: инструмент предоставляет HTML-отчеты для анализа покрытия кода.

Недостатки:

- независимость от CDT: плагин зависит от CDT, что делает его несовместимым с IDE, не использующими этот компонент.

2.1.3. Parsoft C/C++test

Parsoft C/C++test [25] — коммерческий инструмент для тестирования и анализа кода C/C++, поддерживающий статический и динамический анализ, а также покрытие кода.

Достоинства:

- совместимость с Clang: инструмент поддерживает Clang, обеспечивая интеграцию с экосистемой LLVM;
- возможность настройки анализа покрытия: плагин предоставляет интерфейс для настройки параметров анализа покрытия;
- отчетность и визуализация: плагин отображает результаты в Parsoft C++test perspective в Eclipse и предоставляет отчеты в форматах HTML, PDF и XML.

Недостатки:

- независимость от CDT: плагин зависит от CDT, что делает его несовместимым с IDE, не использующими этот компонент.

Проведённый обзор продемонстрировал отсутствие решений, удовлетворяющих ключевому требованию — независимости от CDT. В связи с этим принято решение о разработке нового плагина, полностью соответствующего поставленным техническим и функциональным требованиям.

2.2. Анализ существующих плагинов

На данном этапе проведён анализ плагинов, которые интегрируют инструменты анализа покрытия кода в среду разработки Eclipse. Исследовались технологии визуализации, взаимодействие с инструментами, конфигурация окружения и пользовательский интерфейс с акцентом на возможность реализации без CDT.

Категория	LinuxTools GCov	Testwell CTC++	Parasoft C/C++test
Визуализация результатов	SWT, JFace, BIRT	HTML-отчёты	Parasoft C++test perspective, HTML/PDF/XML-отчёты
Взаимодействие с инструментом	CDT Build Settings (флаги компиляции)	CDT Build Settings (инструментация через ctc)	CDT Build Settings
Конфигурация	CDT Build Settings (параметры компиляции)	CDT Build Settings (настройка ctc)	CDT Build Settings
UI/UX	JFace, Eclipse Action Framework	HTML-отчёты	Parasoft C++test perspective

Рис. 1: Сравнительный анализ плагинов для Eclipse

2.2.1. LinuxTools GCov

На первом этапе анализа была рассмотрена реализация плагина LinuxTools GCov, обеспечивающего интеграцию инструмента GCov в Eclipse.

Визуализация результатов

Результаты анализа покрытия отображаются в Gcov View с использованием следующих технологий:

- таблицы с сортировкой, фильтрацией и группировкой данных, реализованные через SWT [18] и JFace [17];
- диаграммы статистики покрытия, генерируемые с помощью Eclipse BIRT [12].

Эти технологии независимы от CDT, что позволяет использовать их в разрабатываемом плагине без дополнительных зависимостей.

Взаимодействие с инструментами

Плагин использует CDT для автоматизации взаимодействия с GCov:

- добавление флагов `-ftest-coverage` и `-fprofile-arcs` через CDT Build Settings [29];
- запуск утилиты GCov по завершении сборки;
- связывание файлов `.gcov` с исходным кодом через настройки путей CDT.

Пример флагов в CDT Build Settings:

```
1 gcc -c -ftest-coverage -fprofile-arcs source.c -o source.o
```

Без CDT возможно применение следующих технологий:

- ручное добавление флагов в CMake [31];
- запуск GCov через Eclipse External Tools [9] с использованием скриптов;
- парсинг файлов .gcov внутренними парсерами Eclipse.

Пример настройки в CMakeLists.txt:

```
1 add_executable(myapp source.c)
2 target_compile_options(myapp PRIVATE -ftest-coverage -fprofile-arcs)
3 target_link_options(myapp PRIVATE -fprofile-arcs)
```

Конфигурация окружения

Конфигурация в CDT включает:

- настройку путей к библиотекам через CDT Build Settings;
- автоматизацию запуска GCov через CDT;
- управление сборкой через CDT Managed Build [28].

Без CDT возможно применение следующих технологий:

- ручная настройка в CMake;
- сохранение конфигурации в XML-файлах;
- выполнение команд через Eclipse External Tools.

Пример настройки в CMakeLists.txt:

```
1 add_executable(myapp source.c)
2 target_compile_options(myapp PRIVATE -ftest-coverage -fprofile-arcs)
3 target_link_options(myapp PRIVATE -fprofile-arcs)
```

Пользовательский интерфейс

Интерфейс реализован с использованием:

- таблицы с сортировкой и фильтрацией в Gcov View на основе JFace;
- элементы управления для экспорта данных (например, в CSV) через Eclipse Action Framework [10].

Эти технологии независимы от CDT и применимы для разрабатываемого плагина.

2.2.2. Testwell CTC++

На следующем этапе анализа была рассмотрена реализация плагина Testwell CTC++, обеспечивающего интеграцию инструмента CTC++ в Eclipse.

Визуализация результатов

Результаты анализа покрытия отображаются в HTML-отчетах, включающих:

- таблицы с метриками покрытия (покрытие строк, ветвей);
- диаграммы покрытия функций и модулей.

HTML-отчеты генерируются инструментом CTC++ и отображаются в браузере, независимо от CDT.

Взаимодействие с инструментами

Плагин использует CDT для взаимодействия с CTC++:

- инструментацию через утилиту `ctc` в CDT Build Settings;
- автоматизацию сборки через CDT Managed Build;
- анализ зависимостей через CDT Indexer [2].

Пример команды в CDT Build Settings:

```
1 ctc -i m gcc -c source.c -o source.o
```

Без CDT возможно применение следующих технологий:

- ручная настройка `ctc` в CMake;
- запуск `ctc` через Eclipse External Tools;
- парсинг отчетов CTS++ скриптами для интеграции с Eclipse.

Пример настройки в `CMakeLists.txt`:

```
1 add_custom_command(  
2     OUTPUT source.o  
3     COMMAND ctc -i m gcc -c source.c -o source.o  
4     DEPENDS source.c  
5 )
```

Конфигурация окружения

Конфигурация в CDT включает:

- настройку параметров `ctc` через CDT Build Settings;
- автоматизацию сборки через CDT Managed Build и CMake;
- интеграцию с Project Explorer [11].

Без CDT возможно применение следующих технологий:

- ручная настройка в CMake;
- сохранение конфигурации в XML-файлах;

- выполнение `ctc` через Eclipse External Tools.

Пример настройки в `CMakeLists.txt`:

```
1 add_custom_command(  
2     OUTPUT source.o  
3     COMMAND ctc -i m gcc -c source.c -o source.o  
4     DEPENDS source.c  
5 )
```

Пользовательский интерфейс

Интерфейс реализован с использованием:

- HTML-отчеты, отображаемые в браузере;
- элементы управления для настройки отчетов через CTC++.

Эти технологии независимы от CDT и применимы для разрабатываемого плагина.

2.2.3. Parsoft C/C++test

На заключительном этапе анализа была рассмотрена реализация плагина Parsoft C/C++test, обеспечивающего интеграцию инструмента C/C++test в Eclipse.

Визуализация результатов

Результаты анализа покрытия отображаются в Parsoft C++test perspective, включающей:

- таблицы с метриками покрытия в Violations View и Test Results View;
- отчеты в HTML, PDF, XML с графиками и статистикой.

Эти технологии, основанные на JFace, независимы от CDT.

Взаимодействие с инструментами

Плагин использует CDT для взаимодействия с C/C++test:

- настройку компиляции через CDT Build Settings;
- автоматизацию сборки через CDT Managed Build;
- анализ кода через CDT Indexer.

Пример команды в CDT Build Settings:

```
1 cpptesttrace gcc -c source.c -o source.o
```

Без CDT возможно применение следующих технологий:

- ручная настройка `cpptesttrace` в CMake;
- запуск C/C++test через Eclipse External Tools;
- парсинг результатов скриптами для интеграции с Eclipse.

Пример настройки в `CMakeLists.txt`:

```
1 add_custom_command(  
2     OUTPUT source.o  
3     COMMAND cpptesttrace gcc -c source.c -o source.o  
4     DEPENDS source.c  
5 )
```

Конфигурация окружения

Конфигурация в CDT включает:

- настройку параметров через CDT Build Settings;
- автоматизацию сборки через CDT Managed Build и CMake;
- интеграцию с Project Explorer.

Без CDT возможно применение следующих технологий:

- ручная настройка в CMake;
- сохранение конфигурации в XML-файлах;

- выполнение `cpptesttrace` через Eclipse External Tools.

Пример настройки в `CMakeLists.txt`:

```
1 add_custom_command(  
2     OUTPUT source.o  
3     COMMAND cpptesttrace gcc -c source.c -o source.o  
4     DEPENDS source.c  
5 )
```

Пользовательский интерфейс

Интерфейс реализован с использованием:

- Parasoft C++test perspective с таблицами на основе JFace;
- элементы управления для экспорта отчетов в HTML, PDF, XML.

Эти технологии независимы от CDT и применимы для разрабатываемого плагина.

2.2.4. Выводы

Анализ показывает, что CDT упрощает интеграцию инструментов покрытия кода за счёт автоматизации настройки и сборки. Однако технологии, такие как SWT, JFace, Eclipse Action Framework и Eclipse External Tools, позволяют создать независимое решение с дополнительной настройкой, минимизируя зависимость от CDT.

2.3. Обзор используемых технологий

Для разработки плагина анализа покрытия кода применялись следующие технологии:

- `llvm-cov` [26]: инструмент из набора LLVM для анализа покрытия кода; позволяет собирать информацию о выполнении участков программы во время тестирования и генерировать отчёты в различных форматах;

- `llvm-profdata` [27]: утилита из экосистемы LLVM для обработки и объединения профильных данных, собранных во время выполнения программы; преобразует сырые профильные данные в формат, пригодный для дальнейшего анализа;
- `Clang`: компилятор из экосистемы LLVM для языков C, C++ и других; поддерживает генерацию данных о покрытии кода с использованием специальных флагов компиляции;
- `Eclipse Platform`: платформа для разработки и интеграции плагинов; предоставляет API и инфраструктуру для управления проектами, редактирования кода и расширения функциональности;
- `Eclipse PDE` [16]: набор инструментов для создания, конфигурации и отладки плагинов в среде Eclipse; обеспечивает поддержку разработки расширений для платформы;
- `CMake`: кроссплатформенная система сборки для управления процессом компиляции и конфигурации проектов; используется для автоматизации сборки и управления зависимостями.

3. Ход работы

3.1. Архитектура плагина

Перед началом реализации была спроектирована архитектура плагина для анализа покрытия кода в среде Eclipse.

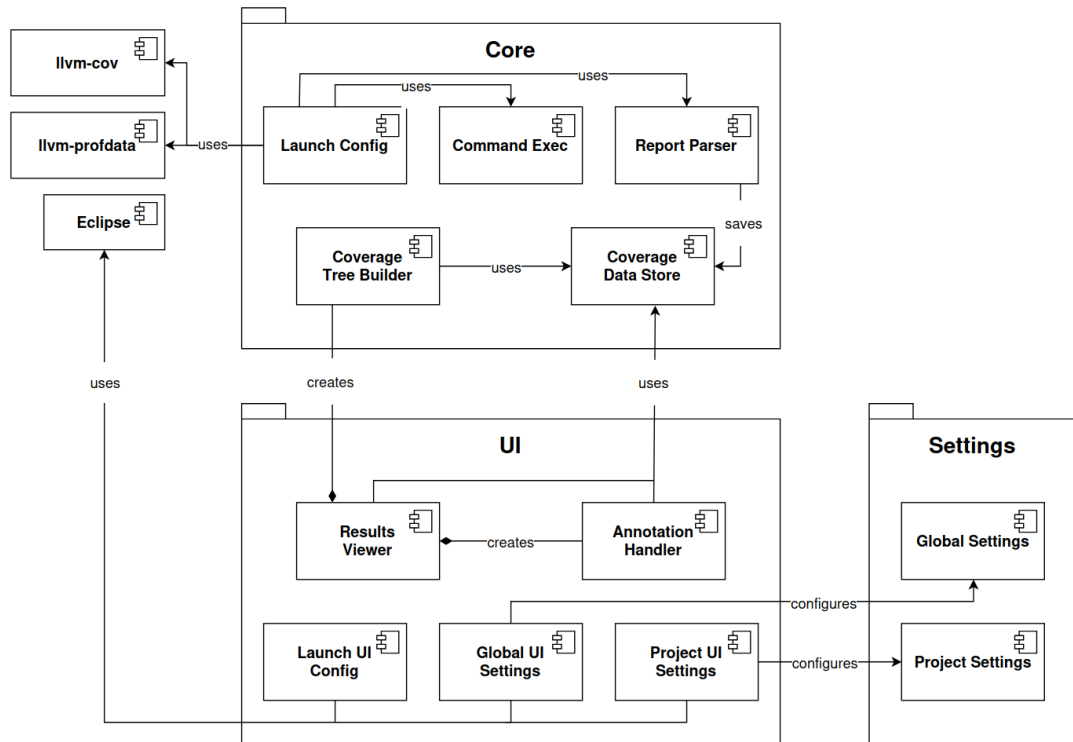


Рис. 2: Диаграмма компонентов плагина

3.1.1. Структура диаграммы

Плагин состоит из трёх пакетов, каждый из которых выполняет определённые функции:

- **Core:** управляет логикой покрытия, включая запуск тестов, парсинг отчетов о покрытии кода, хранение данных и построение дерева покрытия; включает пять компонентов;
- **UI:** обеспечивает взаимодействие с пользователем через настройку запусков, отображение результатов и аннотирование кода; содержит пять компонентов;

- **Settings**: хранит глобальные и проектные настройки; состоит из двух компонентов.

Внешние элементы включают `llvm-cov`, `llvm-profdata` для генерации данных покрытия и платформу Eclipse для API.

3.1.2. Взаимодействия между пакетами

Пакеты связаны функциональными зависимостями:

- **Core** использует внешние инструменты для генерации отчетов и предоставляет данные UI;
- UI получает данные от core, использует Eclipse API для визуализации и взаимодействует с **Settings** для настройки;
- **Settings** предоставляет настройки и интегрируется с Eclipse.

3.2. Создание конфигурации запуска

На первом этапе разработки была создана конфигурация запуска, представляющая собой набор параметров и настроек, необходимых для выполнения тестов и анализа покрытия кода. Конфигурация позволяет определять проект, цель сборки и файлы для анализа, обеспечивая интеграцию с инструментами LLVM (`llvm-cov`, `llvm-profdata`) для генерации данных покрытия.

3.2.1. Вкладка конфигурации

Важным аспектом реализации являлась вкладка конфигурации, созданная в классе `CoverageTab` на основе платформы Eclipse. Она предоставляет интерфейс для задания параметров и взаимодействия с системой хранения настроек. Основные функции вкладки включают:

- выбор области анализа: обеспечивается возможность указывать проект и цель сборки через диалог, а также выбирать файлы для анализа в таблице, отображающей доступные исходные файлы;

- сохранение параметров: предоставляется возможность сохранять настройки в конфигурации запуска для их использования в последующих запусках;
- использование настроек: реализуется применение глобальных и проектных параметров для корректной работы вкладки.

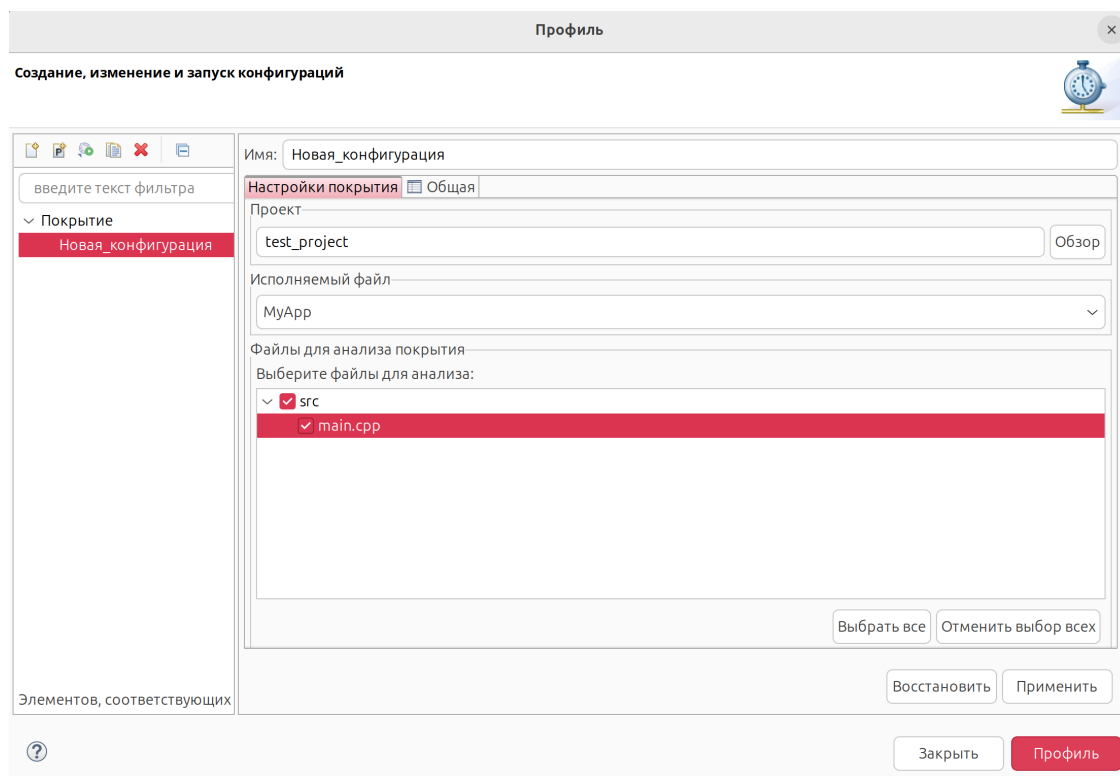


Рис. 3: Вкладка создания конфигурации

3.2.2. Реализация делегата запуска

В процессе создания конфигурации был разработан делегат запуска, реализованный в классе `CoverageLaunchDelegate`. Он отвечает за управление выполнением тестов и обработку результатов покрытия. Делегат выполняет следующие задачи:

1. Определяет пути к ресурсам. Извлекаются данные о проекте, цели сборки и пути к скомпилированному бинарному файлу. При их отсутствии выбрасывается исключение, предотвращающее запуск.

2. Подготавливает окружение. Выполняется сборка проекта с флагами, необходимыми для генерации профильных данных:

```
-fprofile-instr-generate -fcoverage-mapping
```

Эти флаги обеспечивают сбор данных покрытия во время компиляции.

3. Выполняет тесты. Запускаются тесты с установленной переменной окружения, указывающей путь для сохранения сырых данных покрытия:

```
LLVM_PROFILE_FILE=coverage.profraw
```

Переменная создает файл `coverage.profraw` с профильными данными.

4. Обрабатывает профильные данные. Создается промежуточный профиль покрытия с помощью команды:

```
llvm-profdata merge -sparse coverage.profraw -o  
↪ coverage.profdata
```

Команда объединяет сырые данные в файл `coverage.profdata`.

5. Генерирует отчет. Формируется отчет о покрытии с использованием команды:

```
llvm-cov export -instr-profile=coverage.profdata --format=lcov
```

Команда создает отчет в формате `.lcov`, сохраняемый в `coverage_report.lcov`, с применением скомпилированного бинарного файла и промежуточного профиля.

3.2.3. Интеграция с платформой Eclipse

Для интеграции конфигурации запуска с платформой Eclipse использовались точки расширения, которые представляют собой программный механизм, позволяющий регистрировать новые компоненты и расширять функциональность системы без изменения её ядра. Эти

точки, заданные в файле `plugin.xml`, обеспечивают взаимодействие компонентов с платформой:

- `.launchConfigurationTypes` [3]: регистрирует тип конфигурации запуска и связывает его с классом `CoverageLaunchDelegate`, управляющим процессом выполнения;
- `.launchConfigurationTabGroups` [4]: регистрирует группу вкладок, включая `CoverageTab`, для настройки параметров пользователем.

```
<extension point="org.eclipse.debug.core.launchConfigurationTypes">
  <launchconfigurationtype id="su.softcom.cldt.testing.launchConfigType"
    ↪ delegate="su.softcom.cldt.testing.core.CoverageLaunchDelegate"
    ↪ modes="profile" name="%launchConfigurationType.name">
  </launchconfigurationtype>
</extension>
<extension point="org.eclipse.debug.ui.launchConfigurationTabGroups">
  <launchconfigurationtabgroup
    ↪ class="su.softcom.cldt.testing.ui.CoverageTabGroup"
    ↪ id="su.softcom.cldt.testing.tabGroup"
    ↪ type="su.softcom.cldt.testing.launchConfigType">
  </launchconfigurationtabgroup>
</extension>
```

3.3. Отображение результатов покрытия

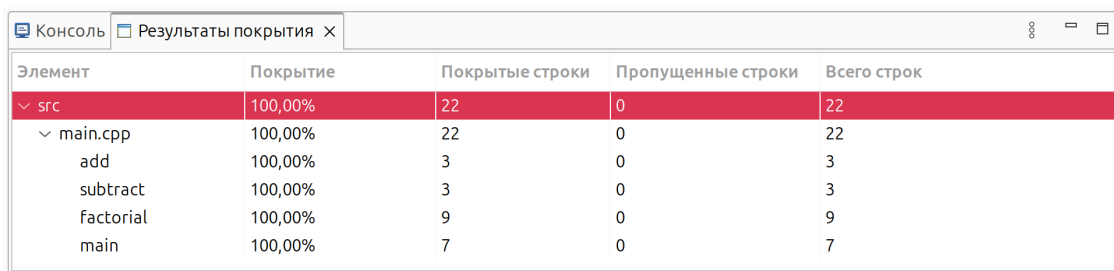
На следующем этапе разработки была реализована система отображения результатов анализа покрытия, обеспечивающая наглядное представление данных в интерфейсе Eclipse. Результаты визуализируются в виде иерархического дерева, что позволяет пользователю удобно анализировать покрытие на уровне папок, файлов и функций.

3.3.1. Вкладка результатов покрытия

Ключевым компонентом стала вкладка результатов, созданная в классе `CoverageResultView` на основе платформы Eclipse. Она предо-

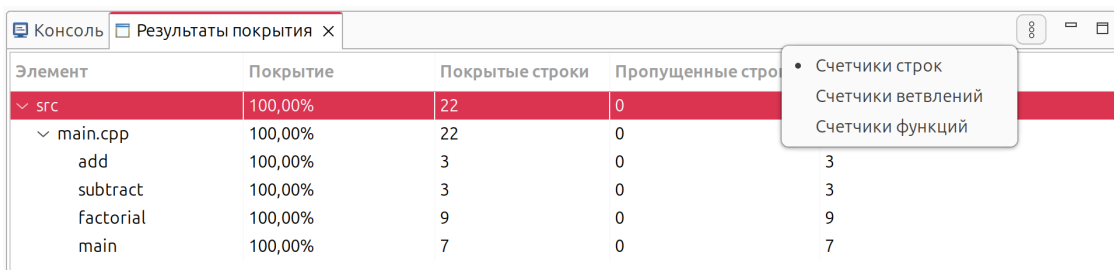
ставляет интерфейс для отображения данных покрытия с использованием компонента **TreeViewer**. Основные функции вкладки включают:

- формирование иерархического дерева: формируется структура данных в виде дерева, отражающая иерархию проекта, с узлами для папок, файлов и функций;
- отображение метрик покрытия: обеспечивается отображение значений выбранных метрик, таких как строки, ветки или функции, для каждого узла дерева;
- переключение метрик: реализуется возможность выбора метрик через контекстное меню, адаптирующее представление данных под аналитические задачи.



Элемент	Покрытие	Покрытые строки	Пропущенные строки	Всего строк
src	100,00%	22	0	22
main.cpp	100,00%	22	0	22
add	100,00%	3	0	3
subtract	100,00%	3	0	3
factorial	100,00%	9	0	9
main	100,00%	7	0	7

Рис. 4: Окно результатов анализа



Элемент	Покрытие	Покрытые строки	Пропущенные строки	Всего строк
src	100,00%	22	0	22
main.cpp	100,00%	22	0	22
add	100,00%	3	0	3
subtract	100,00%	3	0	3
factorial	100,00%	9	0	9
main	100,00%	7	0	7

- Счетчики строк
- Счетчики ветвлений
- Счетчики функций

Рис. 5: Меню выбора метрик

3.3.2. Парсинг отчета LCOV

Для получения данных покрытия используется отчет в формате LCOV, сгенерированный инструментом `llvm-cov`. Отчет содержит информацию о покрытии строк, веток и функций для каждого исходного

файла. Парсинг отчета, реализованный в классе `ReportParser`, позволяет извлечь данные и структурировать их для последующей обработки. Пример отчета LCOV представлен ниже:

```
SF:/home/user/project/src/sample.cpp
FN:10,main
FN:15,_Z3sumii
FNDA:1,main
FNDA:1,_Z3sumii
FNF:2
FNH:2
DA:10,1
DA:11,1
DA:15,1
DA:16,2
DA:17,1
BRDA:16,0,0,1
BRDA:16,0,1,1
BRF:2
BRH:2
LF:5
LH:5
end_of_record
```

Отчет включает следующие элементы:

- **SF**: указывает путь к исходному файлу;
- **FN**: определяет функции и их начальные строки в декорированном формате;
- **FNDA**: отражает выполнение функции, где значение больше нуля указывает на вызов функции;
- **FNF** и **FNH**: показывают общее и выполненное количество функций соответственно;
- **DA**: описывает покрытие строк, включая номер строки и количество выполнений;

- **BRDA**: детализирует покрытие веток, включая номер строки, идентификатор блока, номер ветки и количество выполнений;
- **BRF** и **BRH**: указывают общее и выполненное количество веток соответственно;
- **LF** и **LH**: отражают общее и выполненное количество строк соответственно.

Класс `ReportParser` обрабатывает эти данные, преобразуя их в структуры, пригодные для использования в `CoverageDataManager` и `CoverageDataProcessor`.

3.3.3. Обработка и структурирование данных

Для обеспечения корректного отображения данные покрытия подвергаются предварительной обработке, которая реализуется в классах `CoverageDataManager`, `CoverageDataProcessor` и `CoverageNode`. Процесс включает следующие задачи:

1. Извлекает данные покрытия. Класс `ReportParser` анализирует отчет LCOV, извлекая информацию о покрытии строк, веток и функций, и сохраняет ее в структуры `CoverageDataManager` для дальнейшего использования.
2. Преобразует имена. Класс `CoverageDataProcessor` преобразует декорированные имена функций из отчета в читаемый формат с использованием утилиты `c++filt` и класса `DemangledNameUtils`.
3. Формирует дерево. Класс `CoverageDataProcessor` создает иерархическое дерево узлов `CoverageNode`, где папки и файлы формируются на основе путей файлов, а функции добавляются как дочерние узлы файлов в соответствии с данными отчета.
4. Агрегирует метрики. Класс `CoverageDataProcessor` собирает метрики покрытия (строк, веток и функций) от дочерних узлов к родительским, рассчитывая процент покрытия для каждого узла папки.

3.3.4. Интеграция с платформой Eclipse

Для интеграции вкладки результатов с платформой Eclipse использовалась точка расширения `.views` [7], которая регистрировала представление `CoverageResultView` в интерфейсе Eclipse. Настройка осуществлялась в файле `plugin.xml`:

```
<extension point="org.eclipse.ui.views">
    <view id="su.softcom.cldt.testing.ui.CoverageResultView"
        ↪ name="%view.name"
        ↪ class="su.softcom.cldt.testing.ui.CoverageResultView"
        ↪ category="org.eclipse.ui.category.views">
    </view>
</extension>
```

3.4. Подсветка строк в редакторе

На следующем этапе разработки была создана система подсветки строк, интегрированная в редактор кода Eclipse. Она обеспечивает наглядное отображение статуса покрытия строк, веток и функций в виде цветowych аннотаций и маркеров, размещенных в области кода и боковой панели редактора.

3.4.1. Интеграция подсветки в редактор

Для интеграции с текстовым редактором Eclipse была разработана система аннотаций, реализованная в классе `AnnotationUpdater`. Она предоставляет интерфейс для визуализации данных покрытия с использованием механизмов `IAnnotationModel` и `IMarker`. Основные функции системы включают:

- создание аннотаций: обеспечивается отображение цветowych индикаторов в области кода, отражающих статус покрытия строк, веток или функций;
- формирование маркеров: реализуется создание маркеров в боковой панели редактора, содержащих метаданные о покрытии;

- динамическое обновление: предоставляется возможность обновления аннотаций и маркеров при изменении данных покрытия или открытии новых файлов в редакторе.

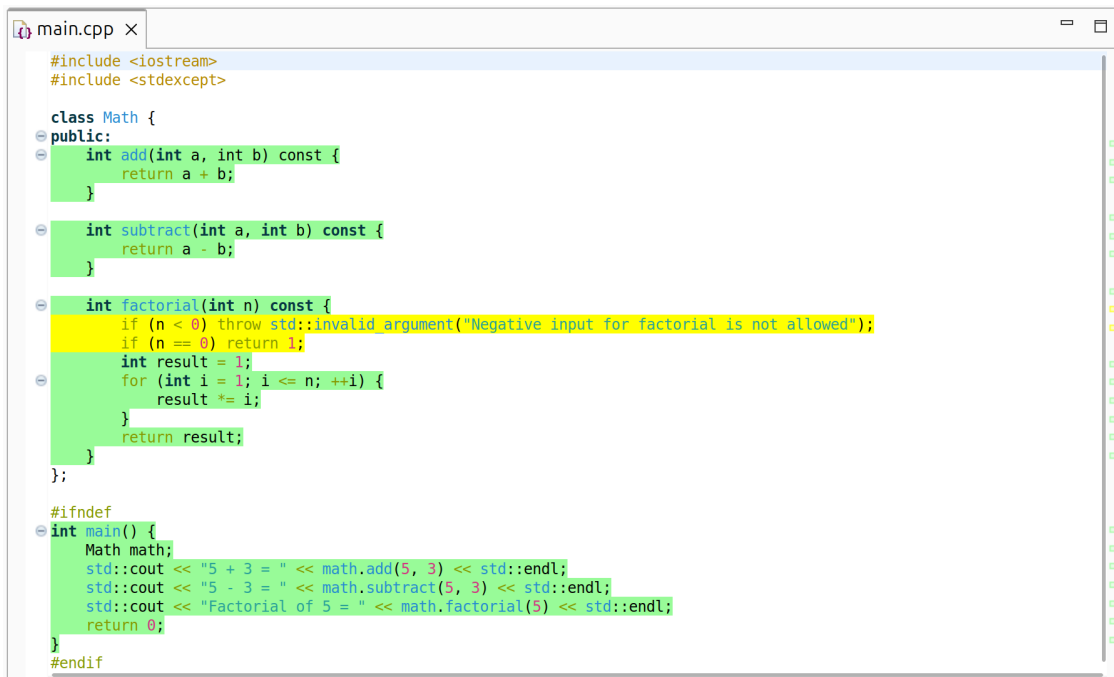


Рис. 6: Подсветка строк в редакторе

3.4.2. Обработка данных для подсветки

Для визуализации покрытия кода данные, хранимые в классе `CoverageDataManager`, обрабатываются с использованием классов `AnnotationUpdater` и `CoverageAnnotation`. Процесс обработки включает следующие задачи:

Обработка аннотаций:

1. Извлекает данные покрытия. Данные о покрытии строк, веток и функций извлекаются из `CoverageDataManager`.
2. Определяет статусы покрытия. Статусы строк, веток и функций (полностью покрыта, непокрыта, частично покрыта) определяются и отображаются зелеными, красными или желтыми индикаторами.

3. Рассчитывает позиции аннотаций. Координаты строк в редакторе определяются для точного размещения аннотаций в соответствии с исходным кодом.
4. Создает аннотации. Аннотации формируются с указанием типа и описания для отображения в редакторе.

Обработка маркеров:

1. Извлекает данные покрытия. Данные о покрытии строк, веток и функций извлекаются из `CoverageDataManager`.
2. Формирует сообщения маркеров. Сообщения для маркеров создаются на основе статусов покрытия (полностью покрыта, непокрыта, частично покрыта).
3. Создает маркеры. Маркеры формируются с метаданными, включающими номер строки, статус покрытия, тип элемента и описание, для отображения на боковой панели.
4. Очищает устаревшие маркеры. Существующие маркеры удаляются перед созданием новых для предотвращения дублирования.

3.5. Создание страницы глобальных настроек

На очередном этапе разработки была создана страница глобальных настроек, интегрированная в интерфейс Eclipse. Она предоставляет удобный способ конфигурации параметров анализа покрытия, применяемых ко всем проектам в рабочем пространстве, обеспечивая централизованное управление поведением плагина.

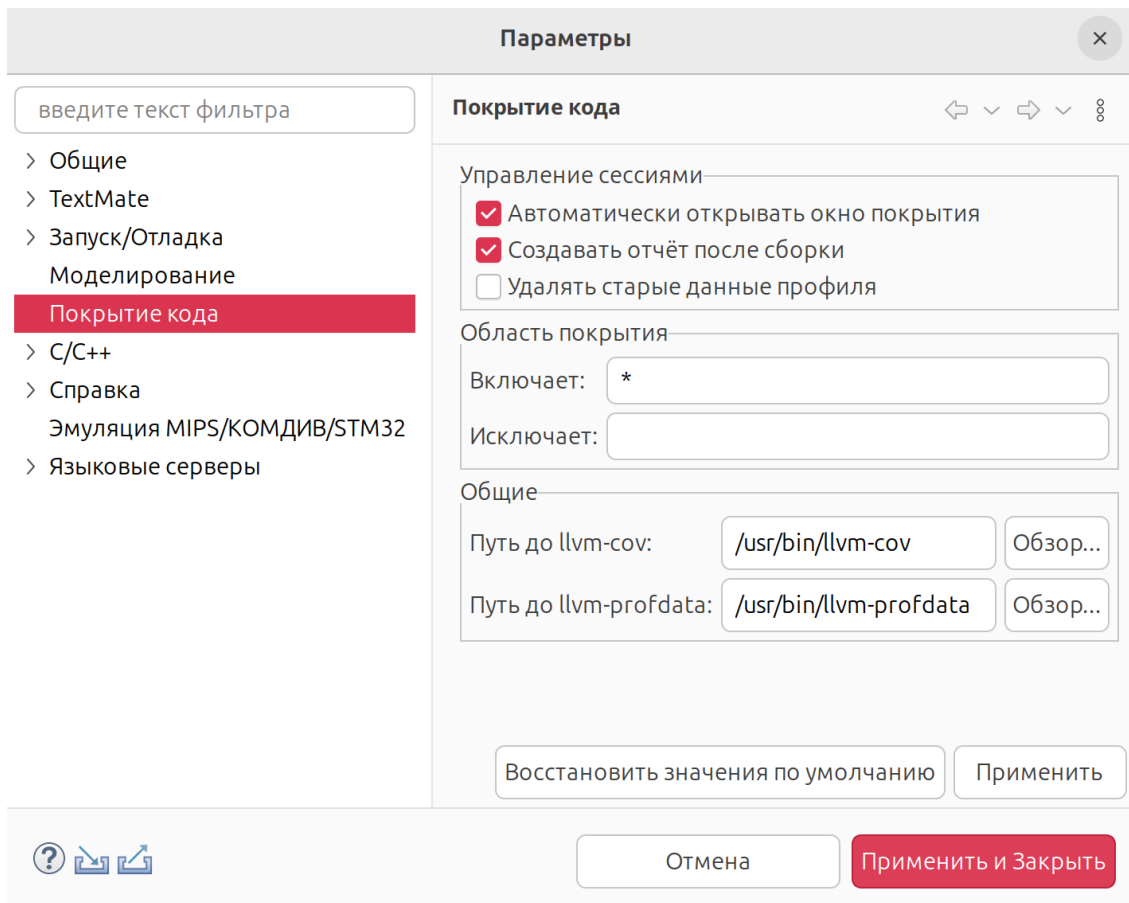


Рис. 7: Страница глобальных настроек

3.5.1. Интерфейс страницы настроек

Для реализации пользовательского интерфейса была разработана страница настроек, реализованная в классе `CoveragePreferencePage` на основе платформы Eclipse. Она предоставляет интерфейс для конфигурации параметров анализа покрытия. Основные функции страницы включают:

- управление сессиями покрытия: обеспечивается настройка поведения сессий анализа;
- фильтрация области анализа: реализуется возможность указания шаблонов для включения или исключения файлов;
- конфигурация инструментов: предоставляется возможность задания путей к исполняемым файлам `llvm-cov` и `llvm-profdata`.

3.5.2. Компоненты интерфейса

Для создания пользовательского интерфейса использовались стандартные компоненты библиотеки SWT. Основные элементы интерфейса включают:

- чекбоксы: переключают параметры с использованием компонента `Button` с типом `SWT.CHECK`;
- текстовые поля: обеспечивают ввод данных через компонент `Text`;
- кнопки выбора файла: позволяют выбирать файлы с помощью компонента `Button`, открывающего `FileDialog`.

3.5.3. Инициализация и хранение настроек

Для настройки параметров на уровне рабочей области используется класс `CoveragePreferenceSettings`, а обработка изменений осуществляется через `CoverageSettingsManager`. Процесс включает следующие задачи:

1. Инициализирует настройки. Загружаются параметры из хранилища `IPreferenceStore` с установкой значений по умолчанию через `CoveragePreferenceSettings`.
2. Формирует интерфейс. Создаются группы настроек с использованием компонентов SWT для управления параметрами рабочей области.
3. Сохраняет параметры. Записываются пользовательские настройки в хранилище `IPreferenceStore` для их применения в последующих сессиях.
4. Уведомляет об изменениях. Обеспечивается динамическое оповещение плагина об изменениях настроек через `CoverageSettingsManager`.

3.5.4. Интеграция с платформой Eclipse

Для интеграции страницы глобальных настроек с платформой Eclipse использовалась точка расширения `.preferencePages` [5], которая позволяет добавлять пользовательские страницы в интерфейс Eclipse. Настройка осуществлялась в файле `plugin.xml`:

```
<extension point="org.eclipse.ui.preferencePages">
  <page id="su.softcom.cldt.testing.ui.CoveragePreferencePage"
    ↪ name="%preferencePage.name"
    ↪ class="su.softcom.cldt.testing.ui.CoveragePreferencePage"
    ↪ category="org.eclipse.ui.preferencePages">
  </page>
</extension>
```

3.6. Создание страницы проектных настроек

На очередном этапе разработки была создана страница проектных настроек, интегрированная в интерфейс Eclipse. Она предоставляет удобный способ конфигурации параметров анализа покрытия, специфичных для отдельного проекта, дополняя глобальные настройки и обеспечивая гибкость в управлении поведением плагина на уровне проекта.

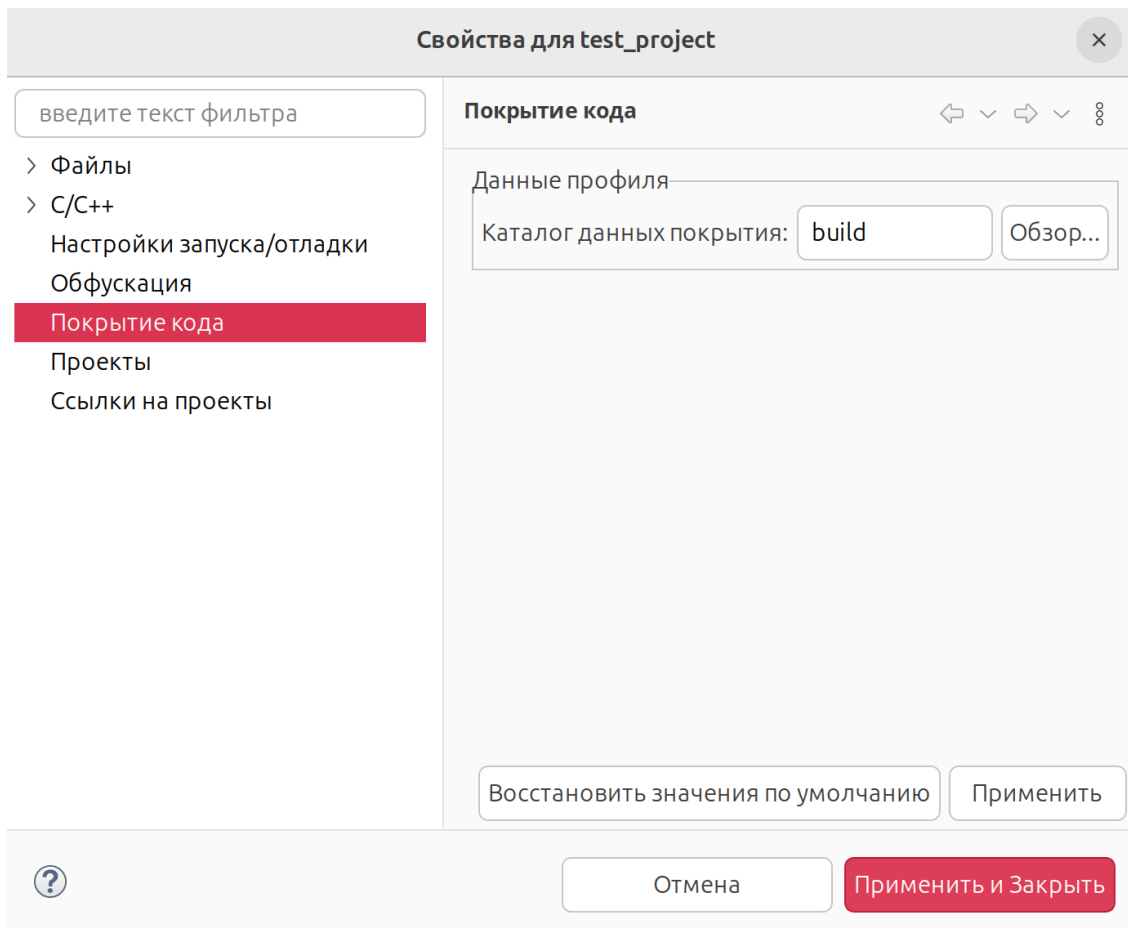


Рис. 8: Страница проектных настроек

3.6.1. Интерфейс страницы настроек

Для реализации пользовательского интерфейса была разработана страница свойств, реализованная в классе `CoveragePropertyPage` на основе платформы Eclipse. Она предоставляет интерфейс для конфигурации параметров анализа покрытия, специфичных для проекта. Основные функции страницы включают:

- настройка директории хранения данных: обеспечивается возможность указания директории для хранения данных покрытия, таких как файлы `coverage.profrw`, `coverage.profddata` и `coverage_report.lcov`;
- фильтрация области анализа: реализуется возможность задания шаблонов для включения или исключения файлов и папок из анализа, совместно с глобальными настройками.

3.6.2. Компоненты интерфейса

Для реализации пользовательского интерфейса использовались стандартные компоненты библиотеки SWT, упомянутые ранее.

3.6.3. Инициализация и хранение настроек

Для настройки параметров на уровне проекта используется класс `CoveragePropertySettings`. Процесс настройки включает следующие задачи:

1. Инициализирует настройки. Загружаются параметры из хранилища `IProject` с использованием `CoveragePropertySettings`, с учетом значений по умолчанию, основанных на директории сборки проекта.
2. Формирует интерфейс. Создается группа настроек с использованием компонентов SWT для управления параметрами проекта.
3. Сохраняет параметры. Записываются пользовательские настройки в проектное хранилище `IProject` с использованием `CoveragePropertySettings` для их применения в последующих сессиях.
4. Уведомляет об изменениях. Обеспечивается динамическое оповещение плагина об изменениях настроек через `CoverageSettingsManager`.

3.6.4. Интеграция с платформой Eclipse

Для интеграции страницы проектных настроек с платформой Eclipse использовалась точка расширения `.propertyPages` [6], которая позволяет добавлять пользовательские страницы в интерфейс Eclipse. Настройка осуществлялась в файле `plugin.xml`:

```
<extension point="org.eclipse.ui.propertyPages">
  <page id="su.softcom.cldt.testing.ui.CoveragePropertyPage"
    ↪ name="%propertyPage.name"
    ↪ class="su.softcom.cldt.testing.ui.CoveragePropertyPage"
    ↪ adaptable="org.eclipse.core.resources.IProject">
  </page>
</extension>
```

4. Эксперимент

4.1. Тестирование

Тестирование проводилось в режиме ручного тестирования на открытых проектах C/C++ с GitHub для проверки отображения результатов анализа покрытия и интеграции плагина с функциональностью платформы Eclipse.

4.1.1. Критерии выбора проектов

Для тестирования были отобраны проекты, удовлетворяющие следующим критериям:

- сборка осуществляется с использованием CMake;
- компиляция выполняется с применением компилятора Clang;
- в проекте имеются тесты, обеспечивающие возможность анализа покрытия.

Выбранные проекты:

- `fmtlib/fmt` [30]: библиотека форматирования строк;
- `nlohmann/json` [23]: библиотека для работы с JSON;
- `fpagliughi/sockpp` [24]: библиотека для работы с сокетами;
- `CrowCpp/Crow` [8]: веб-фреймворк для C++.

4.1.2. Сценарии тестирования

В рамках тестирования реализованы следующие сценарии:

- выполнение анализа покрытия и формирование отчета в формате `.lcov`;
- проверка стабильности работы плагина при обработке больших отчетов `.lcov`;

- оценка корректности отображения иерархического дерева результатов;
- проверка точности расчета метрик покрытия и корректности подсветки строк исходного кода;
- тестирование глобальных и проектных настроек системы анализа покрытия.

4.1.3. Результаты тестирования

Все сценарии тестирования выполнены успешно, подтвердив корректную работу плагина на выбранных проектах и его интеграцию с Eclipse.

4.2. Апробация

Для проведения апробации была организована встреча с сотрудниками, в ходе которой была получена обратная связь по работе плагина. Сотрудники предложили рекомендации по улучшению компонентов пользовательского интерфейса, которые были учтены в процессе доработки плагина. Консультант также подтвердил стабильную работу плагина на закрытых проектах заказчика.

Сотрудники отметили положительный опыт взаимодействия с плагином, выделив интуитивное удобство интерфейса и наглядность отображения результатов анализа покрытия.

Заключение

В рамках данной работы были достигнуты следующие результаты:

1. Проведён обзор существующих плагинов для анализа покрытия кода в Eclipse. Проанализированы решения, такие как LinuxTools GCov, Testwell CTC++ и Parasoft C/C++test, по критериям независимости от CDT, совместимости с компилятором Clang, возможностей настройки анализа и качества визуализации результатов. Установлено, что ни одно из решений не удовлетворяет требованию независимости от CDT, что подтвердило необходимость разработки нового плагина, соответствующего заданным требованиям.
2. Разработан механизм конфигурации запуска анализа покрытия. Реализована поддержка процессов запуска тестирования и генерации отчётов с использованием инструментов LLVM, включая компилятор и утилиты для обработки профильных данных и формирования отчётов в формате LCOV. Использование системы CMake для настройки сборки обеспечило независимость от CDT и упростило интеграцию с процессами разработки.
3. Реализован пользовательский интерфейс, включающий следующие компоненты:
 - отображение результатов анализа в виде иерархического дерева с использованием библиотек SWT и JFace, обеспечивающего наглядное представление метрик покрытия на уровнях папок, файлов и функций;
 - страница глобальных настроек, позволяющая задавать параметры анализа на уровне рабочего пространства, включая настройку поведения сессий анализа покрытия, фильтры для включения или исключения файлов и пути к инструментам LLVM;

- страница проектных настроек, обеспечивающая настройку параметров анализа на уровне проекта, включая директорию для хранения данных покрытия.
4. Реализована подсветка строк и маркеров в редакторе кода Eclipse. Визуализация статуса покрытия выполнена с использованием цветowych индикаторов и маркеров на боковой панели, что позволяет разработчикам быстро выявлять непротестированные участки кода.
 5. Проведено ручное тестирование и апробация разработанного решения. Тестирование выполнено на открытых проектах с GitHub, использующих CMake и Clang, и включало проверку формирования отчётов в формате LCOV, стабильности при обработке больших данных, корректности отображения результатов, точности расчёта метрик и работоспособности настроек. Апробация с участием сотрудников подтвердила интуитивность интерфейса и наглядность визуализации, а на основе их обратной связи были внесены улучшения в пользовательский интерфейс. Консультант также отметил стабильную работу плагина на закрытых проектах заказчика.

Программный код выполненной работы доступен на GitHub ¹.

¹<https://github.com/softcom-su/CLDT-extensions/pull/2>

Список литературы

- [1] Clang: C Language Family Frontend for LLVM. — URL: <https://clang.llvm.org/> (дата обращения: 2025-05-26).
- [2] Corporation IBM. Eclipse CDT Indexer: Source Navigation and Search. — URL: https://help.eclipse.org/latest/topic/org.eclipse.cdt.doc.user/concepts/cdt_c_indexer.htm (дата обращения: 2025-05-26).
- [3] Corporation IBM. Eclipse Launch Configuration Extension Point. — URL: https://help.eclipse.org/latest/topic/org.eclipse.platform.doc.isv/reference/extension-points/org_eclipse_debug_core_launchConfigurationTypes.html (дата обращения: 2025-05-26).
- [4] Corporation IBM. Eclipse Launch Tab Groups Extension Point. — URL: https://help.eclipse.org/latest/topic/org.eclipse.platform.doc.isv/reference/extension-points/org_eclipse_debug_ui_launchConfigurationTabGroups.html (дата обращения: 2025-05-26).
- [5] Corporation IBM. Eclipse Preference Pages Extension Point. — URL: https://help.eclipse.org/latest/topic/org.eclipse.platform.doc.isv/reference/extension-points/org_eclipse_ui_preferencePages.html (дата обращения: 2025-05-26).
- [6] Corporation IBM. Eclipse Property Pages Extension Point. — URL: https://help.eclipse.org/latest/topic/org.eclipse.platform.doc.isv/reference/extension-points/org_eclipse_ui_propertyPages.html (дата обращения: 2025-05-26).
- [7] Corporation IBM. Eclipse Views Extension Point. — URL: https://help.eclipse.org/latest/topic/org.eclipse.platform.doc.isv/reference/extension-points/org_eclipse_ui_views.html (дата обращения: 2025-05-26).

- [8] Crow: C++ Microframework for Web APIs. — URL: <https://github.com/CrowCpp/Crow> (дата обращения: 2025-05-26).
- [9] Eclipse External Tools. — URL: <https://help.eclipse.org/latest/topic/org.eclipse.platform.doc.user/concepts/concepts-exttools.htm> (дата обращения: 2025-05-26).
- [10] Eclipse JFace Action Framework: API Reference. — URL: <https://help.eclipse.org/latest/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/action/package-summary.html> (дата обращения: 2025-05-26).
- [11] Eclipse Project Explorer. — URL: <https://help.eclipse.org/latest/topic/org.eclipse.platform.doc.user/reference/ref-27.htm> (дата обращения: 2025-05-26).
- [12] Foundation Eclipse. Eclipse BIRT: Business Intelligence and Reporting Tools. — URL: <https://eclipse-birt.github.io/birt-website/> (дата обращения: 2025-05-26).
- [13] Foundation Eclipse. Eclipse CDT: C/C++ Development Tooling. — URL: <https://projects.eclipse.org/projects/tools.cdt> (дата обращения: 2025-05-26).
- [14] Foundation Eclipse. Eclipse IDE: Integrated Development Environment. — URL: <https://eclipseide.org/> (дата обращения: 2025-05-26).
- [15] Foundation Eclipse. Eclipse Linux Tools: GCov Integration. — URL: <https://eclipse.dev/linuxtools/projectPages/gcov/index.html> (дата обращения: 2025-05-26).
- [16] Foundation Eclipse. Eclipse PDE: Plug-in Development Environment. — URL: <https://projects.eclipse.org/projects/eclipse.pde> (дата обращения: 2025-05-26).
- [17] Foundation Eclipse. JFace: UI Toolkit for Eclipse. — URL: <https://wiki.eclipse.org/JFace> (дата обращения: 2025-05-26).

- [18] Foundation Eclipse. SWT: Standard Widget Toolkit for Java.— URL: <https://eclipse.dev/eclipse/swt/> (дата обращения: 2025-05-26).
- [19] Foundation Free Software. GCC: The GNU Compiler Collection.— URL: <https://gcc.gnu.org/> (дата обращения: 2025-05-26).
- [20] GCov: Code Coverage Analysis for GCC.— URL: <https://gcc.gnu.org/onlinedocs/gcc/Gcov.html> (дата обращения: 2025-05-26).
- [21] GmbH Verifysoft Technology. Testwell CTC++: Code Coverage Analyzer for C and C++.— URL: https://www.verifysoft.com/en_ctcpp.html (дата обращения: 2025-05-26).
- [22] LLVM Compiler Infrastructure Project.— URL: <https://llvm.org/> (дата обращения: 2025-05-26).
- [23] Lohmann Niels. nlohmann/json: JSON Library for Modern C++.— URL: <https://github.com/nlohmann/json> (дата обращения: 2025-05-26).
- [24] Pagliughi Frank. sockpp: Modern C++ Socket Library.— URL: <https://github.com/fpagliughi/sockpp> (дата обращения: 2025-05-26).
- [25] Parasoft. Parasoft C/C++test: Static Analysis and Unit Testing.— URL: <https://www.parasoft.com/products/parasoft-c-ctest/> (дата обращения: 2025-05-26).
- [26] Project LLVM. llvm-cov: Code Coverage Tool for LLVM.— URL: <https://llvm.org/docs/CommandGuide/llvm-cov.html> (дата обращения: 2025-05-26).
- [27] Project LLVM. llvm-profdata: Profile Data Tool for LLVM.— URL: <https://llvm.org/docs/CommandGuide/llvm-profdata.html> (дата обращения: 2025-05-26).

- [28] Sennikovsky Mikhail. CDT Managed Build System. — URL: <https://www.eclipse.org/events/2005/langsymp/cdt-managed-build-system.pdf> (дата обращения: 2025-05-26).
- [29] Systems QNX Software. Eclipse CDT: Project Build Settings. — URL: https://help.eclipse.org/latest/topic/org.eclipse.cdt.doc.user/tasks/cdt_t_proj_build_set.htm (дата обращения: 2025-05-26).
- [30] fmt: Modern C++ Formatting Library. — URL: <https://github.com/fmtlib/fmt> (дата обращения: 2025-05-26).
- [31] kitware. CMake: Cross-Platform Build System. — URL: <https://cmake.org/> (дата обращения: 2025-05-26).